Functional Programming and Erratic Non-Determinism

Corin Steven Pitcher

Trinity College

June 3, 2001

Submitted in partial fulfilment of the requirements for the Doctor of Philosophy in Computation

Oxford University Computing Laboratory

Programming Research Group

Abstract

Non-deterministic programs can represent specifications, and non-determinism arises naturally in concurrent programming languages. In this dissertation, λ-calculi exhibiting erratic non-determinism are studied in order to identify definitions and techniques that may be applicable to higher-order programming languages for specification or concurrency.

The non-deterministic λ-calculi arise as fragments of an infinitary, non-deterministic λ-calculus $\mathscr{L}$ with countably indexed erratic choice. The operational semantics for $\mathscr{L}$ induces a uniform operational semantics upon each fragment, facilitating arguments that apply to different non-deterministic λ-calculi.

The behaviour of programs in each fragment is abstracted to a form of labelled transition system with divergence called a typed transition system. Several applicative similarity and bisimilarity relations are defined upon the states of each typed transition system, including the fragments. Examples that distinguish the relations are constructed in a simple typed transition system $\mathscr{S}$ and are later shown to have analogues in non-deterministic λ-calculi. Maps that preserve and reflect the structure of typed transition systems are investigated because they reflect the finest relation, convex bisimilarity, and it is proven that there is a map to $\mathscr{S}$ from every typed transition system satisfying a mild condition.

Using operational techniques, the lower, upper, and convex variants of similarity are shown to be compatible and to satisfy the Scott induction principle for every fragment. In addition, the other relations are compatible for a useful collection of fragments. Relative definability of non-deterministic programs is considered with respect to convex bisimilarity, and a chain of fragments is presented for which the corresponding chain of convex bisimilarity relations are related by strict inclusions, i.e., more expressive forms of erratic non-determinism distinguish terms that cannot be distinguished by less expressive forms of erratic non-determinism.

Contents

List of Figures

Acknowledgements

Firstly, I would like to thank my supervisor Luke Ong for his advice and encouragement throughout the course of my studies. His advice has proven invaluable. I am also grateful to my second supervisor Lincoln Wallen.

I have had profitable discussions on the topic of non-deterministic λ-calculi with Søren Lassen, Andrew Moran, and Russell Harmer. I have had enjoyable discussions on other topics with Corina Cîrstea, Thomas Hildebrandt, Dominic Hughes, Ranko Lazić, Ralph Loader, Michael Marz, Guy McCusker, Richard McPhee, Julian Rathke, Charles Stewart, and James Ben Worrell.

Andrew Gordon arranged an internship at Microsoft Research, Cambridge that I enjoyed a great deal.

The faculty at CTI, DePaul University have been very supportive during the final stages of writing this dissertation, and, in particular, I thank Alan Jeffrey and Marcus Schaefer for their comments.

My parents, sister, and grandparents have unfailingly provided, sometimes undeserved, love, support, and understanding.

Finally, I would like to thank Jessalynn. Her love and support make life worthwhile.

Chapter 1

Introduction

1.1 Motivation

Non-determinism is frequently encountered, or introduced deliberately, when reasoning about specifications or concurrent systems. For example, some program development formalisms identify specifications with non-deterministic systems; and concurrent systems can exhibit non-determinism when timing or scheduling information is unavailable or difficult to calculate. A non-deterministic system is generally easier to work with than a collection of systems that implement a specification, or a collection of outcomes of a concurrent system under all possible timing or scheduling behaviours.

We now examine several representative scenarios involving a non-deterministic system and reasonable implementations. Suppose that M and N denote state-less, deterministic systems that accept a natural number and return another natural number or fail to terminate. The time and space properties of M and N are left unspecified. When they exist, we write $M(x)$ and $N(x)$ for the results of passing the natural number x as an argument to the systems M and N respectively.

Consider a non-deterministic system $M \cup N$ that accepts a natural number and then behaves either as $M(x)$ or as $N(x)$. The system $M \cup N$ could be *implemented* or *refined* by any of the following systems:

System A is M.

System B is N.

System C reads one digit from an unbounded tape of binary digits, winds the tape on, and then behaves as M if the digit is 0 and as N if the digit is 1. Nothing further is assumed about the tape.

System D behaves as M with probability 0.75 and as N otherwise.

System E accepts a natural number x, forwards x to systems A and B, and returns the first result that it receives. Nothing is assumed about the relative performance of the systems A and B or the connections to them.

System F accepts a natural number x, runs M and N on a sequential, multi-tasking system, and returns the first result that it finds. Nothing is assumed about the scheduling algorithm in use.

The relationship between M and systems A and B is typical of program development formalisms, but $M \cup N$ is an atypical specification because of its simplicity. More often, the specification (a non-deterministic system) is a term of an expressive language, and it is challenging to find implementations.

Systems C, D, E, and F are non-deterministic because of incomplete information, but each one can be refined to different deterministic systems when additional information is available. For example, the behaviour of system C is determined by the contents of the tape, and system D may be found to base its decision upon an internal counter that chooses N when the counter is divisible by 4. In both cases, a state-less, non-deterministic system $M \cup N$ is refined by stateful, deterministic systems.

Pursuing system C leads to a strategy for modelling non-determinism using oracles, where the tape is an oracle that can be called upon to resolve non-deterministic choices. The probabilities associated with the behaviours of system D can be useful, in particular for showing that the probability of an undesirable behaviour is small. Neither oracles nor probabilistic systems are considered in this dissertation.

System E executes $M(x)$ and $N(x)$ concurrently. In general, the outcome depends upon the relative performance of the component systems and connections. Without this information, we must consider all possible performance characteristics. However, it is possible to describe the non-deterministic behaviour of system E without reference to performance characteristics, and this facilitates reasoning about such concurrent systems.

There is a subtlety in system E and its permissible implementations. Suppose that x is passed to the systems A and B, but that only system A terminates, with result $M(x)$. In this case, system E will always terminate with result $M(x)$, thus avoiding the non-termination of system B. In contrast, system C may read 1 from the tape and then fail to terminate because it is committed to behaving as N.

System F is similar to system E. The description of the system is incomplete without the scheduling algorithm. In particular, it is not known whether the scheduling algorithm is *fair*: are the computations of M and N interleaved, or dove-tailed, so that a non-zero, finite number of steps are carried out before the scheduler switches to the other computation? For example, a scheduling algorithm that never schedules N is not fair. If the scheduling algorithm is known to be fair, then system F also has the property that it can avoid non-termination of one of the computations, and so cannot be distinguished from system E.

The non-termination avoidance property of systems E and F is relevant to many real systems because multi-tasking operating systems with fair scheduling algorithms are the norm for current desktop and server computers (and are becoming more common in embedded systems). When termination properties are important, it is useful to identify different ways of combining systems as a single non-deterministic system. For example, the *erratic choice* $M \cup N$ does not have the non-termination avoidance property because it can be implemented by system C. On the other hand, the *ambiguous choice* of M and N is effectively the same as system E, or system F with a fair scheduling algorithm, and thus does have the non-termination avoidance property. The ambiguous choice of M and N is a valid implementation or refinement of the erratic choice $M \cup N$.

In summary, non-determinism is used to package a family of systems as a single system to provide an abstraction from implementation details, and hence facilitate reasoning.

1.2 Related Work

There is a considerable body of research involving non-determinism. This section describes some of the relevant literature according to the categories: non-deterministic imperative programming languages, process calculi, and non-deterministic and concurrent functional programming languages.

1.2.1 Non-Deterministic Imperative Programming Languages

Hoare's seminal paper [Hoa69] introduces a formal method for proving the correctness of programs of a simple deterministic imperative programming language (see [Gri81] for a historical account). The programs of the language are composed of: assignment statements (to integer-valued variables), sequential composition, conditionals, and while loops. The expressions of the language are expressions of the underlying logic. Programs are annotated with predicates about the state of the machine. A program is correct if the annotations are consistent with the behaviour of the statements.

A *Hoare triple* $\{P\}\,S\,\{Q\}$ consists of a statement S, a predicate P called the *precondition*, and a predicate Q called the *postcondition*. The Hoare triple $\{P\}\,S\,\{Q\}$ asserts that, when started in a state that satisfies the predicate P, the statement S will either fail to terminate or will terminate in a state that satisfies the predicate Q. The judgements of the proof system are Hoare triples. For example, the axiom for assigning the value of the formula E to the variable x is:

$$\{P[E/x]\}\,x := E\,\{P\}$$

Compound statements also have proof rules. For example, the proof rule for the sequential composition of statements S_1 and S_2 is (where P, Q, and R are predicates):

$$\frac{\{P\}\,S_1\,\{Q\} \qquad \{Q\}\,S_2\,\{R\}}{\{P\}\,S_1; S_2\,\{R\}}$$

The while loop is written as $\mathsf{do}\,P \to S\,\mathsf{od}$ where the predicate P, called a *guard*, is the condition that is tested before each loop iteration and S is the body of the loop. The corresponding proof rule is (where I is a predicate):

$$\frac{\{I \wedge P\}\,S\,\{I\}}{\{I\}\,\mathsf{do}\,P \to S\,\mathsf{od}\,\{I \wedge \neg P\}}$$

The proof rule for while loops depends upon the idea of an *invariant* predicate I that always holds before the condition is tested, i.e., it holds before the while loop starts and after each iteration. The proof system establishes properties of the final state of a program but does not address termination. This can be seen in the proof rule for the while loop which allows a proof of the following statement even though no program can terminate in a state that satisfies false:

$$\{true\}\,\mathsf{do}\,true \rightarrow x := x\,\mathsf{od}\,\{false\}$$

Hoare's original system can be used for *program verification*, checking that an existing program satisfies a specification. This raises the question of how the program and the verification are obtained. For example, a developer may write a program, and pass it to an expert that verifies it. If the expert finds an error, then the program may have to be returned to the developer and the cycle repeated.

Dijkstra [Dij76] pioneers an approach to *program synthesis* where the program and the proof of correctness are developed hand in hand. The developer starts from a formal specification and applies transformations to it until a program is reached. The transformations are chosen so that the resulting program must be a valid implementation of the specification. The sequence of transformations documents the design and the proof of correctness of the program.

In Dijkstra's framework, specifications and programs are elements of the same space. Dijkstra uses the space of predicate transformers, but other spaces have been considered in the literature because the choice affects the style and ease of program development. Accounts of the spaces and the relationships between them can be found in [dB80, Gru93]. In contrast to Hoare's proof system, the predicate transformer framework does take account of termination and so there is an additional proof requirement for iteration or recursion.

Not all predicate transformers are programs. Dijkstra introduces a *language of guarded commands* (often called the *guarded command language*), and gives a denotational semantics for guarded command language programs as predicate transformers. A predicate transformer can be implemented if it is the image of a guarded command language program and the predicates used as guards are computable.

It is straightforward to define a non-deterministic operational semantics for the guarded command language. The non-determinism arises from the general constructs for *alternation*:

$$\mathsf{if}\,P_1 \rightarrow S_1 \,[\!]\, P_2 \rightarrow S_2 \,[\!]\, \ldots \,[\!]\, P_n \rightarrow S_n\,\mathsf{fi}$$

and *iteration*:

$$\mathsf{do}\,P_1 \rightarrow S_1 \,[\!]\, P_2 \rightarrow S_2 \,[\!]\, \ldots \,[\!]\, P_n \rightarrow S_n\,\mathsf{od}$$

The guards $P_1, P_2, \ldots, P_n$ are predicates. Informally, if the thread of execution reaches an alternation or iteration statement and one of the guards is found to be true in that state, then the corresponding branch is executed. If none of the guards are true, then the behaviour of the alternation statement is undefined (often identified with non-termination). If none of the guards are true for an iteration statement, then control passes to the following statement. Alternation and iteration statements are non-deterministic when two or more guards are true, because the order in which guards are tested is not specified. For example, b may contain either 0 or 1 after the following program is executed:

$$\begin{array}{ll} \mathsf{if} & \mathit{true} \rightarrow b := 0 \\ [] & \mathit{true} \rightarrow b := 1 \\ \mathsf{fi} & \end{array}$$

The standard semantics of the alternation and iteration statements is not *fair*. For example, consider the following non-deterministic program:

$$\begin{array}{ll} b := 0; & \\ n := 0; & \\ \mathsf{do} & b = 0 \rightarrow n := n+1 \\ [] & b = 0 \rightarrow b := 1 \\ \mathsf{od} & \end{array}$$

The while loop terminates when the second branch is chosen and b is assigned 1. If it terminates, we know that b contains 1 and that n contains a natural number. However, the while loop will not terminate if the first branch is always chosen. In the standard semantics, any of the following programs would be valid implementations or refinements of the above program:

- $b := 1; n := 0$
- $b := 1; n := 0; \mathsf{do}\, n < 10 \rightarrow n := n+1\, \mathsf{od}$
- $\mathsf{do}\, \mathit{true} \rightarrow n := n\, \mathsf{od}$

The standard operational semantics for the guarded command language is not fair because a branch may not ever be chosen for execution even though its associated guard is true infinitely often. In the example above, the second guard is true whenever the first guard is true. Thus, in a fair setting, the second branch must be chosen eventually, and so the while loop always terminates and n may contain any natural number. Dijkstra rejects the requirement that implementations of the alternation and iteration statements be fair because fairness greatly complicates the semantics but is rarely required to prove a program correct. Francez [Fra86] and Apt and Olderog [AO91] consider fairness in the context of the guarded command language.

Assuming the standard (unfair) semantics, if a guarded command language program may assign any natural number to a variable, then it may fail to terminate (see also lemma 3.4.8(2)). More generally, suppose that we have a non-deterministic guarded command language program. For each initial state, if the program always terminates when started from that state, consider the cardinality of the set of final states that are reachable from that state. If all of the cardinalities are finite, then the program exhibits *finite non-determinism* (or *bounded non-determinism*). If any of the cardinalities are ω, then the program exhibits *countable non-determinism*. We may then state: with the standard operational semantics, no guarded command language program exhibits countable non-determinism.

The definitions of finite non-determinism and countable non-determinism have analogues for predicate transformers. Dijkstra proposes that predicate transformers reflect the behaviour of guarded command language programs and places a restriction upon predicate transformers that prohibits countable non-determinism (p77 of [Dij76]):

> The second reason for reassurance is of a rather different nature. A mechanism of unbounded nondeterminacy yet guaranteed to terminate would be able to make within a finite time a choice out of infinitely many possibilities: if such a mechanism could be formulated in our programming language, that very fact would present an insurmountable barrier to the possibility of the implementation of that programming language.

However, it is sometimes convenient to work with specifications that exhibit countable non-determinism and are subsequently refined to programs. In addition, if countable non-determinism is banned then every specification must be checked for compliance. For these reasons, much of the subsequent research [Bac80, Gri81, Bac88, Kal90, Mor90] allows predicate transformers to exhibit countable non-determinism. Researchers in other areas have also rejected the restriction to finite non-determinism [Par79, Par81, MW95].

This creates an occasionally awkward mismatch between the expressiveness of predicate transformers that can exhibit countable non-determinism and guarded command language programs that cannot. The mismatch can be resolved by extending the guarded command language and operational semantics with new statements that exhibit countable non-determinism. It is straightforward to do this without fair alternation or iteration. For example, the guarded command language can be extended with a statement that always terminates and may assign any natural number to the variable x:

$$x :\in \omega$$

This program has the same behaviour as the program above with a fair semantics.

Similar imperative programming languages that exhibit countable non-determinism are studied in [Bac80, Plo82, AP86, AO91, dGHLP94]. Countable non-determinism turns out to be useful for studying fair operational semantics, because it is possible to define a fair scheduler in terms of countable non-determinism. Using fair schedulers, a syntactic translation can be defined from the guarded command language to the guarded command language extended with $x :\in \omega$. The important property is that the translation of a program with the unfair operational semantics has the same behaviour as the original program with the fair operational semantics. For example, consider execution of the following program with the fair operational semantics:

$$\begin{array}{ll}
\mathsf{do} & P \to S_1 \\
[\!] & P \to S_2 \\
\mathsf{od} &
\end{array}$$

If we choose variables b and n that do not appear in the above program, then it can be rewritten using a scheduler that enforces fairness:

$$\begin{array}{ll}
\multicolumn{2}{l}{b :\in \omega;} \\
\multicolumn{2}{l}{n :\in \omega;} \\
\mathsf{do} & P \wedge b = 0 \wedge n \neq 0 \to S_1; n := n-1 \\
[\!] & P \wedge b \neq 0 \wedge n \neq 0 \to S_2; n := n-1 \\
[\!] & P \wedge b = 0 \wedge n = 0 \to b := 1; n :\in \omega \\
[\!] & P \wedge b \neq 0 \wedge n = 0 \to b := 0; n :\in \omega \\
\mathsf{od} &
\end{array}$$

The variables b and n store the state of the scheduler. The variable b indicates whether the first or second branch is being executed and n contains the number of iterations of that branch that will be taken before the scheduler switches to the other branch. The third and fourth branches of the iteration are taken when the counter n reaches 0 and switch the scheduler from the first branch to the second branch or vice-versa. The rewritten program cannot exclude the first or the second branch infinitely often, even when executed with the unfair operational semantics. Thus fairness of iteration is reduced to countable non-determinism by a syntactic translation in the same way that parallel-or can be implemented upon a sequential, multi-tasking system.

1.2.2 Process Calculi

This section highlights some of the common operational techniques used to define and reason about process calculi such as CSP [Hoa85, Ros98], CCS [Mil89], and the π-calculus [Mil91, MPW92]. Non-determinism plays an important role in these calculi because they are typically given an *interleaving* semantics (see [WN95]), i.e., concurrency is modelled using non-determinism.

CSP, CCS, and the π-calculus can describe erratic non-deterministic choice between two processes P and Q using the notation $P \sqcap Q$ (CSP) or $\tau.P + \tau.Q$ (CCS and the π-calculus). These processes evolve silently into either P or Q. In addition, CSP and CCS permit the erratic non-deterministic choice of an infinite family of processes. If I is a set and, for each $i \in I$, P_i is a process, then:

$$\prod_{i \in I} P_i \qquad \text{and} \qquad \sum_{i \in I} \tau.P_i$$

denote erratic non-deterministic choice between the processes of the $\{P_i \mid i \in I\}$ in CSP and CCS respectively.

The process calculi are each given an operational semantics in the form of a labelled transition relation between processes represented as terms. The labelled transition relation is defined by induction upon the structure of terms following Plotkin [Plo81]. A transition, labelled with α, between processes P and Q indicates that P may evolve to Q and is written $P \xrightarrow{\alpha} Q$. The labels vary according to the calculus, but, in general, record an (atomic) interaction of the left-hand process with its environment. A distinguished label τ is used to record an internal interaction that cannot be seen or influenced by an external party. For example, the operational semantics determines that the behaviour of the non-deterministic choice processes given above is:

- For CSP: $P \sqcap Q \xrightarrow{\tau} P$ and $P \sqcap Q \xrightarrow{\tau} Q$
- For CCS and the π-calculus: $\tau.P + \tau.Q \xrightarrow{\tau} P$ and $\tau.P + \tau.Q \xrightarrow{\tau} P$

The labelled transition relation is the basis for reasoning about a process. The branching behaviour of a process is an important component of the behaviour of a process, and is readily apparent from the labelled transition relation. A process P branches when there is more than one pair consisting of a label α and a process Q such that $P \xrightarrow{\alpha} Q$. The non-determinism in the examples above demonstrates branching. Branching also occurs in the guarded command language when there is more than one terminal state for a program with a fixed initial state because of non-deterministic alternation or iteration statements. For example, running the following program results in one of two states, in one b contains 0, and in the other b contains 1:

$$\begin{array}{ll} \mathsf{if} & \mathit{true} \rightarrow b := 0 \\ [] & \mathit{true} \rightarrow b := 1 \\ \mathsf{fi} & \end{array}$$

The intermediate states of such a program's execution are nearly always ignored, and so guarded command language programs have a simple branching structure: they branch immediately to their final states. In contrast, it is often important to know whether a process branches before or after an interaction with the environment, and this is captured by the labelled transition relation. For example, consider the CCS processes:

$$a.b.0 + a.c.0 \qquad \text{and} \qquad a.(b.0 + c.0)$$

The first process performs an a-labelled transition to one of the non-branching, deterministic processes $b.0$ or $c.0$. The second process performs an a-labelled transition to the branching process $b.0 + c.0$, which can in turn perform either a b-labelled transition or a c-labelled transition to the nil process 0. The decision as to whether b or c will occur is fixed once a has occurred in the first process, but is not fixed in the second process. Of course, the branching structure may or may not be required depending upon the application, but it is nonetheless available in the transition relation (see [Van94] for a good discussion).

The terms and labelled transition relation of each process calculus form a *labelled transition system* (LTS). An LTS consists of a set of states (the processes or terms), a set of labels, and a labelled transition relation. A process is a term, which is in turn a state of the LTS.

Definitions of semantic equivalence processes are often phrased as equivalences upon the states of LTSs. The best known equivalence is *strong bisimilarity* $\simeq$ (see [Par79, Par81, MT88, Mil89, Abr91, Ros98]). If states s and t are related by $\simeq$, then every transition from s must be matched by a transition from t and the two target states must also be related, and vice-versa:

$$\begin{aligned} s \simeq t \Longrightarrow\ & (\forall \alpha, s'. s \xrightarrow{\alpha} s' \Longrightarrow \exists t'. t \xrightarrow{\alpha} t' \wedge s' \simeq t') \wedge \\ & (\forall \alpha, t'. t \xrightarrow{\alpha} t' \Longrightarrow \exists s'. s \xrightarrow{\alpha} s' \wedge s' \simeq t') \end{aligned}$$

However, this property is not sufficient to define strong bisimilarity, and so $\simeq$ is defined to the greatest relation satisfying the above. This is a coinductive definition. It is necessary to use coinduction rather than induction to define $\simeq$ because processes may have infinite sequences of transitions due to cyclic definitions. Strong bisimilarity ensures that states (processes) have precisely the same pattern of interaction with their environments, as given by the labelled transition relation. Many variants of strong bisimilarity have been proposed for different applications, see [Van90, Van93].

It is easier to reason about a system built from components if some of the properties of the composite system can be inferred from known properties of the components. The *compatibility* property of strong bisimilarity allows us to infer that two processes built using the same term constructor are related by strong bisimilarity whenever the immediate sub-terms of the processes are related by strong bisimilarity. A *compatible* equivalence relation is known as a *congruence*. Not all variants of strong bisimilarity are compatible, e.g., *weak bisimilarity* (see [Mil89]).

In related work, Aczel and others [Acz88, Acz94, FHL94, BM96] use a similar definition to construct models of *non-well-founded set theory* by quotienting the class of unlabelled transition systems with a variant of strong bisimilarity, thus establishing a connection between cyclic processes and non-well-founded sets. In addition, the connection between the coinductive definitions of the variants of bisimilarity and certain kinds of *games* with *winning strategies* (see [Acz77]) is applied to model-checking by Stirling [Sti97]. However, such games are not directly comparable with the games used to give semantics to programs in [AJM94, HO00] because they represent the proofs of bisimilarity as opposed to the processes.

1.2.3 Functional Programming, Non-Determinism, and Concurrency

Deterministic λ-Calculi

λ-calculi (see [Bar84]) provide a clean basis for reasoning about functional programming languages. *Contextual equivalence* is an appealing behavioural equality relation that can be defined upon the programs of the pure untyped λ-calculus with a reduction strategy [Abr90], PCF [Sco93, Plo75, Plo77, Plo81, Kah87, Pit97], or FPC [Plo85, Gun92]. A program M is related by contextual equivalence to another program N with the same type as M if, for all (suitable) *contexts* $\mathcal{C}[-]$, $\mathcal{C}[M]$ terminates if and only if $\mathcal{C}[N]$ terminates. The definition of contextual equivalence captures the intuition that two programs should be considered the same if they are interchangeable (with respect to observing termination using tests written in the same programming language). For PCF and FPC, there is some leeway in the definition because the contexts may be restricted to *ground contexts*, where $C[M]$ and $C[N]$ must have a ground or base type such as the type of booleans or natural numbers. This determines whether or not the divergent program Ω is identified with the function $\lambda x.\Omega$ that accepts an argument and then diverges.

Unfortunately, it is hard to work directly with contextual equivalence because of the quantification over contexts. In the case of PCF, it is possible to exploit the simple type structure to find a more convenient characterisation of contextual equivalence. An equivalence $\simeq$ is defined upon programs as a *logical relation* (see [Mit96]) by induction on the structure of types. For programs M and N of the same type ρ:

- If ρ is the natural number type, $M \simeq N$ if and only if, for all natural numbers n, M evaluates to $\underline{n}$ if and only if N evaluates to $\underline{n}$. The term $\underline{n}$ represents the natural number n.
- If ρ is the function type $\sigma \to \tau$, then $M \simeq N$ if and only if, for all programs M' and N' of type σ, $M' \simeq N'$ implies $M\,M' \simeq N\,N'$.

It can be shown that $\simeq$ coincides with contextual equivalence for PCF. This is useful because $\simeq$ uses a more restrictive quantification than the definition of contextual equivalence, and thus is easier to prove. Essentially, *applicative contexts* suffice for testing terms of function type, and contexts of the following form suffice for testing terms of natural number type:

$$\mathsf{if}\,\mathsf{eq}\,(-,\underline{n})\,\mathsf{then}\,\underline{0}\,\mathsf{else}\,\Omega$$

A similar definition and result can be established for the *contextual preorder*, where a program M is related by the contextual preorder to another program N with the same type as M if, for all (suitable) *contexts* $\mathcal{C}[-]$, $\mathcal{C}[M]$ terminates implies that $\mathcal{C}[N]$ terminates.

In order for $\simeq$ to coincide with contextual equivalence when non-ground contexts are permitted, it is necessary to add an additional clause to the second case stating that M terminates if and only if N terminates.

Abramsky [Abr90, AO93] proposes a study of pure untyped λ-calculus with the call-by-name reduction strategy. However, the strategy outlined above for PCF defines $\simeq$ at a function type $\sigma \to \tau$ in terms of $\simeq$ at the types σ and τ, and the use of $\simeq$ at σ is contravariant. It is not possible to make a similar definition for the untyped λ-calculus or FPC because an inductive definition fails at recursive types, and the contravariance prevents a fixed-point definition via a monotone function.

Abramsky's solution is to define an *applicative similarity*[1] relation, also using applicative contexts, and then show that it coincides with the contextual preorder using a domain-theoretic technique. If programs M and N are related by applicative similarity $\lesssim$, then for every term M' such that M evaluates to $\lambda x.M'$, there must be a term N' such that N evaluates to $\lambda x.N'$ and for every program L, the terms $M'[L/x]$ and $N'[L/x]$ are related by applicative similarity, i.e.:

$$M \lesssim N \Longrightarrow \forall M'.\, M \Downarrow \lambda x.M' \Longrightarrow \exists N'.\, N \Downarrow \lambda x.N' \wedge \forall L.\, M'[L/x] \lesssim N'[L/x]$$

As with strong bisimilarity for processes, there are many relations that satisfy this property (including the empty relation). Applicative similarity is the greatest relation that satisfies this property. Intuitively, applicative similarity is defined by coinduction upon the implicit recursive type structure of the untyped λ-calculus (as demonstrated by a translation of the untyped λ-calculus into FPC, see [Gun92]).

Applicative similarity also uses applicative contexts as tests, although it does so using a substitution rather than a real application. In addition, there is a convergence test before substitution. The convergence and application tests can be incorporated into an LTS with programs as states. There is a transition $M \xrightarrow{L} M'[L/x]$ between programs M and $M'[L/x]$, labelled with the program L, if and only M evaluates to $\lambda x.M'$. Using this LTS, applicative similarity can be seen as a preorder variant of strong bisimilarity. Conceptually, it is also useful to consider the LTS unfolded into an infinite tree that describes the behaviour of a term (cf. Böhm trees for the classical λ-calculus, see [Bar84, JR97, AC98]).

There is an important difference between the applicative contexts used for the alternative characterisation of contextual equivalence for PCF and those used for applicative similarity. The

[1] Abramsky uses *applicative bisimilarity* and not *applicative similarity*. We use the latter, for consistency with definitions in the sequel.

former are pairs of contexts $(-M')\ (-N')$ such that M' and N' are programs related by $\simeq$, whereas the latter uses a single context $(-L)$ twice, where L is a program. This modification makes the coinductive definition of applicative similarity possible, but makes it considerably harder to establish that it is a *compatible* relation. In particular, we need to know that for all programs M, M', N, N':

$$(M \lesssim N \wedge M' \lesssim N') \Longrightarrow M\,M' \lesssim N\,N'$$

Abramsky [Abr90] establishes that applicative similarity is a compatible preorder (precongruence) using domain-theoretic techniques, and deduces that it coincides with the contextual preorder.

Howe [How89, How96] establishes the same result in an operational setting without recourse to a denotational semantics. Howe's method turns out to be remarkably robust for variations of the λ-calculus. Gordon [Gor94, Gor95a, Gor95b] defines applicative similarity and bisimilarity for FPC via labelled transition systems, and uses Howe's method to show that they are compatible and hence coincide with the contextual preorder and equivalence respectively. Bernstein [Ber98] proves a general compatibility result for applicative bisimilarity upon higher-order languages defined by a rule format.

Non-Deterministic λ-Calculi

We now turn to non-deterministic variants of the λ-calculus. As discussed earlier, there are at least two motivations for adding non-determinism to a programming language such as the λ-calculus: for specification and refinement, and as a prelude to studying concurrency in a higher-order setting. Nearly all research is directed towards the latter, and can be classified by the operational semantics of the extensions made to the λ-calculus:

- "Pure" non-deterministic operators, e.g. *erratic choice*.
- Parallel operators, e.g., *ambiguous choice*.
- Parallel operators with message-passing primitives.

The "pure" non-deterministic extensions are usually based upon a binary erratic choice operator. We write $M \cup N$ for the binary erratic choice of M and N, but there is little consensus in the literature where $M \oplus N$, $M + N$, $M \mid N$, and $M \,\square\, N$ can be found[2]. Less often, the extension is a construct $?\underline{\omega}$ that can evaluate to any natural number.

The operational semantics of $M \cup N$ and $?\underline{\omega}$ [3] can be specified via a *reduction semantics* or an *evaluation semantics* (see [Plo81, Gun92]):

[2] The notations $M \oplus N$ and $M + N$ are avoided here because of a potential overlap with categorical notation used for describing models, and $M \mid N$ and $M \,\square\, N$ are not used because they often have a different meaning in process calculi.

[3] The operational semantics given for $?\underline{\omega}$ in chapter 3 is slightly different because of a lifting construct.

$$M \cup N \rightarrowtriangle M \qquad M \cup N \rightarrowtriangle N \qquad ?\underline{\omega} \rightarrowtriangle \underline{n} \quad (n \in \omega)$$

$$\frac{M \Downarrow^{\mathrm{may}} K}{M \cup N \Downarrow^{\mathrm{may}} K} \qquad \frac{N \Downarrow^{\mathrm{may}} K}{M \cup N \Downarrow^{\mathrm{may}} K} \qquad ?\underline{\omega} \Downarrow^{\mathrm{may}} \underline{n} \quad (n \in \omega)$$

The *may convergence* relation $\Downarrow^{\mathrm{may}}$ has a "may" annotation to emphasise that there can be more than one result. Note the similarity between the reduction semantics (the top row) for $M \cup N$ and the CSP and CCS labelled transitions for $P \sqcap Q$ and $\tau.P + \tau.Q$.

In a deterministic setting, a λ-calculus program *diverges* (has a non-terminating sequence of reductions) if and only if it does not *converge* (terminate) to a canonical program. In a non-deterministic setting, this is no longer true because a non-deterministic program M that cannot diverge has precisely the same convergence behaviour as the program $M \cup \Omega$ that either run M or the always divergent program Ω. The divergence behaviour can be captured by introducing a *may divergence* predicate $\Uparrow^{\mathrm{may}}$.

The definitions of contextual preorder and applicative similarity can be replayed with $\Downarrow^{\mathrm{may}}$, but it turns out that applicative similarity is a strictly finer relation than the contextual preorder in a non-deterministic setting because applicative similarity is sensitive to more branching behaviour than the contextual preorder (for an example, see p88 of [Las98b]). In fact, the situation is considerably more complicated than this. The obvious contextual preorder, henceforth called the *may contextual preorder*, relates M and N if:

$$\forall \mathcal{C}[-].\, \mathcal{C}[M] \Downarrow^{\mathrm{may}} \Longrightarrow \mathcal{C}[N] \Downarrow^{\mathrm{may}}$$

The *may convergence predicate* is defined using the *may convergence relation* by saying that $M \Downarrow^{\mathrm{may}}$ if and only if there exists a term K such that $M \Downarrow^{\mathrm{may}} K$. Now define a *must convergence predicate* $\Downarrow^{\mathrm{must}}$ as the complement, amongst programs, of the may divergence predicate $\Uparrow^{\mathrm{may}}$, so that $M \Downarrow^{\mathrm{must}}$ if and only if M always terminates. Then the *must contextual preorder* (see [Ong93, Sie93, Mor94, HM95, Las97, Las98b, Mor98, KSS99]) can be defined as:

$$\forall \mathcal{C}[-].\, \mathcal{C}[M] \Downarrow^{\mathrm{must}} \Longrightarrow \mathcal{C}[N] \Downarrow^{\mathrm{must}}$$

This relation is not comparable with the may contextual preorder or applicative similarity.

There are different varieties of applicative similarity also. The relation obtained by substituting $\Downarrow^{\mathrm{may}}$ for $\Downarrow$ in the previous definition of applicative similarity is called *lower similarity*. Three other preorders, *upper similarity*, *convex similarity*, and *refinement similarity*, can be defined using combinations of $\Downarrow^{\mathrm{may}}$ and $\Downarrow^{\mathrm{must}}$. In addition, three equivalence relations, *lower bisimilarity*, *upper bisimilarity*, and *convex bisimilarity*, can be defined. The bisimilarity relations are strictly finer than the largest symmetric relations contained in the similarity relation of the same name. These relations are considered for λ-calculi with erratic choice operators in [HA80, Ong93, Mor94, Las97, Las98b, LP98, Mor98].

As for deterministic λ-calculi, it is non-trivial to show that the similarity and bisimilarity relations are compatible. Lower similarity is the straightforward case because Howe's technique [How89] works without modification, and we find that lower similarity is compatible if one or both of the non-deterministic operators are present. Howe [How96] extends the technique in two ways. He gives a technique for deducing compatibility of a bisimilarity relation from a proof of compatibility of the corresponding similarity relation, and proves that upper similarity and convex similarity are compatible in the presence of a binary erratic choice operator but not $?\underline{\omega}$, i.e., the λ-calculus can exhibit finite but not countable non-determinism. Ong [Ong92a, Ong92b] independently uses the same method to obtain compatibility of convex similarity. Lassen and the author [Las97, LP98] independently establish that Howe and Ong's method and result can be extended to a λ-calculus that exhibits countable non-determinism. The remaining relation, refinement similarity, is shown to be compatible in the presence of the binary erratic choice operator or $?\underline{\omega}$ in theorem 5.4.14.

Concurrent λ-Calculi

Milner [Mil90] defines a translation from the pure untyped λ-calculus to the π-calculus, and proves that the equivalence induced on λ-terms by the equivalence on π-terms is strictly finer than contextual equivalence on λ-terms. This demonstrates that the π-calculus is more expressive for discriminating between (translations of) λ-terms. Sangiorgi [San94], Boudol [Bou93, Bou94a, BL95], and Lavatelli [Lav93] consider non-deterministic extensions of the λ-calculus with translations to the π-calculus. The increase in expressive power due to the extensions means that more λ-terms are distinguished by the variants of contextual equivalence and bisimilarity.

Boudol [Bou94b] extends the pure untyped λ-calculus with a parallel composition operator $M\|N$ in order to prove an expressivity result with respect to domain-theoretic models, i.e., the finite elements of the models are in the range of the denotation functions. The parallel composition operator interleaves reductions of its operands. Formally, $K\|N$ and $M\|K$ are canonical whenever K is canonical, and the following reduction rules apply to the parallel composition operator (where K is canonical):

$$\frac{M \rightarrowtriangle M'}{M\|N \rightarrowtriangle M'\|N} \qquad \frac{N \rightarrowtriangle N'}{M\|N \rightarrowtriangle M\|N'}$$

$$(K\|N)\,L \rightarrowtriangle K\,L\|N\,L \qquad (M\|K)\,L \rightarrowtriangle M\,L\|K\,L$$

The convergence behaviour of the erratic choice operator and the parallel composition operator are closely related (see [Lav93]), because whenever $M \cup N$ can be reduced to a canonical program there is a corresponding reduction sequence to a canonical program for $M\|N$ and vice-versa. The two operators differ in when they commit to one of their branches: erratic choice commits immediately to either the left or the right branch, whereas parallel composition never commits to a branch. The commitment avoidance of the parallel composition allows it to be interpreted as the join of its operands in the domain model, and so more elements of the domain are definable via terms of the programming language.

Jeffrey [Jef99] considers an operator with behaviour in between the two extremes of committing immediately and never committing. The operator $M \mathbin{[\,]} N$ can be defined with the following reduction rules (where K is canonical):

$$\frac{M \rightarrowtriangle M'}{M \mathbin{[\,]} N \rightarrowtriangle M' \mathbin{[\,]} N} \qquad \frac{N \rightarrowtriangle N'}{M \mathbin{[\,]} N \rightarrowtriangle M \mathbin{[\,]} N'}$$

$$K \mathbin{[\,]} N \rightarrowtriangle K \qquad M \mathbin{[\,]} K \rightarrowtriangle K$$

The terms $K \mathbin{[\,]} N$ and $M \mathbin{[\,]} K$ are not canonical even when K is. The operands of this parallel operator are evaluated concurrently, and, when one of the operands is canonical, that operand is chosen and the other is discarded. Note that once operands M and N are placed in parallel with $\talloblong$, it is not possible to extract both. If the results of both operands are required, the best that can be done is to run $M \talloblong N$ twice and hope that the value of M is returned on one run and the value of N on the other.

There is a mismatch between the parallel construct $M \talloblong N$ and the description of McCarthy's *ambiguous choice* operator [McC63] because the latter "avoids" divergence in one operand if the other operand cannot diverge. From the reduction rules give above it can be shown that $\Omega \talloblong \lambda x.x$ can diverge because the left-hand operand may always be chosen for reduction, i.e., the right-hand operand is ignored infinitely often. The reduction rules are too permissive because they allow unfair sequences of decisions between the left-hand and right-hand operands. Hence, they cannot be used to reason about the divergence properties of terms that use ambiguous choice, and the operator cannot be distinguished from erratic choice.

For some applications, it is important that operands are executed in parallel and that the divergence properties of ambiguous choice are considered. This necessitates a modified operational semantics that prevents unfair sequences of decisions. Plotkin [Plo82] defines a reduction semantics for a parallel operator that only allows fair sequences of decisions in a non-deterministic imperative programming language. Hughes and Moran [Mor94, HM95, Mor98] define reduction and evaluation semantics for McCarthy's ambiguous choice. Both approaches use resource annotations to limit the number of reductions that can take place on one operand without the other operand being chosen. In effect, the reduction semantics incorporate a fair scheduler.

To compare the two approaches, Plotkin's strategy can be modified for ambiguous choice in a λ-calculus setting. The language is extended with two variations of the ambiguous choice operator $M\,{}^{m}\!\talloblong N$ and $M \talloblong^{m} N$, where m is a natural number, and the reduction rules given above are replaced with:

$$
\begin{array}{cc}
M \talloblong N \rightarrowtriangle M\,{}^{m}\!\talloblong N \quad (m > 0) & M \talloblong N \rightarrowtriangle M \talloblong^{m} N \quad (m > 0) \\[4pt]
M\,{}^{0}\!\talloblong N \rightarrowtriangle M \talloblong^{m} N \quad (m > 0) & M \talloblong^{0} N \rightarrowtriangle M\,{}^{m}\!\talloblong N \quad (m > 0) \\[4pt]
\dfrac{M \rightarrowtriangle M'}{M\,{}^{m+1}\!\talloblong N \rightarrowtriangle M'\,{}^{m}\!\talloblong N} & \dfrac{N \rightarrowtriangle N'}{M \talloblong^{m+1} N \rightarrowtriangle M \talloblong^{m} N'} \\[4pt]
K\,{}^{m+1}\!\talloblong N \rightarrowtriangle K & M \talloblong^{m+1} K \rightarrowtriangle K
\end{array}
$$

The scheduler makes an initial decision to reduce either the left-hand or the right-hand operand, and chooses a strictly positive number of reductions to carry out on that side. When that number of reductions have taken place, the scheduler switches to reducing the other side. This process repeats until the enabled operand is canonical.

Hughes and Moran use a different scheduling mechanism. The language is extended with one operator that has two natural numbers m and n as resource annotations $M\,{}^{m}\!\talloblong^{n} N$. The term $M \talloblong N$ can be identified with $M\,{}^{0}\!\talloblong^{0} N$, and then the reduction rules are:

$$M\,{}^{0}\!\|^{0}\,N \rightharpoonup M\,{}^{m}\!\|^{n}\,N \quad (m,n>0)$$

$$\frac{M \rightharpoonup M'}{M\,{}^{m+1}\!\|^{n}\,N \rightharpoonup M'\,{}^{m}\!\|^{n}\,N} \qquad \frac{N \rightharpoonup N'}{M\,{}^{m}\!\|^{n+1}\,N \rightharpoonup M\,{}^{m}\!\|^{n}\,N'}$$

$$K\,{}^{m+1}\!\|^{n}\,N \rightharpoonup K \qquad M\,{}^{m}\!\|^{n+1}\,K \rightharpoonup K$$

This scheduler can defer part of the decision about which operand to evaluate because initially both operands are assigned resources and the scheduler can choose any interleaving of reductions that fits with those resources. Unfair sequences are avoided because the scheduler will only assign new resources when both operands have none.

Hughes and Moran define an evaluation semantics consisting of a may convergence relation and a may divergence predicate (see also [CC92]), and show that it captures precisely the convergence and divergence properties of the reduction relation. The rules for the may convergence relation have the same form as those for erratic choice:

$$\frac{M \Downarrow^{\text{may}} K}{M \,\|\, N \Downarrow^{\text{may}} K} \qquad \frac{N \Downarrow^{\text{may}} K}{M \,\|\, N \Downarrow^{\text{may}} K}$$

The may divergence predicate is defined by coinduction. The may divergence rules for erratic choice and ambiguous choice are:

$$\frac{M \Uparrow^{\text{may}}}{M \cup N \Uparrow^{\text{may}}} \qquad \frac{N \Uparrow^{\text{may}} K}{M \cup N \Uparrow^{\text{may}}}$$

$$\frac{M \Uparrow^{\text{may}} \qquad N \Uparrow^{\text{may}}}{M \,\|\, N \Uparrow^{\text{may}}}$$

These rules state that $M \cup N$ may diverge if either M or N does, whereas $M \,\|\, N$ may diverge only if both M and N may diverge. This evaluation semantics also agrees with Plotkin's reduction semantics. Consequently, the differences in the decision-making behaviour between the two reduction semantics are not visible in any of the contextual preorders or equivalences nor in the variants of similarity and bisimilarity because they are all defined in terms of the evaluation semantics. In addition, the operators with resource annotations are only added to the language in order to define the reduction relation, which is in turn used to define the may convergence relation and may divergence predicate. The operators $M\,{}^{m}\!\|\,N$, $M\,\|^{n}\,N$, and $M\,{}^{m}\!\|^{n}\,N$ are not needed with the alternative characterisation via the evaluation semantics, and so can be removed from the language. This is important because the variants of applicative similarity and bisimilarity cannot be compatible when they are present. For example, the terms $\lambda xy.x$ and $(\lambda x.x)\,(\lambda xy.x)$ are always equivalent, but the terms $\lambda xy.x\,{}^{1}\!\|^{1}\,\lambda xy.y$ and $(\lambda x.x)\,(\lambda xy.x)\,{}^{1}\!\|^{1}\,\lambda xy.y$ are not related[4] because the only possible reduction sequences are:

$$\lambda xy.x\,{}^{1}\!\|^{1}\,\lambda xy.y \rightharpoonup \lambda xy.x$$
$$\lambda xy.x\,{}^{1}\!\|^{1}\,\lambda xy.y \rightharpoonup \lambda xy.y$$
$$(\lambda x.x)\,(\lambda xy.x)\,{}^{1}\!\|^{1}\,\lambda xy.y \rightharpoonup \lambda xy.x\,{}^{0}\!\|^{1}\,\lambda xy.y \rightharpoonup \lambda xy.y$$
$$(\lambda x.x)\,(\lambda xy.x)\,{}^{1}\!\|^{1}\,\lambda xy.y \rightharpoonup \lambda xy.y$$

[4] They are related by lower similarity, but only in one direction.

The resource allocation prevents the second term from evaluating to $\lambda xy.x$. The problem is that the annotated operators are sensitive to the number of reduction steps that a computation takes, but the variants of applicative similarity and bisimilarity are not.

The compatibility of variants of applicative similarity and bisimilarity for a language with (unannotated) ambiguous choice is more complex than for erratic choice. Lower similarity and lower bisimilarity are compatible because the convergence behaviour of ambiguous choice is identical to that of erratic choice, and so Howe's technique applies. However, upper similarity and convex similarity fail to be compatible because ambiguous choice is not monotone for either relation (see [Mor98]). Upper bisimilarity also fails to be compatible for a language with ambiguous choice[5]. Compatibility of convex bisimilarity and refinement similarity for ambiguous choice is an open problem. The proof techniques used to establish their compatibility for erratic choice fail because they require convex similarity to be compatible.

The divergence avoidance property of ambiguous choice is useful for constructing certain kinds of systems (see [McC63, Bur88]). An example that recurs in the literature is the definition of a merge operator in terms of ambiguous choice. Several different merge operators are identified in the setting of non-deterministic dataflow (see [Kah74, BA81, Fau82, Den84, Bro86, Sta87, Bro88, Abr89, BPR90, Sta90, Mos91, Whi94, Mos95, Mos98]) by Moitra, Panangaden, Russell, Shanbhogue, and Stark [MP86, PS87, PS88b, PS88a, Rus90, Sha90]. The ideal merge is the *fair merge* that continually polls two input streams to see whether data is available. The *infinity-fair merge* can be defined using $?\underline{\omega}$ (which can in turn be defined from ambiguous choice), and will behave perfectly if the input streams do not ever run out of data, but may block if one of the streams does[6]. Intuitively, once the infinity-fair merge examines one input stream it will not change to the other until it has received some input. The *angelic merge* examines both input streams in parallel (using ambiguous choice), and then accepts an input if any is available. If both inputs streams always have data available, then the angelic merge may always choose the same input stream and ignore the other. Infinity-fair merge can be defined from angelic merge, which can be defined from fair merge, but neither of these relative definability results can be reversed. In addition, angelic merge and ambiguous choice can each be defined in terms of the other. Shanbhogue [Sha90] argues convincingly that these relative definability results preclude the reduction of fairness (such as ambiguous choice) to countable non-determinism.

Dataflow computation can be modelled in a functional programming setting using streams (lazy lists) that connect deterministic components (normal functional programs). Merge operators are often required, e.g., for merging streams carrying input events in functional programming language implementations of operating systems (see [Hen82, HO89, Tur90a]). Moran [Mor98] defines infinity-fair merge and angelic merge in a call-by-name λ-calculus. These merges have the same trace behaviour as the dataflow operators of the same name, but because the usual relations considered on non-deterministic λ-terms are finer than trace equivalence (on terms of stream type) there are likely to be inequivalent terms that could be called infinity-fair merge or angelic merge, and the relative definability results from the dataflow setting are not immediately applicable. It is not clear that it is possible to define a fair merge operator using resource annotations as with the reduction semantics for ambiguous choice, because the naive approach

[5]Pointed out to the author by Lassen (personal communication, 1997).

[6]This description does not distinguish infinity-fair merge from Shanbhogue's infinity-fair2 merge [Sha90].

leaves resource annotations within canonical terms and causes problems with compatibility as described above.

Modelling non-deterministic dataflow requires careful control over sharing of terms and when the non-determinism in merged streams is resolved. Hughes and Moran [HM95, Mor98] develop a theory for ambiguous choice using a call-by-need reduction strategy, based upon Launchbury's evaluation semantics [Lau93], to ensure that sharing occurs. This results in *singular choice* (see [Cli82, SS92, KSS99]).

Concurrent λ-Calculi with Communication

There is a considerable body of research on higher-order concurrent programming languages with message-passing. The design space is large, but the majority of proposals follow one of two related approaches. The first is to extend a λ-calculus with a mechanism for starting new threads of execution to evaluate expressions, together with primitives for passing messages between threads, e.g., Reppy's Concurrent ML (CML) [Rep92, FHJ95, Jef95, Rep99] and Facile [PGM90, Ama94, ALT95]. Another approach is to extend a process calculus such as CCS or the π-calculus with the ability to transmit and receive processes, not just values, e.g., Thomsen's Calculus of Higher-Order Communicating Systems (CHOCS) [Tho89, Ama93, Tho93, AD95, Tho95] and Sangiorgi's higher-order π-calculus [San93], amongst others [Hen94]. Boudol's blue calculus [Bou97a, Bou97b] lies in between these two approaches. Various contextual preorders or equivalences and variants of similarity and bisimilarity are considered for these calculi. For similarity and bisimilarity, the challenge is to ensure that the terms passed on channels are considered up to the semantic relation being defined, not syntactic identity. Fairness is not addressed in the above research, except by Reppy [Rep92] who considers fairness for CML.

1.3 Outline of Dissertation

There are many possible non-deterministic λ-calculi and semantic relations. For example, the following axes can be considered:

- Which non-deterministic operator(s), e.g., binary erratic choice $M \cup N$, countable erratic choice $?\underline{\omega}$, indexed erratic choice as in CSP, binary ambiguous choice $M \,[\!]\, N$, or fair merge?
- Which reduction strategy, e.g., call-by-name, call-by-value, or call-by-need?
- Which semantic relation(s), e.g., contextual preorders and equivalences or variants of applicative similarity and bisimilarity?

The goal of this dissertation is to define and study non-deterministic λ-calculi that can be used as a stepping stone to higher-order specification and refinement formalisms or to higher-order programming languages with communication primitives such as CML. The definitions and results of interest include:

- Definitions of similarity and bisimilarity with compatibility results and additional reasoning principles, e.g., Scott induction for the similarity relations. How are these relations related?
- Relative definability results between different forms of non-determinism. Results are specific to a precise language and semantic relation, but can still help to guide the design of new languages (as happens with the dataflow relative definability results discussed in section 1.2.3).
- How do the semantic relations vary as the form of non-determinism present in the language changes? For example, if a language with binary erratic choice is extended with the countable non-determinism operator $?\underline{\omega}$, do the semantic relations change?

In this dissertation we fill in some of the previously unknown results for a collection of non-deterministic λ-calculi that exhibit erratic non-determinism. The languages are obtained as fragments of a λ-calculus $\mathscr{L}$ that contains all of the non-deterministic terms that we consider. The definitions and results take a fragment of $\mathscr{L}$ as a parameter, so that we obtain, e.g., compatibility of convex similarity for the language fragment containing $?\underline{\omega}$ as an instance of the more general compatibility result.

The non-deterministic extensions have the form $?\langle M_n \mid n < \kappa\rangle$, where κ is a natural number or ω, and $\{M_n \mid n < \kappa\}$ is a set of terms with the same type. With this general form of erratic non-determinism we can define $?\underline{\omega}$ as $?\langle \underline{n} \mid n < \omega\rangle$, and $M \cup N$ using $?\langle \mathsf{false}, \mathsf{true}\rangle$ with a conditional. It is also possible to define non-deterministic terms that serve as specifications, e.g., a program that chooses any prime number. The language is expressive because the set of terms $\{M_n \mid n < \kappa\}$ need not be recursive or even recursively enumerable.

These non-deterministic extensions provide a wide range of non-equivalent non-deterministic terms, perhaps suitable for program specification, whilst retaining a simple, uniform operational semantics: $?\langle M_n \mid n < \kappa\rangle$ reduces in one step to (the lift of) one of the component terms. Ambiguous choice is avoided because some compatibility properties are open problems or known to be false. The lack of a reasonable operational semantics prevents the use of fair merge.

The relative definability properties of the non-deterministic extensions are studied in the language $\mathscr{L}$. Relative definability can be considered with respect to any of the operationally-defined equivalences such as convex bisimilarity. This differs somewhat from the usual approach, as used for *Turing degrees* (see [Rog67, Sho71, Odi89]) or Sazonov's *degrees of parallelism* (see [Saz75, Lic96, Buc97]), where relative definability of the non-definable elements of a denotational model is investigated with respect to equality in the model. The operational approach is possible here because the non-deterministic extensions admit a straightforward operational semantics, and is appropriate because there is no known denotational model for the most interesting relation, convex bisimilarity.

The choice of reduction strategy is important because of the need to control resolution of non-determinism. A purely call-by-name λ-calculus does not offer sufficient control. There are several alternatives:

- Use a call-by-value λ-calculus (see [Ong93, How96, Las98b]). The strictness of function application can be used to resolve non-determinism at specific points in a program, and non-deterministic terms can be duplicated if necessary by using a thunk.

- Use a call-by-need λ-calculus (see [HM95, Mor98]). Non-deterministic terms can be duplicated if necessary by using a thunk (assuming that the operational semantics is non-optimal).

- Use the syntax of Moggi's computational λ-calculus (see [Mog89b, Mog89a, Mog91, Pit91, CP92]) with a call-by-name reduction strategy. Call-by-value can be simulated using the construct for sequencing computations $\mathsf{let}\, x \Leftarrow M\, \mathsf{in}\, N$.

The last option is used here, following Jeffrey [Jef99] for a non-deterministic λ-calculus and Crole, Gordon, and Wadler [Wad92, Gor94, CG95] for deterministic λ-calculi. Non-determinism and non-termination can be restricted to computation types $\mathsf{P}_\perp(\sigma)$ and this proves useful for some results concerning the collapse of the variants of applicative similarity and bisimilarity at types where the occurrences of the computation type constructor are restricted. Intuitively, the semantics of $\mathsf{P}_\perp(\sigma)$ is $\mathcal{P}_{\mathrm{ne}}(\llbracket\sigma\rrbracket_\perp)$, where $\mathcal{P}_{\mathrm{ne}}(X)$ is the set of non-empty subsets of a set X.

Controlling the resolution of non-determinism is more convenient in the computational λ-calculus than in a call-by-value λ-calculus. For example, the following two terms can be applied to the non-deterministic term $\vdash ?\langle \underline{0}, \underline{1}\rangle : \mathsf{P}_\perp(\mathsf{nat})$, but only the second can choose to add 0 and 1:

$$\vdash \lambda x{:}\mathsf{P}_\perp(\mathsf{nat}).\, \mathsf{let}\, y \Leftarrow x\, \mathsf{in}\, [\mathsf{plus}\,(y,y)] : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{nat})$$
$$\vdash \lambda x{:}\mathsf{P}_\perp(\mathsf{nat}).\, \mathsf{let}\, y \Leftarrow x\, \mathsf{in}\, \mathsf{let}\, z \Leftarrow x\, \mathsf{in}\, [\mathsf{plus}\,(y,z)] : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{nat})$$

Call-by-need is not used because there is no straightforward treatment of applicative similarity or bisimilarity for a call-by-need λ-calculus. The problem is demonstrated by the fact that projections are insufficient for testing terms of product type. For example, the first and second projections of the following term do not give any information about the sharing of $?\langle \underline{0}, \underline{1}\rangle$ that occurs with a call-by-need reduction strategy:

$$(\lambda x.\, \mathsf{tuple}\, \langle x, x\rangle)\, (?\langle \underline{0}, \underline{1}\rangle)$$

In addition to function type and computation type constructors, the language $\mathscr{L}$ permits the formation of indexed sum (coproduct) types. The indexing set can be any ordinal less than or equal to ω, and so the sum of a countable set of types can be formed. If unit is the singleton type, then the natural number type can be defined by:

$$\mathsf{nat} \stackrel{\mathrm{def}}{=} \mathsf{sum}\, \langle \mathsf{unit} \mid n < \omega\rangle$$

An indexed case construct is used to decompose terms of an indexed sum type. For any (set-theoretic) function $f \in \omega \to \omega$, we have the corresponding infinite term:

$$\vdash \lambda x{:}\mathsf{nat}.\, \mathsf{case}\, x\, \mathsf{of}\, \langle x_i.\underline{f(i)} \mid i < \omega\rangle : \mathsf{nat} \to \mathsf{nat}$$

As with the indexed erratic choice constructor, there is no requirement that the case construct be recursive. This unusual addition to the language is adopted to facilitate writing non-deterministic programs that act as specifications. For example, the following program takes a natural number as an argument and then returns any number between 0 and its argument:

$$\vdash \lambda x{:}\mathsf{nat}.\, \mathsf{case}\, x\, \mathsf{of}\, \langle x_i.?\langle \underline{n} \mid 0 \le n \le i\rangle \mid i < \omega\rangle : \mathsf{nat} \to \mathsf{P}_\perp(\mathsf{nat})$$

The type system for the language $\mathscr{L}$ does not include recursive or coinductive types because some of the definitions and results in chapter 4 are established via induction on the type structure.

Variants of applicative similarity and bisimilarity are considered instead of contextual preorders or equivalences because they can also be defined upon the abstract structures, typed transition systems, defined in chapter 4, in the same way that applicative similarity or bisimilarity can be defined upon applicative transition systems (see [Abr90, AO93]). Although the type system does not include recursive or coinductive types and it would be possible to define preorders and equivalences as logical relations, we use the coinductively-defined variants of applicative similarity and bisimilarity where possible so that the same techniques can be reused if the language is subsequently extended with recursive types.

Preliminaries

Chapter 2 covers background material about trees, transition systems, coinduction, similarity and bisimilarity, recursive ordinals, and some of the choice operators that appear in the literature. We discuss various kinds of transition systems (labelled and unlabelled, with and without divergence), and consider examples generated by sets using the membership relation to define the transition relation. In subsequent chapters, the examples are modified for more complex transition systems that are suitable for studying a non-deterministic λ-calculus. Similarity and bisimilarity are defined for transition systems with and without divergence. For transition systems with divergence, we consider the possible variants of similarity and bisimilarity that correspond to variants of applicative similarity and bisimilarity that appear in the sequel. The relationship between recursive trees and recursive ordinals is proven in detail, in preparation for a later result concerning the language fragment containing the countable choice operator $?\underline{\omega}$. Finally, we define the actions of global angelic choice, ambiguous choice, erratic choice, local demonic choice, and global demonic choice in a naive model, and show when their behaviour cannot be distinguished by common semantic relations.

The Non-Deterministic λ-Calculus $\mathscr{L}$

In chapter 3 we define the non-deterministic λ-calculus $\mathscr{L}$, its type system, and reduction and evaluation semantics. Reduction is permitted on open terms to facilitate the proof of Scott induction in chapter 5. The evaluation semantics is presented as an inductively-defined may convergence relation $\Downarrow^{\text{may}}$ and a coinductively-defined may divergence predicate $\Uparrow^{\text{may}}$. The may divergence predicate is the complement of an inductively-defined must convergence predicate $\Downarrow^{\text{must}}$. The usual properties are established such as subject reduction, normalisation of terms of value type (as opposed to computation type), and the relationship between the two operational semantics.

The fragments of the language that are used in subsequent proofs are defined with a closure operator on sets of terms. The fragments must be closed under PCF-like operations, substitution, and taking sub-terms. Fragments are shown to be closed under reduction, and thus we obtain a collection of non-deterministic λ-calculi with a uniform operational semantics. Ordinal bounds are proven for the heights of must convergence derivations, following similar results in [AP86].

For example, the must convergence derivations for the smallest fragment containing $?\underline{\omega}$ are bounded by the least non-recursive ordinal ω_1^{CK}.

Typed Transition Systems

Chapter 4 defines the class of typed transition systems (TTS) as a subclass of the class of labelled transition systems with divergence. Each state has a unique type, and the labels on transitions from a state are restricted according to its type, e.g., transitions from a state of function type $\sigma \to \tau$ are labelled with states of type σ. This builds upon previous definitions of quasi-applicative transition systems [Abr90, AO93], applicative structures [Mit96], non-deterministic applicative transition systems [Ong92a], and σ-transition systems [OP93]. Four variants of applicative similarity and bisimilarity (lower, upper, convex, and refinement) are defined upon the states of each TTS. We identify sets of states for which the relations always coincide. To show that the relations are different in general, we define a simple TTS $\mathscr{S}$ by induction on the type structure: the states of sum, product, and function type are interpreted by the set-theoretic coproduct, product, and function spaces respectively, and the states of a computation type $\mathsf{P}_\perp(\sigma)$ are the non-empty subsets of the lift of the states with type σ. Finally, we investigate maps between TTSs that respect transitions. In particular, we consider a restriction operation on TTSs that discards some states, and induces a map from the restricted TTS to the original. We show that convex bisimilarity on the restricted TTS is coarser than convex bisimilarity on the original, and that every sufficiently well-behaved TTS is the restriction of the TTS $\mathscr{S}$. This is pertinent because each fragment of the language determines a well-behaved TTS that is also a restriction of the TTS for the full language $\mathscr{L}$.

Programming Language TTSs

We study the typed transition systems determined by the fragments of the programming language defined in chapter 3. The open extensions of the variants of applicative similarity are shown to be compatible for all fragments, and the open extensions of the variants of applicative bisimilarity are shown to be compatible for a wide collection of fragments. The compatibility proof uses Howe's method [How89] and Howe and Ong's [Ong92a, How96] extension of the method, as well as incorporating new techniques for handling countable non-determinism, refinement similarity, and fragments. The proof makes use of Lassen's algebra of relations [Las98b], which builds upon Gordon's [Gor94] reorganisation of Howe's method.

We prove relative definability properties of some common forms of erratic non-determinism with respect to the variants of applicative bisimilarity. These constitute lower sets of relative definability equivalence classes for closed terms of type $\mathsf{P}_\perp(\mathsf{nat})$. We then show that convex bisimilarity is different for nearly all of the relative definability equivalence classes that we define, e.g., convex bisimilarity for the smallest fragment containing countable choice $?\underline{\omega}$ is strictly finer than convex bisimilarity for the smallest fragment containing $?\langle\mathsf{false},\mathsf{true}\rangle$.

We also consider fixed-points for the lower, upper, and convex variants of applicative similarity and prove the Scott induction principle for these relations for all fragments of the language $\mathscr{L}$.

Discussion

We summarise and consider directions for future research.

1.4 Contributions

The contributions of this dissertation are:

- Chapter 2 provides an account of elementary material including trees, transition systems, coinduction, similarity and bisimilarity, and recursive ordinals. Examples are given to demonstrate the relationships between these objects.

- Typed transition systems abstract the structure required to define the variants of similarity and bisimilarity. Several results are presented in this abstract setting. General inclusions between the relations are established in lemmas 4.2.4, 4.2.5, and 4.2.6. A major case study of a typed transition system is described in sections 4.3 and 4.4, and this provides a rich source of examples of non-inclusions between the relations. A category of typed transition systems is defined, and theorem 4.5.15 shows that the typed transition system from the case study is a weak terminal in a non-trivial subcategory.

- Ong and Howe independently proved compatibility of some of the lower, upper, and convex variants of similarity and bisimilarity for λ-calculi that exhibit finite non-determinism, and Lassen extended their method to countable non-determinism. This extension was proved independently by the author. In this dissertation, the same method is used to prove compatibility of the lower, upper, and convex variants of similarity and bisimilarity for $\mathscr{L}$ and all of its fragments (see theorem 5.4.8). However, compatibility of the variants of bisimilarity (see theorem 5.4.14) requires a new technique and some restrictions upon the fragments. Another new argument is used to establish compatibility of refinement similarity, a previously open problem, for a restricted collection of fragments (see theorem 5.4.18). In each case, the compatibility results apply to many different non-deterministic λ-calculi (fragments of $\mathscr{L}$).

- Theorem 5.7.9 is the Scott induction principle with respect to the lower, upper, and convex variants of similarity for $\mathscr{L}$ and all of its fragments.

- A lower set of relative definability equivalence classes, with respect to convex bisimilarity, is identified in lemmas 5.5.4, 5.5.5, and 5.5.6. The approach to relative definability is novel in that it does not require a denotational model, in contrast to the usual formulations of Turing degrees and Sazonov's degrees of parallelism. New examples are given to show that the relative definability equivalence classes have different theories with respect to convex bisimilarity (see proposition 5.6.4). In particular, the addition of countable non-determinism to a finitely non-deterministic programming language is not conservative with respect to the upper and convex variants of similarity, mutual similarity, and bisimilarity, i.e., there are programs that can be distinguished by countably non-deterministic programs but not by finitely non-deterministic programs.

Chapter 2

Preliminaries

This chapter is a review of basic notions including trees and ordinals (section 2.1), labelled and unlabelled transition systems with and without divergence (section 2.2), induction and coinduction (section 2.3), similarity and bisimilarity (section 2.4), recursive ordinals and recursive trees (section 2.5), and some of the binary choice operators that appear in the literature (section 2.6). Examples are given to demonstrate the relationships between these objects and relations. The typed transition systems defined in chapter 4 build upon the discussion and examples of transition systems in this chapter.

2.1 Ordinals and Trees

We review ordinals and trees in preparation for a discussion of recursive ordinals and recursive trees in section 2.5. Thorough accounts can be found in [Acz77, Kun77, Lev79, Gir87, Odi89, Kun80, Pot90, Gal91, Joh87].

Definition 2.1.1 A set A is an ***ordinal*** if $C \in B \in A$ implies $C \in A$, and, for all $B, C \in A$, either $B \in C$, $C \in B$, or $B = C$ holds.

Example 2.1.2 The natural numbers can be defined as ordinals. Define $0 \stackrel{\text{def}}{=} \emptyset$ and $Succ(A) \stackrel{\text{def}}{=} A \cup \{A\}$. Then the numbers from 0 to 4 are the sets:

$$
\begin{aligned}
&0 = \emptyset \\
&1 = Succ(0) = 0 \cup \{0\} = \{\emptyset\} \\
&2 = Succ(1) = 1 \cup \{1\} = \{\emptyset, \{\emptyset\}\} \\
&3 = Succ(2) = 2 \cup \{2\} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} \\
&4 = Succ(3) = 3 \cup \{3\} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}\}
\end{aligned}
$$

The set of natural numbers ω is the least set with respect to $\subseteq$ that contains 0 and is closed under $Succ(\cdot)$ (the existence of one such set must be postulated). The principle of mathematical induction depends on the fact that ω is the least such set. The empty set $\emptyset$ is an ordinal, and it is straightforward to show that $Succ(A)$ is an ordinal whenever A is an ordinal, so we can use mathematical induction to show that every natural number (element of ω) is an ordinal. In addition, it can be shown that ω is itself an ordinal and that the restriction of $\in$ to $\omega \times \omega$ is an irreflexive, transitive relation that determines a total order on the natural numbers. The set of natural numbers ω is a *limit ordinal* because it is neither empty nor $Succ(A)$ for some other ordinal A. It is possible to construct ordinals greater than ω by using the $Succ(\cdot)$ operator and taking the least upper bounds of sets of ordinals. The arithmetic operations of addition, multiplication, and exponentiation can be defined for all ordinals, but have some unusual properties such as a lack of commutativity for ordinal arithmetic.

Lemma 2.1.3 states that every ordinal is the union of the successors of its elements.

Lemma 2.1.3 Let A be an ordinal. Then:

$$A = \bigcup\{Succ(B) \mid B \in A\}$$

Proof The inclusion $A \subseteq \bigcup\{Succ(B) \mid B \in A\}$ is trivial. For the other direction, consider $D \in \bigcup\{Succ(B) \mid B \in A\}$, so there exists $B \in A$ such that $D \in B \cup \{B\}$, i.e., $D \in B \in A$ or $D = B \in A$. If $D \in B \in A$, then $D \in A$ because A is an ordinal. Therefore $A \supseteq \bigcup\{Succ(B) \mid B \in A\}$. $\square$

The total order $\langle \omega, \in \rangle$ has the property that there are no strictly decreasing ω-chains. More generally, we say that a relation $\mathcal{R}$ is *well-founded relation* if there are no ω-chains with respect to $\mathcal{R}^{\mathrm{op}}$, the dual of $\mathcal{R}$.

Definition 2.1.4 A binary relation $\mathcal{R} \subseteq A \times A$ is ***well-founded*** if there are no ω-chains $\langle a_n \in A \mid n \in \omega \rangle$ such that $\langle a_{n+1}, a_n \rangle \in \mathcal{R}$ for all $n \in \omega$. A ***well-order*** is a total order $\langle A, \leq \rangle$ such that $<$ is well-founded.

Well-founded strict partial orders are often of interest, but, in general, a well-founded relation need not be transitive. It is straightforward to show that a relation is well-founded if and only if its transitive closure is well-founded. Note that a well-founded relation must be irreflexive.

Sets with a well-founded relation admit a form of induction known as *well-founded induction*. The idea is to prove that an element satisfies a property whenever all strictly smaller elements satisfy the property.

Proposition 2.1.5 (Principle of Well-Founded Induction) Let $\mathcal{R} \subseteq A \times A$ be well-founded and $B \subseteq A$ such that:

$$\forall a \in A. (\forall b \in A. \langle b, a \rangle \in \mathcal{R} \Longrightarrow b \in B) \Longrightarrow a \in B$$

Then $B = A$.

Proof Suppose for a contradiction that there exists $a_0 \in A \setminus B$. By assumption, there must exist $a_1 \in A \setminus B$ such that $(a_1, a_0) \in \mathcal{R}$. This process can be iterated to obtain an ω-chain $\langle a_n \in A \setminus B \mid n \in \omega \rangle$ such that $(a_{n+1}, a_n) \in \mathcal{R}$ for all $n \in \omega$, which contradicts the well-foundedness of $\mathcal{R}$. $\square$

Traditional set theory is based upon the notion that sets are well-founded, meaning that $\in$ is well-founded. This is ensured by the axiom of Foundation:

$$\forall A. A \neq \emptyset \Longrightarrow \exists B \in A. B \cap A = \emptyset \qquad \text{(Foundation)}$$

With this axiom we can deduce that $\in$ is well-founded. Suppose for a contradiction that $\langle A_i \mid i \in \omega\rangle$ is an ω-chain such that, for all $i \in \omega$, $A_{i+1} \in A_i$. Let $C = \{A_i \mid i \in \omega\} \neq \emptyset$. By the axiom of Foundation, there exists $A_j \in C$ such that $A_j \cap C = \emptyset$. However, that contradicts $A_{j+1} \in A_j \cap C$. Therefore no strictly descending ω-chains can exist with respect to $\in$. This leads to an $\in$-induction principle (see [Joh87]), which is essentially well-founded induction on $\in$.

One of the consequences of Foundation is that every member of an ordinal is also an ordinal. We can also show that $\in$ coincides with $\subsetneq$ and that $\in$ determines a total order on ordinals, so $\in$ well-orders the ordinals (although the ordinals constitute a proper class not a set, and we are quietly assuming the axiom of Choice).

Lemma 2.1.6 If A and B are ordinals, then $B \subseteq A$ if and only if $B \in A$ or $B = A$.

Proof The right-to-left direction is trivial because A is an ordinal. For the other direction, define:

$$X \stackrel{\text{def}}{=} \{C \in Succ(A) \mid B \subsetneq C\}$$

If $B = A$ we are done. Otherwise, $A \in X$, and by Foundation there exists $D \in X$ such that $D \cap X = \emptyset$. Now $D \in X$ implies that there exists $C \in D \setminus B$. We claim that $B \subseteq C$. Consider any $E \in B \subseteq D$, so that $E \in D$. D is an ordinal because $D \in Succ(A)$, and so $C \in E$, $C = E$, or $E \in C$. However, $C \notin B$ and B an ordinal imply that $C \notin E$ and $C \neq E$, so we have $E \in C$ as desired. Therefore $B \subseteq C$. We can then deduce that $B = C$ because $C \notin X$, and so $B = C \in D \in Succ(A) = A \cup \{A\}$. Therefore $B \in A$ or $B = A$. □

Lemma 2.1.7 If A and B are ordinals, then $A \in B$, $A = B$, or $B \in A$.

Proof By lemma 2.1.6, it suffices to show that $A \subseteq B$ or $B \subseteq A$. Define:

$$X \stackrel{\text{def}}{=} \{C \in Succ(A) \mid \exists D \in Succ(B). C \nsubseteq D \wedge D \nsubseteq C\}$$

We claim that $X = \emptyset$, in which case $A \notin X$ and so $A \subseteq B$ or $B \subseteq A$. For a contradiction, suppose that $X \neq \emptyset$. By Foundation, there exist $E \in X$ and $F \in Succ(B)$ such that $E \cap X = \emptyset$, $E \nsubseteq F$, and $F \nsubseteq E$. There must exist ordinals $C \in E \setminus F$ and $D \in F \setminus E$. Suppose that $C \subseteq D$. By lemma 2.1.6, this is equivalent to $C \in D$ or $C = D$. However, neither of these are possible because F is an ordinal, $D \in F$, and $C \notin F$. Therefore $C \nsubseteq D$, and similarly $D \nsubseteq C$. Thus we have the contradiction $C \in E \cap X \neq \emptyset$. □

Example 2.1.8 Although $\in$ is well-founded, it is easy to see that its dual $\ni$ is not. For example, consider $\langle n \mid n \in \omega\rangle$, where $Succ(n) \ni n$, for all $n \in \omega$.

We now turn to forests and trees. A forest is a partial order such that the corresponding strict partial order is well-founded and the down-set of every element is totally ordered. This definition permits forests with limit elements.

Definition 2.1.9 A ***forest*** is a partial order $\langle A, \le \rangle$ such that $\{b \in A \mid b < a\}$ is well-ordered by $\le$ for all $a \in A$. A ***root*** of a forest is an element of A that is minimal with respect to $\le$. A ***tree*** is a forest with only one root. A forest or tree $\langle A, \le \rangle$ is ***well-founded*** if $>$ is also well-founded, i.e., there are no strictly increasing ω-chains. An element $b \in A$ is a ***successor*** of $a \in A$ if $a < b$ and whenever $a \le c < b$, for some $c \in A$, we have $a = c$. An element $a \in A$ is a ***limit*** if it is not a root or the successor of some other element.

König's lemma identifies a condition under which a tree is not well-founded. This result is used in chapter 3.4.8 to illustrate the difference between binary erratic choice and the countable choice operator $?\underline{\omega}$.

Lemma 2.1.10 (König) If $\langle A, \le \rangle$ is a tree such that A is infinite and every element has a finite number of successors, then it is not a well-founded tree, i.e., there exists a strictly increasing ω-chain $\langle a_n \in A \mid n \in \omega \rangle$ such that, for all $n \in \omega$, $a_n < a_{n+1}$.

Proof Define:

$$B = \{a \in A \mid \{b \in A \mid a < b\} \text{ is infinite}\}$$

By hypothesis, B contains the root of the tree and is thus non-empty. For any $a \in B$, there must be at least one successor $b \in B$, because otherwise the finite number of successors of a would each have finite up-sets, contradicting the fact that the up-set of a is infinite. Pick $a_0 \in B$. This process can be iterated to obtain a strictly increasing ω-chain $\langle a_n \in B \mid n \in \omega \rangle$ such that $a_n < a_{n+1}$, for all $n \in \omega$. □

The primary examples of trees with limit points are ordinals greater than ω.

Example 2.1.11 Every ordinal A determines a tree $\langle A, \subseteq \rangle$ because $\in$ is a well-founded strict total order. Every element has at most one successor, so the trees do not branch at all. The root of the tree is $\emptyset$. If A is greater than or equal to ω, then the tree is not well-founded (cf. example 2.1.8). If A is strictly greater than ω, then the tree contains limit ordinals which are limit elements.

Ordinals also determine another kind of tree that are introduced in definition 2.1.13.

Trees are often constructed from sets of sequences, and, in general, such trees are not total orders.

Example 2.1.12 Let A be a non-empty set and $B \subseteq \bigcup\{A^n \mid n \in \omega\}$ a prefix-closed set. Then $\langle B, \sqsubseteq \rangle$ is a tree, where $\sqsubseteq$ is the prefix order, because a strictly decreasing chain of finite sequences determines a strictly decreasing sequence of natural numbers (the lengths of the sequences) and thus must have finite length. The empty sequence $\langle\rangle$ is the root of the tree. Although every element of B is a finite sequence of elements from A, the trees need not be well-founded because for any element $a \in A$ there is a strictly increasing ω-chain:

$$\langle\rangle \sqsubseteq \langle a \rangle \sqsubseteq \langle a, a \rangle \sqsubseteq \langle a, a, a \rangle \sqsubseteq \dots$$

If the infinite sequence $\langle a \mid n \in \omega \rangle$ is added to the tree, then it is a limit of the above chain. More generally, it is possible to construct trees in this manner using transfinite sequences of elements from A, i.e., functions from ordinals to A.

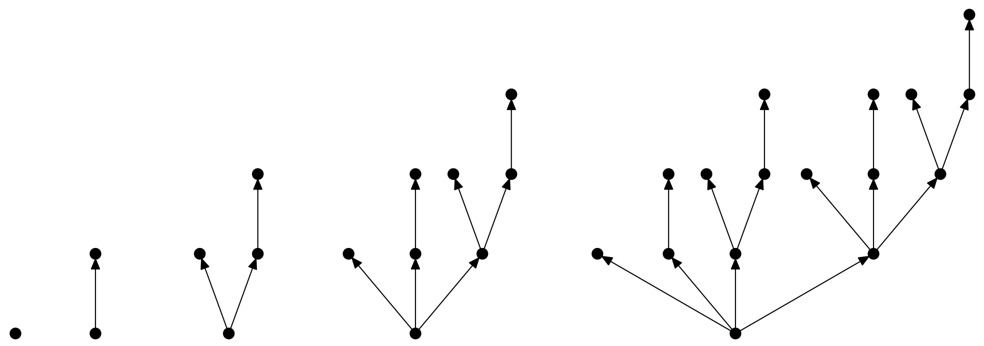


Figure 2.1: $\in$-trees of ordinals from 0 to 4

We now consider a special case of example 2.1.12. Sets are often depicted as $\in$-*trees*. For example, the $\in$-trees for the ordinals from 0 to 4 are illustrated in figure 2.1. The elements of the $\in$-tree for a set A are finite strictly descending chains of sets related by $\in$, where the first element of a non-empty chain is a member of A.

Definition 2.1.13 The $\in$-***tree*** of a set A is $Tree(A) \stackrel{\text{def}}{=} \langle B, \sqsubseteq \rangle$ where $\sqsubseteq$ is the prefix order and B is defined by:

$$B \stackrel{\text{def}}{=} \{\langle\rangle\} \cup \{\langle A_0, A_1, \ldots, A_n \rangle \mid n \in \omega \wedge A_0 \in A \wedge \forall i < n. A_{i+1} \in A_i\}$$

$\in$-trees are always well-founded because the axiom of Foundation ensures that there are no strictly descending ω-chains of sets. However, it is worth considering why the lack of well-foundedness of $\ni$, demonstrated in example 2.1.8, does not prevent an $\in$-tree from being a tree. In fact, the ω-chain $\langle n \mid n \in \omega \rangle$ becomes an ω-antichain $\langle \langle n \rangle \mid n \in \omega \rangle$ (a sequence of singleton sequences) in $Tree(\omega)$. More generally, if A is an ordinal, then $\langle \langle B \rangle \mid B \in A \rangle$ is an A-antichain of successors of the root $\langle\rangle$ in $Tree(A)$.

We can go back from trees to ordinals by associating ordinals with the elements of a set ordered by a well-founded relation.

Definition 2.1.14 Let $\mathcal{R} \subseteq A \times A$ be a well-founded relation. The ***rank*** of $a \in A$ with respect to $\mathcal{R}$ is an ordinal defined by well-founded induction on $\mathcal{R}$:

$$Rank(a, \mathcal{R}) \stackrel{\text{def}}{=} \bigcup \{Succ(Rank(b, \mathcal{R})) \mid \langle b, a \rangle \in \mathcal{R}\}$$

The rank function is a closure operator upon sets ordered by $\in$.

Lemma 2.1.15 Let A be an ordinal. Then $Rank(A, \in) = A$.

Proof Define:

$$X \stackrel{\text{def}}{=} \{B \in Succ(A) \mid Rank(B, \in) \neq B\}$$

If $X = \emptyset$ we are done. For a contradiction, suppose $X \neq \emptyset$. By Foundation, there exists $C \in X$ such that $C \cap X = \emptyset$ and $Rank(C, \in) \neq C$. For all $B \in C$, we know $B \notin X$, so $Rank(B, \in) = B$. Thus, using lemma 2.1.3, we have:

$$Rank(C, \in) = \bigcup\{Succ(Rank(B, \in)) \mid B \in C\} = \bigcup\{Succ(B) \mid B \in C\} = C$$

Therefore we have the contradiction $Rank(C, \in) = C$. $\square$

Two different ordinal measures are associated with trees depending on whether the order or its dual is used. The length of a tree is a measure of how long strictly increasing chains can be, starting from the root. The rank of a well-founded tree is a measure of its breadth, because it is sensitive to the kind of antichains that appear in the $\in$-trees for infinite ordinals (see the discussion after definition 2.1.13).

Definition 2.1.16 Let $\langle A, \leq \rangle$ be a tree.

1. The ***length*** $Len(A, \leq)$ (also known as the ***height***) of the tree is defined by:

$$Len(A, \leq) \stackrel{\text{def}}{=} \bigcup\{Succ(Rank(a, <)) \mid a \in A\}$$

2. If $\langle A, \leq \rangle$ is a well-founded tree with root $a \in A$, then the ***rank*** of the tree is $Rank(a, >)$.

Example 2.1.17 Define a well-founded tree $\langle A, \leq \rangle$ by:

$$A \stackrel{\text{def}}{=} \{\star\} \cup \{\langle m, n \rangle \in \omega \times \omega \mid n \leq m\}$$
$$\star < \langle m, n \rangle \qquad \text{and} \qquad \langle m_1, n_1 \rangle < \langle m_2, n_2 \rangle \Longleftrightarrow m_1 = m_2 \wedge n_1 < n2$$

The strict order $<$ and its dual $>$ are well-founded. This tree is illustrated in figure 2.2. The ranks of the elements with respect to the order and its dual are:

$$\begin{array}{ll} Rank(\star, <) = 0 & Rank(\langle m, n \rangle, <) = n + 1 \\ Rank(\star, >) = \omega & Rank(\langle m, n \rangle, >) = m - n \end{array}$$

In this example the length and the rank of the tree are both ω. In general, the length and rank of a tree are not the same.

The length of any well-founded tree or, more generally, tree without limit elements is less than or equal to ω. The length of a tree with limits is greater than or equal to ω. The rank of a tree is only defined if the tree is well-founded. In contrast to the length, the rank of a well-founded tree is not bounded by ω (or indeed by any ordinal). These points are illustrated in lemma 2.1.18.

Lemma 2.1.18 Let A be an ordinal. Then:

1. The length of the tree $\langle A, \subseteq \rangle$ is A.

2. The rank of the well-founded $\in$-tree $Tree(A)$ is A.

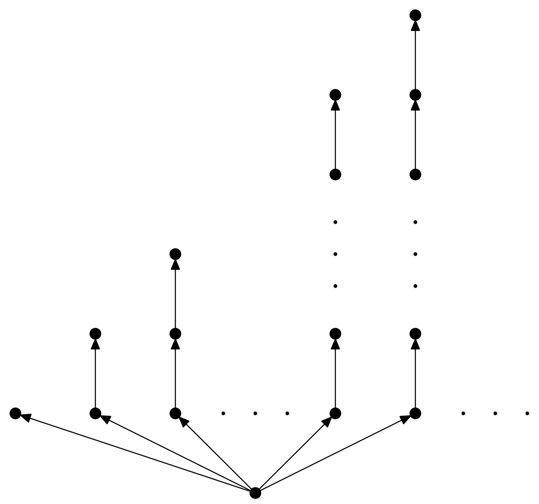


Figure 2.2: A well-founded tree with rank ω

Proof

1. By lemmas 2.1.15 and 2.1.3:

$$Len(A, \subseteq) = \bigcup\{Succ(Rank(B, \in)) \mid B \in A\} = \bigcup\{Succ(B) \mid B \in A\} = A$$

2. Define:

$$X \stackrel{\text{def}}{=} \{B \in Succ(A) \mid \text{the rank of } Tree(B) \neq B\}$$

If $X = \emptyset$ we are done. For a contradiction, suppose $X \neq \emptyset$. By Foundation, there exists $C \in X$ such that $C \cap X = \emptyset$ and the rank of $Tree(C) \neq C$. The rank of $Tree(C)$ is:

$$\bigcup\{Succ(Rank(\langle C_0, C_1, \ldots, C_n\rangle, \sqsubset)) \mid n \in \omega \wedge C_0 \in C \wedge \forall i < n. C_{i+1} \in C_i\}$$

For any chain $\langle C_0, C_1, \ldots, C_n\rangle$ that is an element of $Tree(C)$, we can consider its up-set in $Tree(C)$, i.e., the chains with prefix $\langle C_0, C_1, \ldots, C_n\rangle$. With the prefix order, the up-set is a tree and is order-isomorphic to $Tree(C_n)$. Now C is an ordinal, so $C_n \in C$, and thus $C_n \notin X$. Therefore the rank of $Tree(C_n) = C_n$. It can be shown that the rank of a tree is an invariant under order-isomorphism, so $Rank(\langle C_0, C_1, \ldots, C_n\rangle, \sqsubset)$ is also C_n. Then the rank of $Tree(C)$ is:

$$\begin{aligned} & \bigcup\{Succ(C_n) \mid n \in \omega \wedge C_0 \in C \wedge \forall i < n. C_{i+1} \in C_i\} \\ = \; & \bigcup\{Succ(B) \mid B \in C\} \end{aligned}$$

By lemma 2.1.3, this is equal to C. Thus we have the contradiction that the rank of $Tree(C) = C$. ☐

2.2 Transition Systems

Computational systems can often be modelled as a collection of states together with a description of the circumstances in which one state can evolve to another (see [Plo81, Mil89]).

Definition 2.2.1

1. A ***transition system*** (TS) $\langle S, \rightarrow \rangle$ consists of a set of states S and a transition relation $\rightarrow \subseteq S \times S$.

2. A ***labelled transition system*** (LTS) $\langle S, A, \rightarrow \rangle$ consists of a set of states S, a set of labels A, and a labelled transition relation $\rightarrow \subseteq S \times A \times S$. For states s, t and a label $a \in A$, we write $s \xrightarrow{a} t$ for $\langle s, a, t \rangle \in \rightarrow$. We sometimes use $s \rightarrow t$ to mean there exists $a \in A$ such that $s \xrightarrow{a} t$.

In chapters 3 and 4 we see examples of these structures as components of the operational semantics of the non-deterministic λ-calculus $\mathscr{L}$. For now, we consider the relationships between TSs, LTSs, and trees.

Example 2.2.2 A TS $\langle S, \rightarrow \rangle$ determines an LTS $\langle S, \{\star\}, \rightarrow \rangle$ with a singleton set of labels $\{\star\}$, where an unlabelled transition $s \rightarrow t$ corresponds to a labelled transition $s \xrightarrow{\star} t$. Conversely, an LTS $\langle S, A, \rightarrow \rangle$ determines a TS $\langle S, \rightarrow \rangle$, where $s \rightarrow t$ if and only there exists a label $a \in A$ such that $s \xrightarrow{a} t$. This TS simply forgets the labels upon transitions. Alternatively, we can define a TS $\langle S + (A \times S) + A, \rightarrow \rangle$ that does encode the labelling information in an LTS $\langle S, A, \rightarrow \rangle$ by adding extra states and transitions to terminal states that represent labels. We assume that S, $A \times S$, and A are pairwise disjoint so that $S \cup (A \times S) \cup A$ can be used for the disjoint union $S + (A \times S) + A$. Then the transition relation is defined, for states $s, t \in S$ and a label $a \in A$, by:

- $s \rightarrow \langle a, t \rangle$ if and only if $s \xrightarrow{a} t$
- $\langle a, t \rangle \rightarrow a$
- $\langle a, t \rangle \rightarrow t$

The TS $\langle S + (A \times S) + A, \rightarrow \rangle$ is not entirely satisfactory as a replacement for the LTS $\langle S, A, \rightarrow \rangle$ because terminal states are usually identified by operationally-defined equivalences, and so the encodings of $s \xrightarrow{a} t$ and $s \xrightarrow{b} t$ could not be distinguished by considering only transitions when $a \neq b$. For bisimilarity (see section 2.4), this could be fixed by adding more transitions and states after each state in A, but we do not pursue this here.

Every tree $\langle A, \leq \rangle$ determines several TSs. For example, $\langle A, \leq \rangle$ and $\langle A, < \rangle$ are both TSs. A more useful TS is obtained via the successor relation (cf. Hasse diagrams [DP90]).

(a) $\in$-transition system (b) $\in$-tree

Figure 2.3: $\in$-transition system and $\in$-tree for $\{\{\emptyset,\{\emptyset\}\},\{\{\emptyset\}\}\}$

Example 2.2.3 A tree $\langle A, \leq \rangle$ determines a TS $\langle A, \rightarrow \rangle$, where $\rightarrow \subseteq A \times A$ is defined, for $a, b \in A$, by $a \rightarrow b$ if and only if b is a successor of a. There are no *cycles* in the resulting TS because the transitive closure of the transition relation is contained in the strict order $<$ and is thus irreflexive. Limit elements and their predecessors in the tree are disconnected in the TS.

A set A determines a TS $TS(A)$ with transition relation $\ni$. The states of $TS(A)$ are those sets that are reachable from A by the reflexive, transitive closure of $\ni$.

Definition 2.2.4 The $\in$-TS of a set A is $TS(A) \stackrel{\text{def}}{=} \langle B, \ni \rangle$, where B is defined by:

$$B \stackrel{\text{def}}{=} \{C \mid \exists n \in \omega, A_0, A_1, \ldots, A_n. A = A_0 \wedge C = A_n \wedge \forall i < n. A_{i+1} \in A_i\}$$

In general, the $\in$-TS $TS(A)$ is not the same as the TS determined by the $\in$-tree $Tree(A)$ using example 2.2.3, because the latter has more states. This is illustrated in example 2.2.5.

Example 2.2.5 Consider the set $A \stackrel{\text{def}}{=} \{\{\emptyset,\{\emptyset\}\},\{\{\emptyset\}\}\}$. The $\in$-TS $TS(A)$ and $\in$-tree $Tree(A)$ associated with A are illustrated in figure 2.3.

Recall that, for any set A, the $\in$-tree $Tree(A)$ is well-founded. By Foundation, the $\in$-TS $TS(A)$ also has the property that there are no ω-chains with respect to the transition relation $\ni$. However, the well-founded order $Tree(\omega)$ differs from $TS(\omega)$ in that the latter does have ω-chains with respect to $\in$, the dual of the transition relation $\ni$ (see example 2.1.8), so the transition relation is not well-founded.

We have seen how to obtain TSs from trees. In the other direction, there are several reasons why a TS $\langle S, \rightarrow \rangle$ may fail to be a tree (assuming a state has been chosen as the root):

- The transition relation $\rightarrow$ may not be reflexive or transitive.
- There may be states s_1, s_2, t such that $s_1 \rightarrow t$ and $s_2 \rightarrow t$ but there are no paths from s_1 to s_2 and vice-versa, i.e., the down-set of t is not totally ordered. The $\in$-TS in figure 2.3 provides an example.

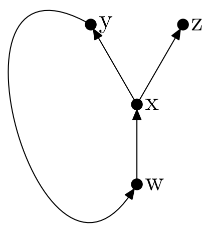


Figure 2.4: The unlabelled transition system associated with a non-well-founded set

- The transition relation $\rightarrow$ may not be well-founded. For example, the transition relation of $TS(\omega)$ is not well-founded.

Each TS with a distinguished root state determines a tree called a *synchronisation tree* (see [WN95]). The elements of the synchronisation tree are paths in the TS from the distinguished state, and are ordered with the prefix order.

Definition 2.2.6 Consider a TS $\langle S, \rightarrow \rangle$ and a state $s \in S$. The ***synchronisation tree*** of $\langle S, \rightarrow \rangle$ rooted at s is $ST(\langle S, \rightarrow \rangle, s) = \langle A, \sqsubseteq \rangle$ where $\sqsubseteq$ is the prefix order and A is defined by:

$$A \stackrel{\text{def}}{=} \{\langle\rangle\} \cup \{\langle s_0, s_1, \ldots, s_n \rangle \mid n \in \omega \wedge s \rightarrow s_0 \wedge \forall i < n.\, s_i \rightarrow s_{i+1}\}$$

It is easy to see that $\in$-trees are the synchronisation trees of $\in$-TSs, i.e., for all sets A, $TS(A) = ST(Tree(A), A)$. A synchronisation tree is obtained by unfolding a TS. A state in the TS will be duplicated in the synchronisation tree if there are multiple paths to it from the root state. This can be seen in figure 2.3. Synchronisation trees do not have limit points, but need not be well-founded. In particular, cycles in the transition system are unfolded to create ω-chains in the synchronisation tree. For example, the synchronisation tree of a TS $\langle \{\star\}, \rightarrow \rangle$, where $\star \rightarrow \star$, is not well-founded.

Example 2.2.7 Forti and Honsell [FH83] and Aczel [Acz88] consider an axiom for set theory that contradicts and replaces the axiom of Foundation (see also [FHL94, BM96]). The Anti-Foundation axiom asserts the existence of *non-well-founded sets* (also known as *hypersets*), upon which the membership relation $\in$ is not well-founded. For example, the following equations define a non-well-founded set w:

$$w = \{x\} \qquad x = \{y, z\} \qquad y = \{w\} \qquad z = \emptyset$$

The definitions of $\in$-trees and $\in$-TSs can be replayed for non-well-founded sets. The $\in$-TS of w is depicted in figure 2.4. Non-well-founded sets determine non-well-founded $\in$-trees, but they do not have limit elements.

We now consider TSs and LTSs with a *may divergence* predicate that partitions the states into those that *may diverge* and those that *must converge*.

Definition 2.2.8 A ***transition system with divergence*** (TSWD) $\langle S, \Uparrow^{\text{may}}, \rightarrow \rangle$ is a TS with a ***may divergence*** predicate $\Uparrow^{\text{may}} \subseteq S$. Similarly, a ***labelled transition system with divergence*** (LTSWD) $\langle S, A, \Uparrow^{\text{may}}, \rightarrow \rangle$ is an LTS with a ***may divergence*** predicate $\Uparrow^{\text{may}} \subseteq S$. The derived ***must convergence*** predicate $\Downarrow^{\text{must}}$ is the complement of $\Uparrow^{\text{may}}$, i.e., $s \in \Downarrow^{\text{must}}$ if and only if $s \notin \Uparrow^{\text{may}}$. For a state $s \in S$, we write $s \Uparrow^{\text{may}}$ for $s \in \Uparrow^{\text{may}}$, and $s \Downarrow^{\text{must}}$ for $s \in \Downarrow^{\text{must}}$.

Abramsky [Abr87b] and Walker [Wal90] consider LTSWDs that are derived from other LTSWDs that include the distinguished label τ. If we suppose that no states may diverge in a LTSWD, then it is just an LTS. In this case, a state in the derived system may diverge if it possible to begin an infinite sequence of τ-labelled transitions from that state. The transitions in the LTSWD are derived from those of the LTS by ignoring certain τ-labelled transitions.

Example 2.2.9 builds upon definition 2.2.4 to show that sets with a distinguished urelement $\perp$ form TSWDs. This example is extended in chapter 4 to *typed transition systems* which are a special case of LTSWDs.

Example 2.2.9 Consider sets (well-founded or non-well-founded) constructed using a single urelement $\perp$. For such a set A, the definition of the $\in$-TS $\mathit{TS}(A)$ can be replayed by not treating $\perp$ as a state. The may divergence predicate of the $\in$-TSWD is defined, for a state B of $\mathit{TS}(A)$, by $B \Uparrow^{\text{may}}$ if and only $\perp \in B$. We write $\mathit{TSWD}(A)$ for this $\in$-TSWD.

As an example, consider the states of the $\in$-TSWD $\mathit{TSWD}(\{\emptyset, \{\emptyset\}, \{\perp, \emptyset\}\})$:

$$\{\emptyset, \{\emptyset\}, \{\perp, \emptyset\}\} \qquad \{\perp, \emptyset\} \qquad \{\emptyset\} \qquad \emptyset$$

The only state that may diverge is $\{\perp, \emptyset\} \Uparrow^{\text{may}}$. All of the other states must converge.

2.3 Induction and Coinduction

Many of the definitions and proofs for operational semantics are inductive or coinductive. In this section we recall the fundamentals of induction and coinduction, and then consider some definitions and results that are of particular use for operational semantics.

We present induction and coinduction abstractly in the framework of order theory (see [DP90] for an excellent introduction to this subject). There is also an appealing category-theoretic account of induction and coinduction (see [JR97]) that generalises the order-theoretic version given below: categories generalise partial orders; functors generalise monotone functions; algebras and coalgebras generalise post-fixed-points and pre-fixed-points; and initiality and finality generalise least and greatest fixed-points. However, the extra generality is not required here.

Definition 2.3.1 Let $\langle A, \leq \rangle$ be a partial order. If $F : A \rightarrow A$ is a monotone function, then an element $a \in A$ is a ***pre-fixed-point*** if $a \leq F(a)$, and a ***post-fixed-point***[1] if $F(a) \leq a$. If it exists, the ***meet*** of a set $B \subseteq A$, is the unique element of A, written $\bigsqcap B$, such that, for all $a \in A$, $a \leq \bigsqcap B$ if and only if, for all $b \in B$, $a \leq b$. If it exists, the ***join*** of a set $B \subseteq A$, is the unique element of

[1] Warning: in [Gun92], this is the definition of a pre-fixed-point.

A, written $\bigsqcup B$, such that, for all $a \in A$, $\bigsqcup B \leq a$ if and only if, $\forall b \in B$, $b \leq a$. We write $a \sqcap b$ for $\bigsqcap\{a,b\}$, $a \sqcup b$ for $\bigsqcup\{a,b\}$, $\top$ for $\bigsqcap\emptyset$, and $\bot$ for $\bigsqcup\emptyset$. The partial order $\langle A, \leq\rangle$ is a ***complete lattice*** if meets exist for all subsets of A. If $\langle A, \leq\rangle$ is a complete lattice, then elements $a, b \in A$ are ***complements*** if $a \sqcap b = \bot$ and $a \sqcup b = \top$. We write $\overline{a}$ for the complement of $a \in A$ when it exists and it is the unique complement. A complete lattice $\langle A, \leq\rangle$ satisfies the ***meet-infinite distributive law*** and ***join-infinite distributive law*** respectively if, for all $a \in A$ and $B \subseteq A$:

$$a \sqcup \bigsqcap\{b \mid b \in B\} = \bigsqcap\{a \sqcup b \mid b \in B\} \qquad \text{(meet-infinite distributive)}$$

$$a \sqcap \bigsqcup\{b \mid b \in B\} = \bigsqcup\{a \sqcap b \mid b \in B\} \qquad \text{(join-infinite distributive)}$$

It is straightforward to show that all joins exist in a complete lattice $\langle A, \leq\rangle$ because they can be defined using meets: if $B \subseteq A$, then:

$$\bigsqcup B = \bigsqcap\{a \in A \mid \forall b \in B.\, b \leq a\}$$

Powersets ordered by set inclusion form an important class of complete lattices. In particular, the similarity and bisimilarity relations defined in the sequel are constructed inside complete lattices of this form.

Example 2.3.2 For any set A, the partial order $\langle \mathcal{P}(A), \subseteq\rangle$ is a complete lattice that satisfies the meet-infinite and join-infinite distributive laws. The meet operation on non-empty sets is set intersection. The meet of the empty set is A. The join operation is set union. Unique complements exist for all $B \in \mathcal{P}(A)$, and are given by $\overline{B} = A \setminus B$.

Complete lattices admit inductive and coinductive definitions as the least and greatest fixed-points (respectively) of monotone functions. The Knaster-Tarski theorem tells us that the least and greatest fixed-points always exist. Moreover, the least fixed-point is the meet of all post-fixed-points and the greatest fixed-point is the join of all pre-fixed-points.

Theorem 2.3.3 (Knaster-Tarski) If $\langle A, \leq\rangle$ is a complete lattice and $F : A \to A$ is monotone, then the least and greatest fixed-points of F are respectively:

$$\mu a.\, F(a) \stackrel{\text{def}}{=} \bigsqcap\{a \in A \mid F(a) \leq a\}$$

$$\nu a.\, F(a) \stackrel{\text{def}}{=} \bigsqcup\{a \in A \mid a \leq F(a)\}$$

Proof We first show that $\mu a.\, F(a)$ is a fixed-point. The proof for $\nu a.\, F(a)$ is dual. We claim $F(\mu a.\, F(a)) \leq \mu a.\, F(a)$. If $b \in A$ is such that $F(b) \leq b$, then $\mu a.\, F(a) \leq \bigsqcap\{a \in A \mid F(a) \leq a\} \leq b$. By monotonicity of F, $F(\mu a.\, F(a)) \leq F(b)$. Hence, $F(\mu a.\, F(a)) \leq b$, and so $F(\mu a.\, F(a)) \leq \bigsqcap\{a \in A \mid F(a) \leq a\} = \mu a.\, F(a)$. For the reverse inequality, note that $F(F(\mu a.\, F(a))) \leq F(\mu a.\, F(a))$ by monotonicity. Therefore:

$$\mu a.\, F(a) = \bigsqcap\{a \in A \mid F(a) \leq a\} \leq F(\mu a.\, F(a))$$

and it follows that $\mu a.\, F(a) = F(\mu a.\, F(a))$ as required. By duality, we see that $\nu a.\, F(a)$ is a fixed-point. Finally we claim that for any fixed-point $b \in A$ of F, we have $\mu a.\, F(a) \leq b \leq \nu a.\, F(a)$. This follows immediately from the universal properties of meet and join given that $b = F(b)$ is a post-fixed-point and a pre-fixed-point of F. $\square$

Park [Par79, Par81] and Milner [Mil89] show the importance of coinduction as a technique for defining semantic relations upon LTSs (see section 2.4 and chapter 4).

The term coinductive was in use by 1974. Moschovakis [Mos74] describes a set as coinductive if it is the complement of an inductively-defined set (as opposed to the greatest fixed-point of a monotone function). Lemma 2.3.5 shows that these definitions coincide for a certain class of complete lattices. Aczel [Acz77] uses the term *kernel* for the greatest fixed-point of a monotone function.

The following induction and coinduction principles, and their strong variants, can be deduced from the Knaster-Tarski theorem.

Lemma 2.3.4 If $\langle A, \leq \rangle$ is a complete lattice and $F : A \rightarrow A$ is monotone, then the following induction and coinduction principles, and their strong variants, are valid for all $b \in A$:

$$F(b) \leq b \Longrightarrow \mu a.F(a) \leq b \qquad \text{(Induction)}$$

$$b \leq F(b) \Longrightarrow b \leq \nu a.F(a) \qquad \text{(Coinduction)}$$

$$(F(b \sqcap \mu a.F(a)) \sqcap \mu a.F(a)) \leq b \Longrightarrow \mu a.F(a) \leq b \qquad \text{(Strong Induction)}$$

$$b \leq (F(b \sqcup \nu a.F(a)) \sqcup \nu a.F(a)) \Longrightarrow b \leq \nu a.F(a) \qquad \text{(Strong Coinduction)}$$

Proof The induction and coinduction principles follow immediately from the universal property defining the meet and join operators. We prove the strong coinduction principle. The strong induction principle is dual. First show that:

$$b \sqcup \nu a.F(a) \leq F(b \sqcup \nu a.F(a))$$

This follows from the hypothesis and:

$$\nu a.F(a) = F(\nu a.F(a)) \leq F(b \sqcup \nu a.F(a))$$

Now, by the ordinary coinduction principle:

$$b \sqcup \nu a.F(a) \leq \nu a.F(a)$$

Therefore $b \leq \nu a.F(a)$ □

In the remainder of this section we derive results, concerning induction and coinduction, that are used for reasoning about the operational semantics and semantic relations considered in the sequel.

Lemma 2.3.5 appears in [Acz77, Lev79], and justifies the use of the term coinductive for both the complement of an inductively-defined set and the greatest fixed-point of a monotone function. It is used in chapter 3 to show that the inductively-defined must convergence predicate on programs is the complement of the coinductively-defined may divergence predicate.

Lemma 2.3.5 Let $\langle A, \leq \rangle$ be a complete lattice satisfying the meet-infinite distributive and join-infinite distributive laws, and $F : A \rightarrow A$ a monotone function. If complements exist for every element in A, then the complements are unique and:

$$\mu a.\overline{F(\overline{a})} = \overline{\nu a.F(a)}$$

$$\nu a.\overline{F(\overline{a})} = \overline{\mu a.F(a)}$$

Proof If $b, c \in A$ are complements of $a \in A$ then:

$$b \sqcup c = \top \sqcap (b \sqcup c) = (a \sqcup c) \sqcap (b \sqcup c) = (a \sqcap b) \sqcup c = \bot \sqcup c = c$$

Similarly, $b \sqcup c = b$ and therefore complements are unique. Consequently, $\overline{\overline{a}} = a$ for all $a \in A$. Note that we only require distributivity over finite meets and joins for unique complements. Next we prove that the following equalities hold for all $B \subseteq A$:

$$\overline{\bigsqcup\{b \mid b \in B\}} = \bigsqcap\{\overline{b} \mid b \in B\} \qquad \text{and} \qquad \overline{\bigsqcap\{b \mid b \in B\}} = \bigsqcup\{\overline{b} \mid b \in B\}$$

For the first, we need to show that $\bigsqcup\{a \mid a \in B\}$ and $\bigsqcap\{\overline{b} \mid b \in B\}$ are complements:

$$\begin{aligned} \bigsqcup\{a \mid a \in B\} \sqcap \bigsqcap\{\overline{b} \mid b \in B\} &= \bigsqcup\{a \sqcap \bigsqcap\{\overline{b} \mid b \in B\} \mid a \in B\} \\ &= \bigsqcup\{\bigsqcap\{a \sqcap \overline{b} \mid b \in B\} \mid a \in B\} \\ &= \bigsqcup\{\bot \mid a \in B\} \\ &= \bot \end{aligned}$$

Similarly, $\bigsqcup\{a \mid a \in B\} \sqcup \bigsqcap\{\overline{b} \mid b \in B\} = \top$ and so they are complements. The second equality follows by duality. From these equalities we deduce that, for all $a, b \in A$, if $a \leq b$, then $\overline{b} \leq \overline{a}$. This is because $a = a \sqcap b$ implies $\overline{a} = \overline{a \sqcap b} = \overline{a} \sqcup \overline{b}$, which in turns implies $\overline{b} \leq \overline{a}$. Now we can prove the main results. For $\mu a. \overline{F(\overline{a})} = \overline{\nu a. F(a)}$, the above properties imply that:

$$\begin{aligned} \mu a. \overline{F(\overline{a})} &= \bigsqcap\{a \in A \mid \overline{F(\overline{a})} \leq a\} \\ &= \bigsqcap\{a \in A \mid \overline{a} \leq F(\overline{a})\} \\ &= \bigsqcap\{\overline{a} \in A \mid a \leq F(a)\} \\ &= \overline{\bigsqcup\{a \in A \mid a \leq F(a)\}} \\ &= \overline{\nu a. F(a)} \end{aligned}$$

The case for $\nu a. \overline{F(\overline{a})} = \overline{\mu a. F(a)}$ is dual. □

The next four lemmas are used later when we work with relations defined by coinduction. Lemma 2.3.6 identifies sufficient conditions for a coinductively-defined relation to be reflexive, symmetric, and transitive.

Lemma 2.3.6 For a set A, consider the complete lattice $\langle \mathcal{P}(A \times A), \subseteq \rangle$ and a monotone function $F : \mathcal{P}(A \times A) \rightarrow \mathcal{P}(A \times A)$. If F preserves reflexivity (respectively symmetry, transitivity) of a relation, then the greatest fixed-point of F is reflexive (respectively symmetric, transitive).

Proof

1. Because F preserves reflexivity and $Id(A) \subseteq Id(A)$, we have $Id(A) \subseteq F(Id(A))$, i.e., $Id(A)$ is a pre-fixed-point of F. Therefore $Id(A) \subseteq \nu\mathcal{R}. F(\mathcal{R})$.

2. Consider a pre-fixed-point of F, $\mathcal{R} \subseteq A \times A$. Now $\mathcal{R} \cup \mathcal{R}^{\mathrm{op}}$ is symmetric, and so $F(\mathcal{R} \cup \mathcal{R}^{\mathrm{op}})$ is also symmetric by hypothesis. By monotonicity of F, we have $F(\mathcal{R}) \subseteq F(\mathcal{R} \cup \mathcal{R}^{\mathrm{op}})$. Then we can deduce that $\mathcal{R} \cup \mathcal{R}^{\mathrm{op}}$ is a pre-fixed-point of F:

$$\mathcal{R} \cup \mathcal{R}^{\mathrm{op}} \subseteq F(\mathcal{R}) \cup (F(\mathcal{R}))^{\mathrm{op}} \subseteq F(\mathcal{R} \cup \mathcal{R}^{\mathrm{op}}) \cup (F(\mathcal{R} \cup \mathcal{R}^{\mathrm{op}}))^{\mathrm{op}} = F(\mathcal{R} \cup \mathcal{R}^{\mathrm{op}})$$

The result follows by taking $\mathcal{R}$ to be $\nu\mathcal{R}.F(\mathcal{R})$.

3. Consider a pre-fixed-point of F, $\mathcal{R} \subseteq A \times A$. Now $\mathcal{R}^+$ is transitive and so $F(\mathcal{R}^+)$ is also transitive by hypothesis. By monotonicity of F, we have $F(\mathcal{R}) \subseteq F(\mathcal{R}^+)$. Then we can deduce that $\mathcal{R}^+$ is a pre-fixed-point of F:

$$\mathcal{R}^+ \subseteq (F(\mathcal{R}))^+ \subseteq (F(\mathcal{R}^+))^+ \subseteq F(\mathcal{R}^+)$$

The result follows by taking $\mathcal{R}$ to be $\nu\mathcal{R}.F(\mathcal{R})$.

□

Lemma 2.3.7 shows that the dual of a least or greatest fixed-point relation is, respectively, a least or greatest fixed-point.

Lemma 2.3.7 For a set A, consider the complete lattice $\langle \mathcal{P}(A \times A), \subseteq \rangle$. If $F : \mathcal{P}(A \times A) \to \mathcal{P}(A \times A)$ is monotone, then:

$$\mu\mathcal{R}.(F(\mathcal{R}^{\mathrm{op}}))^{\mathrm{op}} = (\mu\mathcal{R}.F(\mathcal{R}))^{\mathrm{op}}$$
$$\nu\mathcal{R}.(F(\mathcal{R}^{\mathrm{op}}))^{\mathrm{op}} = (\nu\mathcal{R}.F(\mathcal{R}))^{\mathrm{op}}$$

Proof For the first equality:

$$\begin{aligned}
\mu\mathcal{S}.(F(\mathcal{S}^{\mathrm{op}}))^{\mathrm{op}} &= \textstyle\bigsqcap\{\mathcal{S} \mid (F(\mathcal{S}^{\mathrm{op}}))^{\mathrm{op}} \subseteq \mathcal{S}\} \\
&= \textstyle\bigsqcap\{\mathcal{R}^{\mathrm{op}} \mid (F(\mathcal{R}))^{\mathrm{op}} \subseteq \mathcal{R}^{\mathrm{op}}\} \\
&= \textstyle\bigsqcap\{\mathcal{R}^{\mathrm{op}} \mid F(\mathcal{R}) \subseteq \mathcal{R}\} \\
&= (\textstyle\bigsqcap\{\mathcal{R} \mid F(\mathcal{R}) \subseteq \mathcal{R}\})^{\mathrm{op}} \\
&= (\mu\mathcal{R}.F(\mathcal{R}))^{\mathrm{op}}
\end{aligned}$$

The second equality is obtained by duality. □

Lemma 2.3.8 and corollary 2.3.9 are used to show that variations of bisimilarity are always included in the corresponding mutual similarity.

Lemma 2.3.8 Let $\langle A, \leq \rangle$ be a complete lattice and $F, G \in A \to A$ monotone functions. If $F(a) \leq G(a)$, for all $a \in A$, then:

$$\mu a.F(a) \leq \mu a.G(a)$$
$$\nu a.F(a) \leq \nu a.G(a)$$

Proof For the first inequality, it suffices to show that $\mu a. G(a)$ is a post-fixed-point for F, i.e., $F(\mu a. G(a)) \leq \mu a. G(a)$. This holds because F is bounded by G, and $\mu a. G(a)$ is a fixed-point, so:

$$F(\mu a. G(a)) \leq G(\mu a. G(a)) = \mu a. G(a)$$

The proof for the second inequality is similar. □

Corollary 2.3.9 Let $\langle A, \leq \rangle$ be a complete lattice and $F, G : A \rightarrow A$ monotone functions. Then:

$$\begin{aligned}
&\mu a. F(a) \sqcap G(a) \leq (\mu a. F(a)) \sqcap (\mu a. G(a)) \\
&\nu a. F(a) \sqcap G(a) \leq (\nu a. F(a)) \sqcap (\nu a. G(a))
\end{aligned}$$

Proof In both cases, apply lemma 2.3.8 twice. □

Lemma 2.3.10 is an "up to" result for coinductively-defined relations (see [Mil89, Gor95a, Las98a]). This result is used to simplify proofs that elements are related by a coinductively-defined relation. It is used in chapter 5 to prove Scott induction principles for coinductively-defined preorders (theorem 5.7.9).

Lemma 2.3.10 For a set A, consider the complete lattice $\langle \mathcal{P}(A \times A), \subseteq \rangle$ and a monotone function $F : \mathcal{P}(A \times A) \rightarrow \mathcal{P}(A \times A)$ such that $F(\mathcal{R}); F(\mathcal{S}) \subseteq F(\mathcal{R}; \mathcal{S})$, whenever $\mathcal{R}, \mathcal{S} \subseteq A \times A$. If $\mathcal{T} = \nu \mathcal{R}. F(\mathcal{R})$, then, for any $\mathcal{S} \subseteq A \times A$:

$$\mathcal{S} \subseteq F(\mathcal{T}; \mathcal{S}; \mathcal{T}) \Longrightarrow \mathcal{S} \subseteq \mathcal{T}$$

Proof Assume $\mathcal{S} \subseteq F(\mathcal{T}; \mathcal{S}; \mathcal{T})$. We first establish that $\mathcal{T}; \mathcal{S}; \mathcal{T} \subseteq T$. By coinduction, this follows from $\mathcal{T}; \mathcal{S}; \mathcal{T} \subseteq F(\mathcal{T}; \mathcal{S}; \mathcal{T})$, which holds because:

$$\begin{aligned}
\mathcal{T}; \mathcal{S}; \mathcal{T} &\subseteq \mathcal{T}; F(\mathcal{T}; \mathcal{S}; \mathcal{T}); \mathcal{T} \\
&\subseteq F(\mathcal{T}); F(\mathcal{T}; \mathcal{S}; \mathcal{T}); F(\mathcal{T}) \\
&\subseteq F(\mathcal{T}; \mathcal{T}; \mathcal{S}; \mathcal{T}; \mathcal{T}) \\
&\subseteq F(\mathcal{T}; \mathcal{S}; \mathcal{T})
\end{aligned}$$

The last line requires transitivity of $\mathcal{T}$. To prove transitivity, by lemma 2.3.6, it suffices to show that, for any $\mathcal{R} \subseteq A \times A$, $\mathcal{R}; \mathcal{R} \subseteq \mathcal{R}$ implies that $F(\mathcal{R}); F(\mathcal{R}) \subseteq F(\mathcal{R})$. This follows from the hypotheses because $F(\mathcal{R}); F(\mathcal{R}) \subseteq F(\mathcal{R}; \mathcal{R}) \subseteq F(\mathcal{R})$. Therefore we know $\mathcal{T}; \mathcal{S}; \mathcal{T} \subseteq T$. The result follows by applying the strong coinduction principle to $\mathcal{S} \subseteq F(\mathcal{T}; \mathcal{S}; \mathcal{T}) \subseteq F(\mathcal{T})$. Therefore $\mathcal{S} \subseteq \mathcal{T}$. □

Finally, lemma 2.3.11 identifies a condition under which the least and greatest fixed-points of a monotone function are the same (and thus are the unique fixed-point). Informally, we may consider a complete lattice of the form $\langle \mathcal{P}(A), \subseteq \rangle$ with a well-founded relation $\mathcal{R} \subseteq A \times A$. If $F : A \to A$ is a monotone function generated from rules where every premise of a rule is related to the conclusion by $\mathcal{R}$, then every coinductive proof must be a well-founded tree and thus an inductive proof.

This result is used to show that the variants of similarity and bisimilarity defined in chapter 4 are both least and greatest fixed-points because they are defined for LTSs with a type system that lacks recursive or coinductive types. The well-founded relation in this case is the order on the size of the type of a state.

Lemma 2.3.11 For sets A and $X \subseteq \mathcal{P}(A)$, suppose that $\langle X, \subseteq \rangle$ is a complete lattice. Let $F : X \to X$ be monotone with respect to the inclusion order and $\mathcal{R} \subseteq A \times A$ a well-founded relation such that, for all $a \in \nu B.\, F(B)$, there exists $C \subseteq \nu B.\, F(B)$ such that $a \in F(C)$ and $C \mathcal{R} a$, where $C \mathcal{R} a$ means that, for all $c \in C$, $c \mathcal{R} a$. Then:

$$\mu B.\, F(B) = \nu B.\, F(B)$$

Proof We know $\mu B.\, F(B) \subseteq \nu B.\, F(B)$, and so it suffices to show the reverse inclusion, i.e., $a \in \nu B.\, F(B)$ implies $a \in \mu B.\, F(B)$. The proof is by well-founded induction on a with respect to $\mathcal{R}$. If $a \in \nu B.\, F(B)$, then by assumption there exists $C \subseteq \nu B.\, F(B)$ such that $a \in F(C)$ and $C \mathcal{R} a$. Applying the induction hypothesis to the elements of C yields that $C \subseteq \mu B.\, F(B)$. Therefore, by monotonicity, $a \in F(C) \subseteq F(\mu B.\, F(B)) = \mu B.\, F(B)$. □

2.4 Similarity and Bisimilarity

Park [Par79, Par81] and Milner [Mil89] introduce bisimilarity as an equivalence relation upon processes. Bisimilarity is a coinductively-defined relation upon the states of an LTS, and processes are equivalent if the initial states of the LTSs that they determine are related by bisimilarity. It is also possible to define a preorder called similarity in the same style as bisimilarity. Informally, a state s is related to a state t by similarity if the possible behaviours of s and the states that can be reached via the transition relation are "dominated" by those of t.

The concepts underlying bisimilarity have proven robust. For example, there is a treatment of bisimilarity for CSP (see [BRW88, Ros98]), there are many variants of similarity and bisimilarity for LTSs and related structures such as LTSWDs (see [Abr87b, Wal90, Van90, Abr91, Van93]), and *applicative similarity* and *applicative bisimilarity* can be used to reason about λ-calculi even in the presence of recursive types (see [Abr90, Gor95b]).

The definitions in this section are for LTSs and LTSWDs. They also apply to TSs and TSWD using example 2.2.2.

Similarity for an LTS is defined as the greatest fixed-point of a monotone simulation function $\langle \cdot \rangle_{\mathrm{S}}$ on the set of binary relations on the states. Bisimilarity is also defined in terms of the simulation function.

Definition 2.4.1 Consider an LTS $\langle S, A, \rightarrow\rangle$ and a relation on the states $\mathcal{R} \subseteq S \times S$. Define the relation $\langle\mathcal{R}\rangle_{\mathrm{S}} \subseteq S \times S$, for $s_1, t_1 \in S$, by:

$$\langle s_1, t_1\rangle \in \langle\mathcal{R}\rangle_{\mathrm{S}} \Longleftrightarrow \forall a \in A. \forall s_2 \in S. s_1 \xrightarrow{a} s_2 \Longrightarrow \exists t_2 \in S. t_1 \xrightarrow{a} t_2 \wedge \langle s_2, t_2\rangle \in \mathcal{R}$$

Now similarity and bisimilarity can be defined using coinduction. In addition, mutual similarity is defined to be the greatest symmetric relation contained in similarity.

Definition 2.4.2 For an LTS $\langle S, A, \rightarrow\rangle$, ***similarity***, ***mutual similarity***, and ***bisimilarity*** are the binary relations on S defined by:

$$\begin{array}{ll} \lesssim_{\mathrm{S}} \stackrel{\text{def}}{=} \nu\mathcal{R}. \langle\mathcal{R}\rangle_{\mathrm{S}} & \text{(similarity)} \\ \simeq_{\mathrm{S}} \stackrel{\text{def}}{=} \lesssim_{\mathrm{S}} \cap \lesssim_{\mathrm{S}}^{\mathrm{op}} & \text{(mutual similarity)} \\ \simeq_{\mathrm{B}} \stackrel{\text{def}}{=} \nu\mathcal{R}. \langle\mathcal{R}\rangle_{\mathrm{S}} \cap \langle\mathcal{R}^{\mathrm{op}}\rangle_{\mathrm{S}}^{\mathrm{op}} & \text{(bisimilarity)} \end{array}$$

If $\langle S, A, \rightarrow\rangle$ is an LTS, then the fact that similarity and bisimilarity are pre-fixed-points (actually fixed-points by theorem 2.3.3) means that for all states $s_1, t_1 \in S$:

$$\begin{array}{l} s_1 \lesssim_{\mathrm{S}} t_1 \Longrightarrow \forall a \in A. \forall s_2 \in S. s_1 \xrightarrow{a} s_2 \Longrightarrow \exists t_2 \in S. t_1 \xrightarrow{a} t_2 \wedge t_2 \lesssim_{\mathrm{S}} s_2 \\ s_1 \simeq_{\mathrm{B}} t_1 \Longrightarrow (\forall a \in A. \forall s_2 \in S. s_1 \xrightarrow{a} s_2 \Longrightarrow \exists t_2 \in S. t_1 \xrightarrow{a} t_2 \wedge s_2 \simeq_{\mathrm{B}} t_2) \wedge \\ \qquad\qquad (\forall a \in A. \forall t_2 \in S. t_1 \xrightarrow{a} t_2 \Longrightarrow \exists s_2 \in S. s_1 \xrightarrow{a} s_2 \wedge s_2 \simeq_{\mathrm{B}} t_2) \end{array}$$

However, these properties do not define similarity or bisimilarity. There are many relations that are pre-fixed-points, including the empty set, but similarity and bisimilarity are the greatest pre-fixed-points.

We can use the results developed in section 2.3 to obtain generic results about similarity and bisimilarity.

Lemma 2.4.3 For any LTS, similarity $\lesssim_{\mathrm{S}}$ is a preorder, and mutual similarity $\simeq_{\mathrm{S}}$ and bisimilarity $\simeq_{\mathrm{B}}$ are equivalence relations.

Proof It is straightforward to show that $\langle\cdot\rangle_{\mathrm{S}}$ preserves reflexivity and transitivity, and that $(\mathcal{R} \mapsto \langle\mathcal{R}\rangle_{\mathrm{S}} \cap \langle\mathcal{R}^{\mathrm{op}}\rangle_{\mathrm{S}}^{\mathrm{op}})$ preserves reflexivity, symmetry, and transitivity. The results follow by lemma 2.3.6 □

It is often necessary to compare states from different LTSs with respect to similarity or bisimilarity. The states can be compared inside the disjoint union of the LTSs. Define the disjoint union of LTSs $\langle S_1, A_1, \rightarrow_1\rangle$ and $\langle S_2, A_2, \rightarrow_2\rangle$ to be $\langle S, A, \rightarrow\rangle$ where $S \stackrel{\text{def}}{=} S_1 + S_2$ and $A \stackrel{\text{def}}{=} A_1 + A_2$. If S_1 and S_2 (respectively A_1 and A_2) are disjoint so that S (respectively A) can be represented by $S_1 \cup S_2$ (respectively $A_1 \cup A_2$), then $\rightarrow \subseteq S \times A \times S$ is $\rightarrow_1 \cup \rightarrow_2$.

Example 2.4.4 shows that a state with a transition to itself and no other states is bisimilar to a state that has an infinite sequence of transitions without cycles.

Example 2.4.4 Consider the TSs $\langle\{\star\},\rightarrow_1\rangle$ and $\langle\omega,\rightarrow_2\rangle$ where the transition relations are defined by $\star\rightarrow_1\star$ and $n\rightarrow_2 n+1$, for all $n\in\omega$. We show that $\star\simeq_{\mathrm{B}} n$, for all $n\in\omega$. First define $\mathcal{S}\stackrel{\text{def}}{=}\{\langle\star,n\rangle \mid n\in\omega\}$. It suffices to show that $\mathcal{S}$ is a pre-fixed-point of $(\mathcal{R}\mapsto\langle\mathcal{R}\rangle_{\mathrm{S}}\cap\langle\mathcal{R}^{\mathrm{op}}\rangle_{\mathrm{S}}^{\mathrm{op}})$, in which case $\langle\star,n\rangle\in\mathcal{S}\subseteq\simeq_{\mathrm{B}}$, for all $n\in\omega$. However, $\mathcal{S}$ is clearly a pre-fixed-point because, for any $\langle\star,n\rangle\in\mathcal{S}$, the only transitions from $\star$ and n are $\star\rightarrow_1\star$ and $n\rightarrow_2 n+1$, and we have $\langle\star,n+1\rangle\in\mathcal{S}$.

The states in these TSs are greater than states in any other TS with respect to similarity. To see this, suppose that $\langle S,\rightarrow_3\rangle$ is a TS and $s\in S$ is a state. Then $s\lesssim_{\mathrm{S}}\star$, because $\star$ always has a transition to itself. Formally, $\mathcal{S}\stackrel{\text{def}}{=}\{\langle s,\star\rangle \mid s\in S\}$ is a pre-fixed-point of $\langle\cdot\rangle_{\mathrm{S}}$.

We now examine the inclusions between the three relations.

Lemma 2.4.5 For any LTS, $\simeq_{\mathrm{B}}\subseteq\simeq_{\mathrm{S}}\subseteq\lesssim_{\mathrm{S}}$.

Proof By definition, $\simeq_{\mathrm{S}}\subseteq\lesssim_{\mathrm{S}}$. By lemma 2.3.7 and corollary 2.3.9, we have:

$$\begin{aligned}\simeq_{\mathrm{B}} &= \nu\mathcal{R}.\,\langle\mathcal{R}\rangle_{\mathrm{S}}\cap\langle\mathcal{R}^{\mathrm{op}}\rangle_{\mathrm{S}}^{\mathrm{op}}\\ &\subseteq(\nu\mathcal{R}.\,\langle\mathcal{R}\rangle_{\mathrm{S}})\cap(\nu\mathcal{R}.\,\langle\mathcal{R}^{\mathrm{op}}\rangle_{\mathrm{S}}^{\mathrm{op}})\\ &=(\nu\mathcal{R}.\,\langle\mathcal{R}\rangle_{\mathrm{S}})\cap(\nu\mathcal{R}.\,\langle\mathcal{R}\rangle_{\mathrm{S}})^{\mathrm{op}}\\ &=\simeq_{\mathrm{S}}\end{aligned}$$

□

Example 2.4.6 shows that the inclusions in lemma 2.4.5 are strict.

Example 2.4.6 Define LTSs $\langle S_1,A,\rightarrow_1\rangle$, $\langle S_2,A,\rightarrow_2\rangle$, and $\langle S_3,A,\rightarrow_3\rangle$, where $A=\{a,b,c\}$, by:

$$\begin{aligned}S_1 &\stackrel{\text{def}}{=}\{\langle\rangle,\langle a,0\rangle,\langle a,1\rangle,\langle a,b\rangle,\langle a,c\rangle\}\\ S_2 &\stackrel{\text{def}}{=}\{\langle\rangle,\langle a\rangle,\langle a,b\rangle,\langle a,c\rangle\}\\ S_3 &\stackrel{\text{def}}{=}\{\langle\rangle,\langle a,0\rangle,\langle a,1\rangle,\langle a,b,0\rangle,\langle a,b,1\rangle,\langle a,c\rangle\}\end{aligned}$$

And the transition relations by:

$$\begin{array}{lllll}\langle\rangle\xrightarrow{a}_1\langle a,0\rangle & \langle\rangle\xrightarrow{a}_1\langle a,1\rangle & \langle a,0\rangle\xrightarrow{b}_1\langle a,b\rangle & \langle a,1\rangle\xrightarrow{c}_1\langle a,c\rangle & \\ \langle\rangle\xrightarrow{a}_2\langle a\rangle & \langle a\rangle\xrightarrow{b}_2\langle a,b\rangle & \langle a\rangle\xrightarrow{c}_2\langle a,c\rangle & & \\ \langle\rangle\xrightarrow{a}_3\langle a,0\rangle & \langle\rangle\xrightarrow{a}_3\langle a,1\rangle & \langle a,0\rangle\xrightarrow{b}_3\langle a,b,0\rangle & \langle a,1\rangle\xrightarrow{b}_3\langle a,b,1\rangle & \langle a,1\rangle\xrightarrow{c}_3\langle a,c\rangle\end{array}$$

These LTSs are illustrated in figure 2.5. Now write $\langle\rangle_i$ for $\langle\rangle\in S_i$, where i is 1, 2, or 3. Recall that they are distinct elements in the disjoint union of the LTSs. The relationships between these states are:

$$\langle\rangle_1\lesssim_{\mathrm{S}}\langle\rangle_2\simeq_{\mathrm{S}}\langle\rangle_3 \qquad \langle\rangle_3\not\simeq_{\mathrm{B}}\langle\rangle_2\not\lesssim_{\mathrm{S}}\langle\rangle_1$$

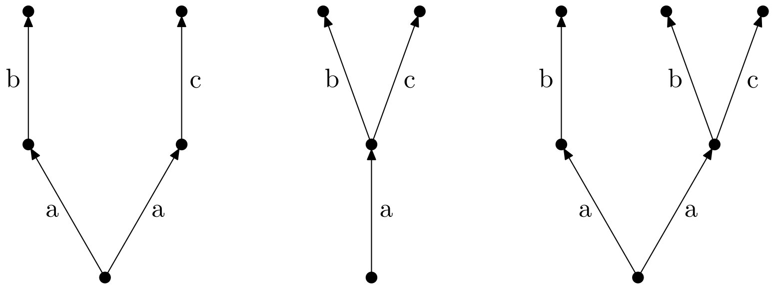


Figure 2.5: LTSs related by similarity

In particular, $\langle\rangle_1$ and $\langle\rangle_2$ are related by similarity but not mutual similarity, and $\langle\rangle_2$ and $\langle\rangle_3$ are related by mutual similarity but not by bisimilarity. Thus mutual similarity is strictly finer than similarity, and bisimilarity is strictly finer than mutual similarity.

Similarity and bisimilarity are defined coinductively (as greatest fixed-points) rather than inductively (as least fixed-points) because there may be infinite paths in an LTS (the dual of the transition relation of a TS may not be well-founded), and in such cases the least fixed-point is unsatisfactory. For example, the least fixed-point of $(\mathcal{R} \mapsto \langle\mathcal{R}\rangle_{\mathrm{S}} \cap \langle\mathcal{R}^{\mathrm{op}}\rangle_{\mathrm{S}}^{\mathrm{op}})$ is irreflexive upon the TS $\langle\{\star\},\rightarrow\rangle$ where $\star \rightarrow \star$. By lemma 2.3.5, the complements of similarity and bisimilarity can be given an inductive definition. Informally, this means that a proof of similarity or bisimilarity may be represented by a potentially non-well-founded derivation tree, whereas a proof of the complement of either similarity or bisimilarity may be represented by a well-founded derivation tree. This is related to the difference between the winning strategies for game-theoretic characterisations of inductively and coinductively-defined sets (see [Acz77, BM96, Sti97]).

In the case of $\in$-TSs obtained from well-founded sets, lemma 2.3.11 and the axiom of Foundation can be used to show that similarity and bisimilarity are also least fixed-points, and so could have been defined by induction. In addition, Extensionality and Foundation can be used to show that bisimilar well-founded sets are equal.

Proposition 2.4.7 Consider (well-founded) sets A and B. We have $A = B$ if and only if $A \simeq_{\mathrm{B}} B$ in the disjoint union of the $\in$-TSs $TS(A)$ and $TS(B)$.

Proof Bisimilarity is reflexive, so we only have to show $A \simeq_{\mathrm{B}} B$ implies $A = B$. We write $C \in TS(A)$ to mean C is a state of the $\in$-TS $TS(A)$. Define:

$$X \stackrel{\mathrm{def}}{=} \{C \in TS(A) \mid \exists D \in TS(B). C \simeq_{\mathrm{B}} D \wedge C \neq D\}$$

We claim that $X = \emptyset$, in which case we are done. For a contradiction, suppose that $X \neq \emptyset$. By Foundation, there exist $E \in TS(A)$ and $F \in TS(B)$ such that $E \cap X = \emptyset$, $E \simeq_{\mathrm{B}} F$, and $E \neq F$. Because $E \cap X = \emptyset$, we know that, for all $C \in E$ and $D \in TS(B) \supseteq F$, if $C \simeq_{\mathrm{B}} D$ then $C = D$. Using $E \simeq_{\mathrm{B}} F$, we have that, for all $C \in E$, there exists $D \in F$ such that $C \simeq_{\mathrm{B}} D$. Therefore $C = D$, so $E \subseteq F$. Similarly, for all $D \in F$, there exists $C \in E$ such that $C \simeq_{\mathrm{B}} D$. Therefore $C = D$, so $F \subseteq E$. Thus $E = F$, contradicting $E \in X$. $\square$

Foundation is required in the above proof, which precludes a similar proof for non-well-founded sets. However, the various approaches to axiomatising non-well-founded sets force the *Super Strong Extensionality Axiom*, which states that sets are equal if they are related by bisimilarity (see [FH83, Acz88, FHL94, BM96]).

Sets are also a rich source of TSs. This is demonstrated by the fact that for any TS and every state that does not have an ω-chain of transitions, there is a bisimilar well-founded set.

Proposition 2.4.8 Consider a TS $\langle S, \rightarrow\rangle$ and a state $s \in S$ such that $s \not\rightarrow^{\omega}$, i.e., there are no chains $\langle s_n \in S \mid n \in \omega\rangle$ such that $s_n \rightarrow s_{n+1}$, for all $n \in \omega$. Then there exists a well-founded set A such that $s \simeq_{\mathrm{B}} A$ in the disjoint union of the TSs $\langle S, \rightarrow\rangle$ and $TS(A)$.

Proof The dual $\rightarrow^{\mathrm{op}}$ of the transition relation $\rightarrow$ is well-founded on $\{t \in S \mid s \rightarrow^* t\}$. For every $t_1 \in S$ such that $s \rightarrow^* t_1$, we use well-founded induction on $\rightarrow^{\mathrm{op}}$ to define the set $f(t_1)$ by:

$$f(t_1) \stackrel{\text{def}}{=} \{f(t_2) \mid t_1 \rightarrow t_2\}$$

It is straightforward to show that for all $t \in S$ such that $s \rightarrow^* t$, $t \simeq_{\mathrm{B}} f(t)$ in the disjoint union of the TSs $\langle S, \rightarrow\rangle$ and $TS(f(t))$. The result follows by taking $A = f(s)$. $\square$

The next result shows that the root of a synchronisation tree of a state in a TS is bisimilar to the state itself. This means that only trees need to be considered when studying bisimilarity invariant properties of TSs.

Proposition 2.4.9 Consider a TS $\langle S, \rightarrow\rangle$ and a state $s \in S$. Then $s \simeq_{\mathrm{B}} \langle\rangle$ in the disjoint union of $\langle S, \rightarrow\rangle$ and the TS determined by the synchronisation tree $ST(\langle S, \rightarrow\rangle, s)$ via the successor relation (see example 2.2.3).

Proof We assume without loss of generality that the TSs are disjoint. Define:

$$\mathcal{R} \stackrel{\text{def}}{=} \{\langle s, \langle\rangle\rangle\} \cup \{\langle s_n, \langle s_0, s_1, \ldots, s_n\rangle\rangle \mid n \in \omega \wedge s \rightarrow s_0 \wedge \forall i < n.\, s_i \rightarrow s_{i+1}\}$$

It is straightforward to show that $\mathcal{R}$ is a pre-fixed-point, and so $s \simeq_{\mathrm{B}} \langle\rangle$. $\square$

We now consider TSWDs and LTSWDs. There are several variants of similarity and bisimilarity when divergence is introduced.

Recall that similarity and bisimilarity for LTSs are defined in terms of $\langle\cdot\rangle_{\mathrm{S}}$. For LTSWDs, the variants of similarity and bisimilarity are generated by a lower simulation function $\langle\cdot\rangle_{\mathrm{LS}}$ and an upper simulation function $\langle\cdot\rangle_{\mathrm{US}}$.

Definition 2.4.10 For an LTSWD $\langle S, A, \Uparrow^{\mathrm{may}}, \rightarrow \rangle$ and a relation on the states $\mathcal{R} \subseteq S \times S$, define the relations $\langle \mathcal{R} \rangle_{\mathrm{LS}}, \langle \mathcal{R} \rangle_{\mathrm{US}} \subseteq S \times S$ by:

$$
\begin{aligned}
\langle s_1, t_1 \rangle \in \langle \mathcal{R} \rangle_{\mathrm{LS}} &\iff \forall a \in A. \forall s_2 \in S. s_1 \xrightarrow{a} s_2 \implies \exists t_2 \in S. t_1 \xrightarrow{a} t_2 \wedge \langle s_2, t_2 \rangle \in \mathcal{R} \\
\langle s_1, t_1 \rangle \in \langle \mathcal{R} \rangle_{\mathrm{US}} &\iff s_1 \Downarrow^{\mathrm{must}} \implies \\
&\qquad (t_1 \Downarrow^{\mathrm{must}} \wedge \\
&\qquad \forall a \in A. \forall t_2 \in S. t_1 \xrightarrow{a} t_2 \implies \exists s_2 \in S. s_1 \xrightarrow{a} s_2 \wedge \langle s_2, t_2 \rangle \in \mathcal{R})
\end{aligned}
$$

For LTSWDs where no states may diverge, and a binary relation on the states R, the lower simulation function $\langle \mathcal{R} \rangle_{\mathrm{LS}}$ is the same as $\langle \mathcal{R} \rangle_{\mathrm{S}}$, and $\langle \mathcal{R}^{\mathrm{op}} \rangle_{\mathrm{US}}^{\mathrm{op}}$ is the same as $\langle \mathcal{R} \rangle_{\mathrm{S}}$.

There are four variants for each of similarity, mutual similarity, and bisimilarity obtained from combinations of the lower and upper simulation functions.

Definition 2.4.11 For an LTSWD $\langle S, A, \Uparrow^{\mathrm{may}}, \rightarrow \rangle$, the lower, upper, convex, and refinement variants of similarity, mutual similarity, and bisimilarity are the binary relations on S defined by:

$$
\begin{aligned}
&\lesssim_{\mathrm{LS}} \stackrel{\mathrm{def}}{=} \nu \mathcal{R}. \langle \mathcal{R} \rangle_{\mathrm{LS}} && \text{(lower similarity)} \\
&\simeq_{\mathrm{LS}} \stackrel{\mathrm{def}}{=} \lesssim_{\mathrm{LS}} \cap \lesssim_{\mathrm{LS}}^{\mathrm{op}} && \text{(mutual lower similarity)} \\
&\simeq_{\mathrm{LB}} \stackrel{\mathrm{def}}{=} \nu \mathcal{R}. \langle \mathcal{R} \rangle_{\mathrm{LS}} \cap \langle \mathcal{R}^{\mathrm{op}} \rangle_{\mathrm{LS}}^{\mathrm{op}} && \text{(lower bisimilarity)} \\
&\lesssim_{\mathrm{US}} \stackrel{\mathrm{def}}{=} \nu \mathcal{R}. \langle \mathcal{R} \rangle_{\mathrm{US}} && \text{(upper similarity)} \\
&\simeq_{\mathrm{US}} \stackrel{\mathrm{def}}{=} \lesssim_{\mathrm{US}} \cap \lesssim_{\mathrm{US}}^{\mathrm{op}} && \text{(mutual upper similarity)} \\
&\simeq_{\mathrm{UB}} \stackrel{\mathrm{def}}{=} \nu \mathcal{R}. \langle \mathcal{R} \rangle_{\mathrm{US}} \cap \langle \mathcal{R}^{\mathrm{op}} \rangle_{\mathrm{US}}^{\mathrm{op}} && \text{(upper bisimilarity)} \\
&\lesssim_{\mathrm{CS}} \stackrel{\mathrm{def}}{=} \nu \mathcal{R}. \langle \mathcal{R} \rangle_{\mathrm{LS}} \cap \langle \mathcal{R} \rangle_{\mathrm{US}} && \text{(convex similarity)} \\
&\simeq_{\mathrm{CS}} \stackrel{\mathrm{def}}{=} \lesssim_{\mathrm{CS}} \cap \lesssim_{\mathrm{CS}}^{\mathrm{op}} && \text{(mutual convex similarity)} \\
&\simeq_{\mathrm{CB}} \stackrel{\mathrm{def}}{=} \nu \mathcal{R}. \langle \mathcal{R} \rangle_{\mathrm{LS}} \cap \langle \mathcal{R} \rangle_{\mathrm{US}} \cap \langle \mathcal{R}^{\mathrm{op}} \rangle_{\mathrm{LS}}^{\mathrm{op}} \cap \langle \mathcal{R}^{\mathrm{op}} \rangle_{\mathrm{US}}^{\mathrm{op}} && \text{(convex bisimilarity)} \\
&\lesssim_{\mathrm{RS}} \stackrel{\mathrm{def}}{=} \nu \mathcal{R}. \langle \mathcal{R}^{\mathrm{op}} \rangle_{\mathrm{LS}}^{\mathrm{op}} \cap \langle \mathcal{R} \rangle_{\mathrm{US}} && \text{(refinement similarity)} \\
&\simeq_{\mathrm{RS}} \stackrel{\mathrm{def}}{=} \lesssim_{\mathrm{RS}} \cap \lesssim_{\mathrm{RS}}^{\mathrm{op}} && \text{(mutual refinement similarity)} \\
&\simeq_{\mathrm{RB}} \stackrel{\mathrm{def}}{=} \nu \mathcal{R}. \langle \mathcal{R}^{\mathrm{op}} \rangle_{\mathrm{LS}}^{\mathrm{op}} \cap \langle \mathcal{R} \rangle_{\mathrm{US}} \cap \langle \mathcal{R} \rangle_{\mathrm{LS}} \cap \langle \mathcal{R}^{\mathrm{op}} \rangle_{\mathrm{US}}^{\mathrm{op}} && \text{(refinement bisimilarity)}
\end{aligned}
$$

It is straightforward to show that the variants of similarity are preorders, and that the variants of mutual similarity and bisimilarity are equivalences using lemma 2.3.6.

The variants are named lower, upper, and convex because of the correspondence with constructions used to obtain the lower (Hoare), upper (Smyth), and convex (Plotkin) powerdomains, e.g., using the Egli-Milner construction on preorders (see [Plo76, Smy78, Plo83, Gun92, AJ94, AC98]). Lassen [Las98b] proposes naming the final variant refinement similarity because it seems to be the most suitable order for refining non-deterministic programs that use ambiguous choice.

By definition, convex bisimilarity and refinement bisimilarity are identical, but, in general, the variants of similarity, mutual similarity, and bisimilarity are distinct. Example 2.4.12 briefly illustrates some of the differences between the variants of similarity on TSWDs arising from well-founded sets with a $\perp$ urelement. The relations are also used in section 2.6 to examine the relationship between different binary choice operators, and are studied in detail in chapter 4.

Example 2.4.12 Amongst the $\in$-TSWDs obtained in example 2.2.9, the following inequalities hold:

$$\begin{array}{ll}
\{\perp\} \lesssim_{\text{LS}} \emptyset \lesssim_{\text{LS}} \{\perp\} & \{\perp\} \lesssim_{\text{LS}} \{\perp,\emptyset\} \not\lesssim_{\text{LS}} \{\perp\} \\
\{\perp\} \lesssim_{\text{US}} \emptyset \not\lesssim_{\text{US}} \{\perp\} & \{\perp\} \lesssim_{\text{US}} \{\perp,\emptyset\} \lesssim_{\text{US}} \{\perp\} \\
\{\perp\} \lesssim_{\text{CS}} \emptyset \not\lesssim_{\text{CS}} \{\perp\} & \{\perp\} \lesssim_{\text{CS}} \{\perp,\emptyset\} \not\lesssim_{\text{CS}} \{\perp\} \\
\{\perp\} \lesssim_{\text{RS}} \emptyset \not\lesssim_{\text{RS}} \{\perp\} & \{\perp\} \not\lesssim_{\text{RS}} \{\perp,\emptyset\} \lesssim_{\text{RS}} \{\perp\}
\end{array}$$

2.5 Recursive Ordinals and Recursive Trees

Some countable well-orders are recursively decidable. Ordinals that are order-isomorphic to such a well-order are called *recursive*, and constitute a down-set (a proper subset) of the countable ordinals. There is a close correspondence between recursive ordinals and well-founded trees for which membership of the underlying set of the tree is decidable. This correspondence is exploited in chapter 3 when non-deterministic operators are classified by countable ordinals associated with derivation trees for the operational semantics. The texts [Rog67, Gir87, Odi89] are good references for recursion theory, recursive ordinals, and recursive trees.

Definition 2.5.1 An ordinal is ***recursive*** if it is order-isomorphic to a recursive well-ordering of a subset of ω, i.e., there exists a set $A \subseteq \omega$, a well-order $\preceq \subseteq A \times A$, and a recursive function $f : \omega \times \omega \to \omega$ such that, for all $m,n \in \omega$, $f(m,n)$ is defined and:

$$f(m,n) = \begin{cases} 0 & \text{if } m \not\preceq \mathrm{n} \vee m \notin A \vee n \notin A \\ 1 & \text{if } m \preceq \mathrm{n} \end{cases}$$

Example 2.5.2 The ordinal ω^ω is recursive. To see this, note that any ordinal $A < \omega^\omega$ has unique *Cantorian normal form* $A = \omega^n \cdot a_n + \omega^{n-1} \cdot a_{n-1} + \cdots + \omega \cdot a_1 + a_0$, where $a_0, a_1, \ldots, a_n$ are natural numbers and $a_n \neq 0$ (see [Pot90]). Hence the ordinals strictly less than ω^ω can be represented as finite sequences of natural numbers, and there is an order-isomorphism between ω^ω and $\langle B, \prec \rangle$, where B is defined by:

$$B \stackrel{\text{def}}{=} \{\langle a_0, a_1, \ldots, a_n \rangle \mid n \in \omega \wedge a_n \neq 0 \wedge \forall i \leq n. a_i \in \omega\}$$

The strict order $\prec \subseteq B \times B$ is defined, for $\vec{a} = \langle a_0, a_1, \ldots, a_m \rangle, \vec{b} = \langle b_0, b_1, \ldots, b_n \rangle \in B$, by:

$$\vec{a} \prec \vec{b} \stackrel{\text{def}}{=} m < n \vee (m = n \wedge \exists i \in \omega. i \leq m \wedge a_i < b_i \wedge \forall j \in \omega. i < j \leq n \Longrightarrow a_j = b_j)$$

A finite sequence of natural numbers can be encoded as a number, so the order $\preceq$ is decidable on such encodings. Therefore ω^ω is a recursive ordinal.

Lemma 2.5.3 If A is a recursive ordinal and $B \in A$, then B is a recursive ordinal.

Proof Let $C \subseteq \omega$, $\preceq_1 \subseteq C \times C$, and $f : \omega \times \omega \to \omega$ be the set, well-order, and recursive function associated with A as in definition 2.5.1. Suppose that $c \in C$ corresponds to B in the order-isomorphism, and define $D \stackrel{\text{def}}{=} \{d \in C \mid d \prec_1 c\}$ and $\preceq_2 \stackrel{\text{def}}{=} \preceq_1 \cap (D \times D)$. Then $\langle D, \preceq_2 \rangle$ is a well-order that is order-isomorphic to B. We can now define the total recursive function $g : \omega \times \omega \to \omega$, for $m, n \in \omega$, by:

$$g(m,n) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } f(m,n) = 0 \vee f(n,c) = 0 \vee n = c \\ 1 & \text{if } f(m,n) = 1 \wedge f(n,c) = 1 \wedge n \neq c \end{cases}$$

It can be verified that g and $\preceq_2$ are related as required by definition 2.5.1. Therefore B is a recursive ordinal. □

Using example 2.5.2 and lemma 2.5.3 we can find many infinite recursive ordinals. However, a diagonalisation argument can be used to show that not all countable ordinals are recursive, and so there must be a least non-recursive ordinal.

Definition 2.5.4 The ***least non-recursive ordinal*** is denoted ω_1^{CK}.

Church and Kleene [CK37, Chu38, Kle38] identified the ordinal ω_1^{CK}. In the literature both ω_1^{CK} and ω_1 are used for the least non-recursive ordinal. We use ω_1 for the least non-countable ordinal. The subscript 1 is present because it is possible to define a sequence of countable ordinals by defining $\omega_{\mathrm{n+1}}^{\mathrm{CK}}$ in terms of $\omega_{\mathrm{n}}^{\mathrm{CK}}$. From this perspective ω_0^{CK} is ω.

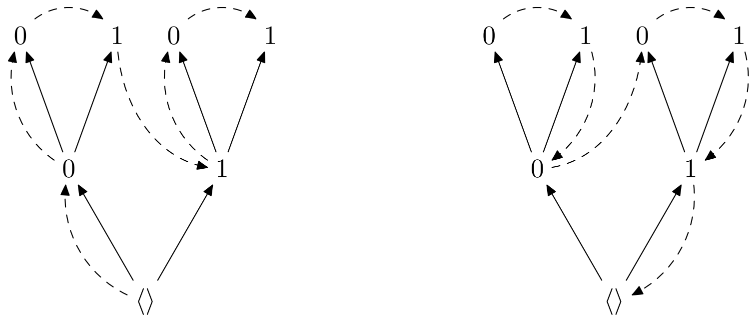
There are many different characterisations of ω_1^{CK}. We now work towards a characterisation in terms of trees consisting of finite sequences of natural numbers (see example 2.1.12) with the prefix order. The *Kleene-Brouwer order* on the elements of such a tree is used to construct an ordinal from that tree (see [Gir87, Odi89, Mos90]). The definition of the Kleene-Brouwer order is similar to that of the well-known *lexicographic order*, so both are defined in definition 2.5.5 for the sake of comparison.

Definition 2.5.5 Consider a tree $\langle A, \sqsubseteq \rangle$, where $A \subseteq \bigcup\{\omega^n \mid n \in \omega\}$, A is prefix closed, and $\sqsubseteq$ is the prefix order. The ***lexicographic order*** $\leq_{\mathrm{LX}}$ and the ***Kleene-Brouwer order*** $\leq_{\mathrm{KB}}$ (also known as the ***Lusin-Sierpinski order***) are defined, for $\vec{a} = \langle a_0, a_1, \ldots, a_m \rangle, \vec{b} = \langle b_0, b_1, \ldots, b_n \rangle \in A$, by:

$$\vec{a} <_{\mathrm{LX}} \vec{b} \stackrel{\text{def}}{=} \vec{a} \sqsubset \vec{b} \vee (\exists i \in \omega. i \leq m \sqcap n \wedge a_i < b_i \wedge \forall j < i. a_j = b_j)$$
$$\vec{a} <_{\mathrm{KB}} \vec{b} \stackrel{\text{def}}{=} \vec{b} \sqsubset \vec{a} \vee (\exists i \in \omega. i \leq m \sqcap n \wedge a_i < b_i \wedge \forall j < i. a_j = b_j)$$

The lexicographic and Kleene-Brouwer orders differ in whether or not a prefix $\vec{a}$ of a sequence $\vec{b}$ is less than or greater than $\vec{b}$. For example, the empty sequence $\langle\rangle$ is the bottom element for the lexicographic order, and the top element for the Kleene-Brouwer order. Figure 2.6 illustrates the successor relation for the orders on finite trees.

The lexicographic and Kleene-Brouwer orders are total, and are well-orders whenever the tree is well-founded. However, the reverse implication only holds for the Kleene-Brouwer order, because of their different behaviour with respect to $\sqsubset$.



(a) Lexicographic order (b) Kleene-Brouwer order

Figure 2.6: Orders on recursive trees

Lemma 2.5.6 Consider a tree $\langle A, \sqsubseteq \rangle$, where $A \subseteq \bigcup\{\omega^n \mid n \in \omega\}$, A is prefix closed, and $\sqsubseteq$ is the prefix order. The following properties hold of the lexicographic and Kleene-Brouwer orders:

1. $\langle A, \leq_{\mathrm{LX}} \rangle$ and $\langle A, \leq_{\mathrm{KB}} \rangle$ are total orders.

2. If $\langle A, \sqsubseteq \rangle$ is a well-founded tree, then $<_{\mathrm{LX}}$ and $<_{\mathrm{KB}}$ are well-founded on A.

3. If $<_{\mathrm{KB}}$ is well-founded on A, then $\langle A, \sqsubseteq \rangle$ is a well-founded tree.

Proof

1. The relations are reflexive by definition, and it is straightforward to show that they are transitive. If $\vec{a} = \langle a_0, a_1, \ldots, a_m \rangle, \vec{b} = \langle b_0, b_1, \ldots, b_n \rangle \in A$ such that $\vec{a} \not\sqsubseteq \vec{b}$ and $\vec{b} \not\sqsubseteq \vec{a}$, then there exists a unique $i \in \omega$ such that $i \leq m \sqcap n$, $a_i \neq b_i$, and, for all $j \in \omega$, $j < i$ implies $a_j = b_j$. It follows that $\vec{a}$ and $\vec{b}$ are comparable by $\leq_{\mathrm{LX}}$ and $\leq_{\mathrm{KB}}$, and so both orders are total.

2. Consider $\leq_{\mathrm{LX}}$ first. For a contradiction, suppose that $\langle A, \sqsubseteq \rangle$ is a well-founded tree and that there exists an ω-chain $\langle \vec{a}_i \mid i \in \omega \rangle$ such that $\vec{a}_{i+1} <_{\mathrm{LX}} \vec{a}_i$, for all $i \in \omega$. Set $\vec{a}_i = \langle a_{i,0}, a_{i,1}, \ldots, a_{i,m_i} \rangle$, for all $i \in \omega$. We define an ω-sequence of natural numbers $\langle b_i \mid i \in \omega \rangle$ such that $\langle b_0, b_1, \ldots, b_i \rangle \in A$, for all $i \in \omega$. If such a sequence exists, then $\langle \langle b_0, b_1, \ldots, b_i \rangle \mid i \in \omega \rangle$ is a strictly increasing ω-chain in A with respect to $\sqsubseteq$, and thus $\langle A, \sqsubseteq \rangle$ is not a well-founded tree. How can we define $b_0 \in \omega$? For all $i \in \omega$, $\vec{a}_i \neq \langle\rangle$ because $\langle\rangle$ is the bottom element for $\leq_{\mathrm{LX}}$. Hence there is an ω-sequence $\langle a_{i,0} \mid i \in \omega \rangle$ of the first elements from each sequence. This sequence is decreasing because $\vec{a}_{i+1} <_{\mathrm{LX}} \vec{a}_i$, for all $i \in \omega$, and so must eventually be constant. Thus there exists $k \in \omega$ such that, for all $i \in \omega$, $a_{k,0} = a_{k+i,0}$. Set $b_0 = a_{k,0}$. We know that $\langle b_0 \rangle \in A$ because A is prefix-closed. To define the remainder of the sequence $\langle b_i \mid i \in \omega \rangle$, consider the sequence $\langle \langle a_{k+i,1}, a_{k+i,2}, \ldots, a_{k+i,m_i} \rangle \mid i \in \omega \rangle$ (the sequence obtained by removing the first element b_0 from each sequence in $\langle \vec{a}_{k+i} \mid i \in \omega \rangle$). It can be verified that this is a strictly decreasing sequence with respect to $\leq_{\mathrm{LX}}$, and so

we can iterate this process to obtain $\langle b_i \mid i \in \omega\rangle$ such that, for all $i \in \omega$, there exists $j \in \omega$ such that, for all $k \in \omega$, $k \geq j$ implies $\langle b_0, b_1, \ldots, b_i\rangle \sqsubseteq \vec{a}_k$. Therefore $\langle b_0, b_1, \ldots, b_i\rangle \in A$, for all $i \in \omega$, because A is prefix-closed. This contradicts the assumption that $\langle A, \sqsubseteq\rangle$ is a well-founded tree, and so $<_{\mathrm{LX}}$ is well-founded on A. A similar argument shows that $<_{\mathrm{KB}}$ is well-founded on A. The only difference is that $\langle\rangle$ is the top element for $\leq_{\mathrm{KB}}$ and so it may appear at the start of a strictly descending ω-chain, in which case it can be removed without affecting the rest of the proof.

3. Let $<_{\mathrm{KB}}$ be well-founded on A. For a contradiction, suppose that that there exists an ω-chain $\langle \vec{a}_i \in A \mid i \in \omega\rangle$ such that $\vec{a}_i \sqsubset \vec{a}_{i+1}$, for all $i \in \omega$. But $\vec{a}_i \sqsubset \vec{a}_{i+1}$ implies $\vec{a}_{i+1} <_{\mathrm{KB}} \vec{a}_i$, and so $\langle \vec{a}_i \in A \mid i \in \omega\rangle$ is a strictly decreasing ω-chain with respect to $<_{\mathrm{KB}}$. This contradicts the assumption that $<_{\mathrm{KB}}$ is well-founded, and therefore $\langle A, \sqsubseteq\rangle$ is a well-founded tree. $\square$

Example 2.5.7 shows that the lexicographic order can be well-founded on a non-well-founded tree.

Example 2.5.7 Consider the tree $\langle A, \sqsubseteq\rangle$, where $\sqsubseteq$ is the prefix order and $A \stackrel{\text{def}}{=} \{0\}^*$ is the set of all finite sequences of 0 (including the empty sequence). The strict lexicographic order $<_{\mathrm{LX}}$ is well-founded on $A \times A$, but the tree is not well-founded because of the ω-chain $\langle\rangle \sqsubset \langle 0\rangle \sqsubset \langle 0,0\rangle \sqsubset \ldots$

A well-founded tree is well-ordered by the Kleene-Brouwer order, so we can compare the ranks of elements of the tree with respect to $\sqsupset$ and $<_{\mathrm{KB}}$.

Lemma 2.5.8 Consider a tree $\langle A, \sqsubseteq\rangle$, where $A \subseteq \bigcup\{\omega^n \mid n \in \omega\}$, A is prefix closed, and $\sqsubseteq$ is the prefix order. If the tree $\langle A, \sqsubseteq\rangle$ is well-founded, then, for all $\vec{a} \in A$, $Rank(\vec{a}, \sqsupset) \leq Rank(\vec{a}, <_{\mathrm{KB}})$ (when the relations are restricted to $A \times A$). In particular, the rank of the tree, given by $Rank(\langle\rangle, \sqsupset)$, is less than or equal to $Rank(\langle\rangle, <_{\mathrm{KB}})$.

Proof By well-founded induction with respect to $<_{\mathrm{KB}}$ on $A \times A$. Consider $\vec{a} \in A$. If we also have $\vec{b} \in A$ such that $\vec{a} \sqsubset \vec{b}$, then $\vec{b} <_{\mathrm{KB}} \vec{a}$. By the induction hypothesis, $Rank(\vec{b}, \sqsupset) \leq Rank(\vec{b}, <_{\mathrm{KB}})$. Then we deduce:

$$\begin{aligned}
Rank(\vec{a}, \sqsupset) &= \bigcup\{Succ(Rank(\vec{b}, \sqsupset)) \mid \vec{b} \in A \wedge \vec{b} \sqsupset \vec{a}\} \\
&\leq \bigcup\{Succ(Rank(\vec{b}, \sqsupset)) \mid \vec{b} \in A \wedge \vec{b} <_{\mathrm{KB}} \vec{a}\} \\
&\leq \bigcup\{Succ(Rank(\vec{b}, <_{\mathrm{KB}})) \mid \vec{b} \in A \wedge \vec{b} <_{\mathrm{KB}} \vec{a}\} \\
&= Rank(\vec{a}, <_{\mathrm{KB}})
\end{aligned}$$

$\square$

Therefore, a well-founded tree determines a well-order that, as an ordinal, is greater than or equal to the rank of the tree.

The alternative characterisation of ω_1^{CK} is in terms of *recursive trees*.

Definition 2.5.9 A tree $\langle A, \sqsubseteq \rangle$ is ***recursive*** if $A \subseteq \bigcup\{\omega^n \mid n \in \omega\}$, A is prefix closed, $\sqsubseteq$ is the prefix order, and the characteristic function of A is recursive (with respect to a suitable encoding of $\bigcup\{\omega^n \mid n \in \omega\}$ in ω).

Proposition 2.5.10 gives an alternative characterisation of ω_1^{CK} as the least ordinal that is greater than the rank of every recursive well-founded tree. The Kleene-Brouwer order is used in the proof to construct a recursive ordinal from a recursive well-founded tree, and, by lemma 2.5.8, that ordinal is greater than or equal to the rank of the tree.

Proposition 2.5.10 An ordinal A is recursive if and only if there exists a recursive well-founded tree $\langle B, \sqsubseteq \rangle$ such that the rank of the tree is A, i.e., $Rank(\langle\rangle, \sqsupset) = A$ (when $\sqsupset$ is restricted to $B \times B$).

Proof Suppose that A is a recursive ordinal, so there exists $B \subseteq \omega$, a well-order $\preceq \subseteq B \times B$, and a recursive function $f : \omega \times \omega \to \omega$ as in definition 2.5.1. Recall that the $\in$-tree $Tree(A)$ has rank A by lemma 2.1.15. The underlying set of $Tree(A)$ is

$$\{\langle\rangle\} \cup \{\langle a_0, a_1, \ldots, a_n\rangle \mid n \in \omega \wedge a_0 \in A \wedge \forall i < n.\, a_{i+1} \in a_i\}$$

The order-isomorphism between $\langle A, \subseteq \rangle$ and $\langle B, \preceq \rangle$ induces an order-isomorphism between $Tree(A)$ and the tree $\langle C, \sqsubseteq \rangle$, where $C \subseteq \bigcup\{\omega^n \mid n \in \omega\}$ is defined by:

$$C \stackrel{\text{def}}{=} \{\langle\rangle\} \cup \{\langle c_0, c_1, \ldots, c_n\rangle \mid n \in \omega \wedge f(c_0, c_0) = 1 \wedge \forall i < n.\, f(c_{i+1}, c_i) = 1 \wedge c_{i+1} \neq c_i\}$$

The rank of a tree is invariant under order-isomorphism, so the rank of the tree $\langle C, \sqsubseteq \rangle$ is also A. By the fact that f is recursive, $\langle C, \sqsubseteq \rangle$ is a recursive tree, and we are done.

For the other direction, suppose that A is an ordinal, $\langle B, \sqsubseteq \rangle$ is a recursive well-founded tree, and the rank of $\langle B, \sqsubseteq \rangle$ is A, i.e., $Rank(\langle\rangle, \sqsupset) = A$ (when restricted to $B \times B$). We need to show that A is a recursive ordinal. Using lemmas 2.5.6 and 2.5.8, the Kleene-Brouwer order $\leq_{\mathrm{KB}}$ is a well-order on A, and $Rank(\langle\rangle, <_{\mathrm{KB}}) \geq Rank(\langle\rangle, \sqsupset) = A$ (when restricted to $B \times B$). By lemma 2.5.3, it suffices to show that $Rank(\langle\rangle, <_{\mathrm{KB}})$ is a recursive ordinal. A finite sequence of natural numbers can be encoded as a natural number, and, by the fact that $\langle B, \sqsubseteq \rangle$ is a recursive tree, the Kleene-Brouwer order $<_{\mathrm{KB}}$ is decidable on the encodings. Of course, we can decide whether one sequence is less than another with respect to the Kleene-Brouwer order even if the tree is not recursive. However, we also need to be able to decide membership of B in order to construct a recursive ordinal. Therefore we have a recursive ordinal greater than or equal to A, and so A is also a recursive ordinal. $\square$

2.6 Binary Choice Operators

In this section, we consider some of the binary choice operators considered in the literature. The discussion is based upon a simple set-based model of non-deterministic computation, as opposed to an operational semantics for the choice operators. We assume that non-deterministic programs can be modelled within $\mathcal{P}_{\text{ne}}(\omega_\perp)$ (the set of non-empty subsets of $\omega_\perp$), and a non-deterministic program is represented by $A \in \mathcal{P}_{\text{ne}}(\omega_\perp)$ if, for all $n \in \omega$, the program may terminate with result n if and only if $n \in A$, and the program may fail to terminate if and only if $\perp \in A$. Binary choice operators are modelled by functions $\mathcal{P}_{\text{ne}}(\omega_\perp) \times \mathcal{P}_{\text{ne}}(\omega_\perp) \to \mathcal{P}_{\text{ne}}(\omega_\perp)$. After defining these functions, we show that there is a natural LTSWD for $\mathcal{P}_{\text{ne}}(\omega_\perp)$, and so we have definitions of the variants of similarity and bisimilarity upon $\mathcal{P}_{\text{ne}}(\omega_\perp)$. We then describe representations of the equivalence classes with respect to lower mutual similarity and upper mutual similarity, and identify which choice operators are well-defined on the equivalence classes.

The binary choice operators that we consider are:

1. *Global angelic choice*: returns a value that one of the arguments can return, but only fails to terminate if both arguments always fail to terminate.
2. *Ambiguous choice*: evaluates both arguments and returns the value returned by the first argument to terminate (the relative speed of evaluation is deliberately unspecified).
3. *Erratic choice*: chooses between the arguments before evaluating just one of them, and returning the value if it terminates.
4. *Local demonic choice*: evaluates both arguments and, if both terminate with a value, it returns one of those values, otherwise it fails to terminate.
5. *Global demonic choice*: always fail to terminate if either argument may fail to terminate, but otherwise returns a value that one of the arguments can return.

The descriptions above are rather imprecise, so we define a function for each operator.

Definition 2.6.1 The binary choice operators are represented by functions:

$$GAng, Amb, Err, LDem, GDem : \mathcal{P}_{\text{ne}}(\omega_\perp) \times \mathcal{P}_{\text{ne}}(\omega_\perp) \to \mathcal{P}_{\text{ne}}(\omega_\perp)$$

which are defined, for $A, B \in \mathcal{P}_{\text{ne}}(\omega_\perp)$ by:

$$\begin{aligned}
GAng(A,B) &\stackrel{\text{def}}{=} \{\perp \mid A \cup B = \{\perp\}\} \cup \{m \in \omega \mid m \in A\} \cup \{n \in \omega \mid n \in B\} \\
Amb(A,B) &\stackrel{\text{def}}{=} \{\perp \mid \perp \in A \cap B\} \cup \{m \in \omega \mid m \in A\} \cup \{n \in \omega \mid n \in A\} \\
Err(A,B) &\stackrel{\text{def}}{=} \{\perp \mid \perp \in A \cup B\} \cup \{m \in \omega \mid m \in A\} \cup \{n \in \omega \mid n \in A\} \\
LDem(A,B) &\stackrel{\text{def}}{=} \{\perp \mid \perp \in A \cup B\} \cup \{m \in \omega \mid m \in A \wedge (\exists n \in \omega . n \in B)\} \\
&\qquad \cup \{n \in \omega \mid (\exists m \in \omega . m \in A) \wedge n \in B\} \\
GDem(A,B) &\stackrel{\text{def}}{=} \{\perp \mid \perp \in A \cup B\} \cup \{m \in \omega \mid m \in A \wedge \perp \notin A \cup B\} \\
&\qquad \cup \{n \in \omega \mid n \in B \wedge \perp \notin A \cup B\}
\end{aligned}$$

	$GAng$	Amb	Err	$LDem$	$GDem$
$\{0\}$ and $\{1\}$	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$
$\{0\}$ and $\{\perp\}$	$\{0\}$	$\{0\}$	$\{\perp,0\}$	$\{\perp\}$	$\{\perp\}$
$\{0\}$ and $\{\perp,1\}$	$\{0,1\}$	$\{0,1\}$	$\{\perp,0,1\}$	$\{\perp,0,1\}$	$\{\perp\}$
$\{\perp\}$ and $\{\perp\}$	$\{\perp\}$	$\{\perp\}$	$\{\perp\}$	$\{\perp\}$	$\{\perp\}$
$\{\perp\}$ and $\{\perp,1\}$	$\{1\}$	$\{\perp,1\}$	$\{\perp,1\}$	$\{\perp\}$	$\{\perp\}$
$\{\perp,0\}$ and $\{\perp,1\}$	$\{0,1\}$	$\{\perp,0,1\}$	$\{\perp,0,1\}$	$\{\perp,0,1\}$	$\{\perp\}$

Figure 2.7: Action of binary choice operators

We have the following equalities, for all $A,B \in \mathcal{P}_{\mathrm{ne}}(\omega_\perp)$:

$$\begin{aligned}
GAng(A,B) &= \begin{cases} \{\perp\} & \text{if } A \cup B = \{\perp\} \\ (A \cup B) \setminus \{\perp\} & \text{otherwise} \end{cases} \\
Amb(A,B) &= \begin{cases} A \cup B & \text{if } \perp \in A \cap B \\ (A \cup B) \setminus \{\perp\} & \text{otherwise} \end{cases} \\
Err(A,B) &= A \cup B \\
GDem(A,B) &= \begin{cases} \{\perp\} & \text{if } \perp \in A \cup B \\ A \cup B & \text{otherwise} \end{cases}
\end{aligned}$$

Figure 2.7 shows the behaviour of the choice operators on a representative collection of sets.

The set $\mathcal{P}_{\mathrm{ne}}(\omega_\perp)$ can be considered as the set of states of an LTSWD. As before, a $\perp$ urelement signifies divergence, but now the natural numbers are treated as urelements, not as sets. Definition 2.6.2 describes an LTSWD for the special case $\mathcal{P}_{\mathrm{ne}}(\omega_\perp)$, but a more general definition could be given for arbitrary sets of urelements.

Definition 2.6.2 The $\in$-LTSWD for $\mathcal{P}_{\mathrm{ne}}(\omega_\perp)$ is $\langle \mathcal{P}(\omega_\perp), \omega, \Uparrow^{\mathrm{may}}, \rightarrow \rangle$, where the may divergence predicate and labelled transition relation are defined, for $A \subseteq \omega_\perp$, by:

- $A \Uparrow^{\mathrm{may}}$ if and only if $\perp \in A$.
- For all $n \in \omega$, $A \xrightarrow{n} \emptyset$ if and only if $n \in A$.

Note that it is necessary to add a terminal state in the LTSWD for $\mathcal{P}_{\mathrm{ne}}(\omega_\perp)$, and the empty set serves this purpose.

The variants of similarity, mutual similarity, and bisimilarity apply to the states of the $\in$-LTSWD for $\mathcal{P}_{\mathrm{ne}}(\omega_\perp)$, but it is useful to have an elementary definition. The following equivalences hold for the variants of similarity, for all $A,B \in \mathcal{P}_{\text{ne}}(\omega_\perp)$:

$$\begin{aligned}
A \lesssim_{\text{LS}} B &\iff \forall n \in \omega. n \in A \implies n \in B \\
A \lesssim_{\text{US}} B &\iff \perp \notin A \implies (\perp \notin B \wedge \forall n \in \omega. n \in B \implies n \in A) \\
A \lesssim_{\text{CS}} B &\iff (\forall n \in \omega. n \in A \implies n \in B) \wedge \\
&\qquad \perp \notin A \implies (\perp \notin B \wedge \forall n \in \omega. n \in B \implies n \in A) \\
A \lesssim_{\text{RS}} B &\iff (\forall n \in \omega. n \in B \implies n \in A) \wedge \\
&\qquad \perp \notin A \implies (\perp \notin B \wedge \forall n \in \omega. n \in B \implies n \in A)
\end{aligned}$$

For lower similarity and refinement similarity, we can simplify to:

$$\begin{aligned}
A \lesssim_{\text{LS}} B &\iff A \setminus \{\perp\} \subseteq B \setminus \{\perp\} \\
A \lesssim_{\text{RS}} B &\iff B \subseteq A
\end{aligned}$$

For upper similarity and the second part of convex similarity, we have that, for all $A,B \in \mathcal{P}_{\text{ne}}(\omega_\perp)$:

$$\begin{aligned}
&\perp \notin A \implies (\perp \notin B \wedge \forall n \in \omega. n \in B \implies n \in A) \\
\iff &\perp \notin A \implies (\perp \notin B \wedge B \subseteq A) \\
\iff &(\perp \in B \implies \perp \in A) \wedge (\perp \in A \vee B \subseteq A) \\
\iff &(\perp \in B \implies \perp \in A) \wedge (\forall n \in \omega. n \in B \implies \perp \in A \vee n \in A)
\end{aligned}$$

In the $\in$-LTSWD for $\mathcal{P}_{\text{ne}}(\omega_\perp)$, each variant of mutual similarity coincides with the corresponding variant of bisimilarity, because there no sequences of transitions with length greater than one (a similar result is proved in lemma 4.2.5). In addition, convex bisimilarity, which is the same as refinement bisimilarity, coincides with equality. It follows that convex similarity and refinement similarity are anti-symmetric, and thus partial orders.

Lower similarity and upper similarity are not anti-symmetric, and so lower bisimilarity and upper bisimilarity have non-trivial equivalence classes. All of the non-trivial equivalences are generated by:

- For $A \in \mathcal{P}_{\text{ne}}(\omega)$, $A \simeq_{\text{LB}} A \cup \{\perp\}$.
- For $A,B \in \mathcal{P}(\omega)$, $A \cup \{\perp\} \simeq_{\text{UB}} B \cup \{\perp\}$.

For lower bisimilarity and upper bisimilarity, the above equivalences imply that there is a unique member of each equivalence class in $\mathcal{P}_{\text{ne}}(\omega) \cup \{\perp\}$, and so lower similarity and upper similarity are partial orders on this set. The bijection between $\mathcal{P}_{\text{ne}}(\omega) \cup \{\perp\}$ and $\mathcal{P}(\omega)$ that maps $\perp$ to the empty set can be turned into order-isomorphisms for lower similarity and upper similarity by defining partial orders $\leq_{\text{LS}} \subseteq \mathcal{P}(\omega) \times \mathcal{P}(\omega)$ and $\leq_{\text{US}} \subseteq \mathcal{P}(\omega) \times \mathcal{P}(\omega)$, for $A,B \in \mathcal{P}(\omega)$, by:

$$\begin{aligned}
A \leq_{\text{LS}} B &\iff A \subseteq B \\
A \leq_{\text{US}} B &\iff A = \emptyset \vee (B \neq \emptyset \wedge B \subseteq A)
\end{aligned}$$

	GAng	*Amb*	*Err*	*LDem*	*GDem*
Lower Similarity	✓	✓	✓	✓	✗
Upper Similarity	✗	✗	✓	✓	✓
Convex Similarity	✗	✗	✓	✓	✓
Refinement Similarity	✗	✓	✓	✓	✗

Figure 2.8: Monotonicity of choice operators

Then $\langle \mathcal{P}_{\mathrm{ne}}(\omega) \cup \{\bot\}, \lesssim_{\mathrm{LS}}\rangle$ is order-isomorphic to $\langle \mathcal{P}(\omega), \leq_{\mathrm{LS}}\rangle$, and $\langle \mathcal{P}_{\mathrm{ne}}(\omega) \cup \{\bot\}, \lesssim_{\mathrm{US}}\rangle$ is order-isomorphic to $\langle \mathcal{P}(\omega), \leq_{\mathrm{US}}\rangle$. The partial orders on $\mathcal{P}(\omega)$ are used more often than those on $\mathcal{P}_{\mathrm{ne}}(\omega) \cup \{\bot\}$ in the specification and refinement literature. In particular, the partial order $\langle \mathcal{P}(\omega), \leq_{\mathrm{US}}\rangle$ is often embedded into the space of predicate transformers which, in this case, is a subset of $\mathcal{P}(\omega) \to \mathcal{P}(\omega)$.

If the choice functions are monotone for lower similarity and upper similarity, then they determine well-defined monotone choice functions on the equivalence classes of $\mathcal{P}(\omega)$. The choice functions and their monotonicity properties for the different variants of similarity are presented in figure 2.8. Counter examples for the non-monotonic combinations are:

$$
\begin{aligned}
&\mathit{GDem}(\{0\},\{1\}) = \{0,1\} \not\lesssim_{\mathrm{LS}} \{\bot\} = \mathit{GDem}(\{\bot,0\},\{1\})\\
&\mathit{GAng}(\{\bot\},\{1\}) = \{1\} \not\lesssim_{\mathrm{US}} \{0,1\} = \mathit{GAng}(\{\bot,0\},\{1\})\\
&\mathit{Amb}(\{\bot\},\{1\}) = \{1\} \not\lesssim_{\mathrm{US}} \{0,1\} = \mathit{Amb}(\{\bot,0\},\{1\})\\
&\mathit{GAng}(\{\bot\},\{1\}) = \{1\} \not\lesssim_{\mathrm{CS}} \{0,1\} = \mathit{GAng}(\{\bot,0\},\{1\})\\
&\mathit{Amb}(\{\bot\},\{1\}) = \{1\} \not\lesssim_{\mathrm{CS}} \{0,1\} = \mathit{Amb}(\{\bot,0\},\{1\})\\
&\mathit{GAng}(\{\bot,0\},\{\bot\}) = \{0\} \not\lesssim_{\mathrm{RS}} \{\bot\} = \mathit{GAng}(\{\bot\},\{\bot\})\\
&\mathit{GDem}(\{\bot,0\},\{1\}) = \{\bot\} \not\lesssim_{\mathrm{RS}} \{0,1\} = \mathit{GDem}(\{0\},\{1\})
\end{aligned}
$$

Global angelic and global demonic choice are unusual because they require knowledge of all possible terminating and non-terminating behaviour of their arguments. Adding a terminating or non-terminating behaviour to one of the arguments of global angelic or global demonic choice may remove a terminating or non-terminating behaviour from the result, as demonstrated by the lack of monotonicity of global angelic and global demonic choice with respect to refinement similarity.

Informal operational accounts of global angelic and global demonic choice are sometimes described, and a formal semantics can be given (see [CC92]). However, these choice operators are normally studied in conjunction with lower similarity and upper similarity (respectively). Then there is no need to give an operational account of the global choice operators because they coincide with the erratic choice operator which has a straightforward operational semantics. It can be shown that, for all $A,B \in \mathcal{P}_{\mathrm{ne}}(\omega_\perp)$:

$$\begin{aligned}
&GAng(A,B) \simeq_{\mathrm{LB}} Amb(A,B) \simeq_{\mathrm{LB}} Err(A,B)\\
&Err(A,B) \simeq_{\mathrm{UB}} LDem(A,B) \simeq_{\mathrm{UB}} GDem(A,B)
\end{aligned}$$

The monotonicity of erratic choice with respect to lower similarity and upper similarity ensures that it induces well-defined monotone functions on the equivalence classes for those relations. The image of those functions on the partial orders $\langle \mathcal{P}(\omega), \leq_{\mathrm{LS}} \rangle$ and $\langle \mathcal{P}(\omega), \leq_{\mathrm{US}} \rangle$ are given by $LErr, UErr : \mathcal{P}(\omega) \times \mathcal{P}(\omega) \to \mathcal{P}(\omega)$, which are defined, for $A,B \in \mathcal{P}(\omega)$, by:

$$\begin{aligned}
&LErr(A,B) = A \cup B\\
&UErr(A,B) = \begin{cases} \emptyset & \text{if } (A = \emptyset) \vee (B = \emptyset)\\ A \cup B & \text{otherwise} \end{cases}
\end{aligned}$$

Chapter 3

The Non-Deterministic λ-Calculus $\mathscr{L}$

We define a non-deterministic λ-calculus $\mathscr{L}$ with a call-by-name operational semantics. The type system and syntax are based upon Moggi's computational λ-calculus [Mog89b, Mog89a, Mog91] in order to control the resolution of non-determinism. Non-determinism is introduced via indexed erratic choice terms. The non-deterministic terms that can be formed are not equally expressive, and in chapter 5 we study relative definability properties of such terms. A family of non-deterministic λ-calculi is obtained by considering fragments of $\mathscr{L}$. In the sequel, we show that the convex bisimilarity relations for the λ-calculi in this family are not simply restrictions of convex bisimilarity for $\mathscr{L}$.

Sections 3.1 and 3.2 introduce the type system and syntax of $\mathscr{L}$, and section 3.3 presents a type assignment system. Sections 3.4 and 3.5 define operational semantics via a reduction relation and an evaluation relation with a may divergence predicate. Section 3.6 proves that the terms of certain types always terminate. Section 3.7 states the closure conditions needed to obtain reasonable fragments of $\mathscr{L}$, and section 3.8 proves results about the ranks of derivation trees for the operational semantics in certain fragments of $\mathscr{L}$.

3.1 Types

This section defines a type system for the non-deterministic λ-calculus $\mathscr{L}$ introduced in section 3.2. It is a Church-style type assignment system (see [Bar92]), where terms are annotated with types in such a way that the type of a term can be inferred.

The set of types includes indexed coproducts and products, where the indexing set may be countably infinite. The natural numbers type can be defined as a coproduct type, as opposed to an inductive or recursive type. However, it is necessary to consider infinite terms in order to make full use of such types. For example, a case statement upon a term of natural numbers type must have a branch for each natural number. The map from the natural numbers to (codings of) the branches need not be computable, and this provides a richer structure in which to study relative definability.

The programming language is based upon Moggi's computational λ-calculus and thus has a computation type constructor $\mathsf{P}_\perp(\cdot)$. Programs that are non-terminating or non-deterministic must have a type of the form $\mathsf{P}_\perp(\sigma)$, for some type σ. For example, there is a program with type $\mathsf{P}_\perp(\mathsf{nat})$ that sometimes fails to terminate, sometimes terminates with result 0, and sometimes terminates with result 1.

Definition 3.1.1 The set of ***types*** is defined by:

$$\begin{array}{rcll} \sigma,\tau,\rho & ::= & \mathsf{sum}\,\langle \sigma_n \mid n<\kappa\rangle & \text{(indexed coproducts, } \kappa\le\omega) \\ & | & \mathsf{prod}\,\langle \sigma_n \mid n<\kappa\rangle & \text{(indexed products, } \kappa\le\omega) \\ & | & \sigma\to\tau & \text{(functions)} \\ & | & \mathsf{P}_\perp(\sigma) & \text{(non-terminating, non-deterministic computations)} \end{array}$$

The ***computation types*** are those of the form $\mathsf{P}_\perp(\sigma)$. The remaining types are called ***value types***.

The variable κ ranges over cardinals less than or equal to ω, and hence is always a natural number or ω. This restricts n to natural numbers. The set of types is well-defined even though the indexing set for coproducts and products may be countably infinite.

The following type abbreviations are used in the sequel:

$$\begin{aligned} \mathsf{unit} &\stackrel{\text{def}}{=} \mathsf{prod}\,\langle\rangle \\ \mathsf{bool} &\stackrel{\text{def}}{=} \mathsf{sum}\,\langle \mathsf{unit},\mathsf{unit}\rangle \\ \mathsf{nat} &\stackrel{\text{def}}{=} \mathsf{sum}\,\langle \mathsf{unit} \mid n<\omega\rangle \\ \sigma\times\tau &\stackrel{\text{def}}{=} \mathsf{prod}\,\langle\sigma,\tau\rangle \\ \sigma+\tau &\stackrel{\text{def}}{=} \mathsf{sum}\,\langle\sigma,\tau\rangle \end{aligned}$$

In section 4.2 we require a measure of the occurrences of the computation type constructor $\mathsf{P}_\perp(\cdot)$ in a type. A suitable measure is formalised in definition 3.1.2.

Definition 3.1.2 The ***P-order*** of a type σ is an ordinal defined by induction on σ:

$$\begin{aligned} POrd(\mathsf{sum}\,\langle\sigma_n \mid n<\kappa\rangle) &\stackrel{\text{def}}{=} \bigcup\{POrd(\sigma_n) \mid n<\kappa\} \\ POrd(\mathsf{prod}\,\langle\sigma_n \mid n<\kappa\rangle) &\stackrel{\text{def}}{=} \bigcup\{POrd(\sigma_n) \mid n<\kappa\} \\ POrd(\sigma\to\tau) &\stackrel{\text{def}}{=} POrd(\tau) \\ POrd(\mathsf{P}_\perp(\sigma)) &\stackrel{\text{def}}{=} POrd(\sigma)+1 \end{aligned}$$

The P-order of a type is the rank of a tree derived from the type by discarding the source type from function types, and folding coproduct, product, and function types into the nearest enclosing computation type. It can be shown that the image of the P-order function is ω_1.

3.2 Language

We define the syntax of the non-deterministic λ-calculus $\mathscr{L}$ and discuss features of the language.

$$
\begin{array}{rcll}
L,M,N & ::= & x & \text{(variable, } x \in \mathit{Var}\text{)} \\
& | & \mathsf{inj}\, n, \sigma\, \mathsf{of}\, M & \text{(injection into component } n,\ n < \omega\text{)} \\
& | & \mathsf{case}\, M\, \mathsf{of}\, \langle x_n.N_n \mid n < \kappa \rangle & \text{(case, } x_n \text{ bound in } N_n,\ \kappa \leq \omega\text{)} \\
& | & \mathsf{tuple}\, \langle M_n \mid n < \kappa \rangle & \text{(tuple, } \kappa \leq \omega\text{)} \\
& | & \mathsf{proj}\, n\, \mathsf{of}\, M & \text{(projection of component } n,\ n < \omega\text{)} \\
& | & \lambda x{:}\sigma.\, M & \text{(abstraction, } x \text{ bound in } M\text{)} \\
& | & M\, N & \text{(application)} \\
& | & [M] & \text{(unit)} \\
& | & \mathsf{let}\, x{:}\sigma \Leftarrow M\, \mathsf{in}\, N & \text{(sequencing, } x \text{ bound in } N\text{)} \\
& | & \mathsf{fix}\, x{:}\sigma.\, M & \text{(fixed-point, } x \text{ bound in } M\text{)} \\
& | & ?\langle M_n \mid n < \kappa \rangle & \text{(indexed erratic choice, } 0 < \kappa \leq \omega\text{)} \\
\\
K & ::= & \mathsf{inj}\, n, \sigma\, \mathsf{of}\, M & \\
& | & \mathsf{tuple}\, \langle M_n \mid n < \kappa \rangle & \\
& | & \lambda x{:}\sigma.\, M & \\
& | & [M] &
\end{array}
$$

Figure 3.1: Terms and canonical terms

Definition 3.2.1 A set of variables Var with cardinality ω_1 is assumed. The ***terms*** and ***canonical terms*** of the language $\mathscr{L}$ are defined in figure 3.1. The scope of variable binding constructs extends as far to the right as possible. The ***free variables*** of a term are given by a function $Fv(\cdot)$ defined in figure 3.2 by induction on terms. A term M is ***closed*** if $Fv(M) = \emptyset$, and ***open*** otherwise.

Set-theoretic tuples of terms are written $\langle M_n \mid n < \kappa \rangle$, whereas tuples of the language have the form $\mathsf{tuple}\,\langle M_n \mid n < \kappa \rangle$. A term of $\mathscr{L}$ may be an infinite object and should be thought of as a countably-branching well-founded tree, where each instance of a term constructor corresponds to a node in the tree, rather than as a sequence of symbols. Formally, the set of terms can be constructed as the least fixed-point of a monotone function (determined by the grammar in figure 3.1) on a set of trees composed of sequences (see example 2.1.12).

Infinitary languages have been used for mathematical and program logics with infinitary conjunctions, disjunctions, or quantifiers (see [Kei77, Mil89, Abr87a]). Several process calculi also permit infinitary term constructors: CCS has $\sum_{i\in I} P_i$, and CSP with unbounded non-determinism [Ros88, Ros98] has $\bigsqcap_{i\in I} P_i$. In contrast with these languages, $\mathscr{L}$ allows only natural numbers or ω as indexing sets. This does not reduce the expressiveness of $\mathscr{L}$ and makes it easier to work with the definition of compatibility in chapter 5. However, it does imply that, in general, the terms $?\langle M,N\rangle$, $?\langle N,M\rangle$, and $?\langle M,M,N\rangle$ are not syntactically identical, although they are identified by all of the semantic relations we consider because binary erratic choice is expected to be commutative, associative, and idempotent (cf. the non-idempotent category-theoretic semantics for erratic non-determinism in [Leh76, Abr83, PR88, Rus90]). Note that $?\langle\rangle$ is not a term because κ must be non-zero for indexed erratic choice.

As with types, a set-theoretic tuple of terms $\langle M_n \mid n < \kappa \rangle$ that appears in another term need not arise as a computable map with respect to some coding. For the indexed erratic choice

$$
\begin{aligned}
Fv(x) &\stackrel{\text{def}}{=} \{x\} \\
Fv(\mathsf{inj}\, n, \sigma\, \mathsf{of}\, M) &\stackrel{\text{def}}{=} Fv(M) \\
Fv(\mathsf{case}\, M\, \mathsf{of}\, \langle x_n.N_n \mid n < \kappa\rangle) &\stackrel{\text{def}}{=} Fv(M) \cup \bigcup\{Fv(N_n) \setminus \{x_n\} \mid n < \kappa\} \\
Fv(\mathsf{tuple}\, \langle M_n \mid n < \kappa\rangle) &\stackrel{\text{def}}{=} \bigcup\{Fv(M_n) \mid n < \kappa\} \\
Fv(\mathsf{proj}\, n\, \mathsf{of}\, M) &\stackrel{\text{def}}{=} Fv(M) \\
Fv(\lambda x{:}\sigma.\, M) &\stackrel{\text{def}}{=} Fv(M) \setminus \{x\} \\
Fv(M\, N) &\stackrel{\text{def}}{=} Fv(M) \cup Fv(N) \\
Fv([M]) &\stackrel{\text{def}}{=} Fv(M) \\
Fv(\mathsf{let}\, x{:}\sigma \Leftarrow M\, \mathsf{in}\, N) &\stackrel{\text{def}}{=} Fv(M) \cup (Fv(N) \setminus \{x\}) \\
Fv(\mathsf{fix}\, x{:}\sigma.\, M) &\stackrel{\text{def}}{=} Fv(M) \setminus \{x\} \\
Fv(?\langle M_n \mid n < \kappa\rangle) &\stackrel{\text{def}}{=} \bigcup\{Fv(M_n) \mid n < \kappa\}
\end{aligned}
$$

Figure 3.2: Free variables

constructor this is reasonable because we intend to reason about families of implementations, and there is no suggestion that a single implementation should be able to generate all possible outcomes (or test whether an outcome is possible). For indexed coproducts and products, we expect to write specifications using the full language and subsequently refine them to a fragment of the language that uses only a restricted collection of countably infinite case statements for arithmetic.

In the presence of countably-infinite term constructors it is necessary to assume an uncountable set of variables if the usual variable renaming and capture-free substitution conventions are to be used. For example, if $Var = \{x_n \mid n \in \omega\}$, then $\mathsf{tuple}\,\langle x_n \mid n < \omega\rangle$ would exhaust the supply of variables. In fact, such terms cannot be typed using the system defined in section 3.3 because environments are finite and there are no term constructors that bind infinitely many variables simultaneously in the same subterm. A Martin-Löf style "split" term constructor (see [Tho91, Gor94]) would have to bind infinitely many variables.

A set of variables with cardinality ω_1 is adequate because every term uses only countably many variables in free, bound, or binding occurrences. This can be shown by induction on the structure of terms, making use of the fact that ω_1 is a regular cardinal[1]. This property implies that fresh variables can always be chosen, even for a countable set of terms, and so we can establish the usual naming convention for bound variables: each variable appears at most once in a binding occurrence within a term. Every term is *α-equivalent* (see [Bar84]) to a term satisfying the naming convention, and henceforth terms are considered up to α-equivalence instead of syntactic identity. *Capture-free substitution* (or just *substitution*) can be defined in the usual way upon the equivalence classes with respect to α-equivalence (see [Cro93]).

Substitution into a canonical term clearly results in another canonical term. Conversely, if a canonical term is the result of a substitution it can be shown that one of the original terms must have been canonical.

[1] A cardinal κ is regular if it is not the limit of a set of ordinals strictly less than κ unless the set has cardinality greater than or equal to κ (see [Kun80]).

Lemma 3.2.2 If $M[N/x]$ is canonical, then either M is canonical, or $M = x$ and N is canonical.

Proof Case analysis of M. □

The unit and sequencing term constructors are taken from Moggi's computational λ-calculus [Mog89b, Mog91]. The computational λ-calculus arose from the observation that there is commonality between the abstract structures used to model programming languages with different *notions of computation*. The common structure can be formulated as a finite product category with a strong monad and additional structure that depends on the notion of computation embodied by the programming language. For example, in a category of predomains the lifting functor $(\cdot)_\perp$ forms part of a strong monad (see [Fio94]).

There is a well known correspondence between many-sorted equational logic and finite product categories (see [LS86, Cro93]). Moggi shows that there is a similar correspondence between the computational λ-calculus and the class of finite product categories with a strong monad and certain exponentials (if the functor component of the monad is T, then every exponential of the form $T(B)^A$ must exist, for objects A and B). Terms of the form $[M]$ are interpreted using the unit of the monad, allowing values to be converted into computations. Terms of the form $\mathsf{let}\, x \Leftarrow M \,\mathsf{in}\, N$ are interpreted using the functor and multiplication of the monad, allowing computations to be composed.

Cenciarelli and Moggi [CM93] propose structuring complex denotational semantics as a sequence of computational λ-calculi $ML(\Sigma_1), \ldots, ML(\Sigma_n)$, with signatures $\Sigma_1, \ldots, \Sigma_n$, and a sequence of syntactic translations $\phi : ML(\Sigma_i) \to ML(\Sigma_{i+1})$. If there is a syntactic translation from the programming language to $ML(\Sigma_0)$, and a denotational model of the computational λ-calculus $ML(\Sigma_n)$, then a denotational semantics for the programming language can be given via the translations. The benefit of using the computational λ-calculus for this approach is that the unit and sequencing constructors are factored out of the signatures.

Wadler [Wad92] shows that a mechanism for sequencing) computations is useful for introducing notions of computations such as state or input/output into a lazy functional programming language. Independently of Moggi and Wadler, Spivey [Spi89, Spi90] also observes that list comprehensions and exceptions form monads.

The computational λ-calculus has a natural call-by-name operational semantics with strict sequencing (see sections 3.4 and 3.5). For example, the term $\mathsf{let}\, x \Leftarrow M \,\mathsf{in}\, N$ is evaluated by evaluating M, substituting the result for x in N, and then evaluating the result of that substitution. Crole and Gordon [Gor94, CG95] use this operational semantics for deterministic λ-calculi with input/output, and Jeffrey [Jef99] uses it for a non-deterministic λ-calculus.

A call-by-name operational semantics with strict sequencing for the non-deterministic λ-calculus $\mathscr{L}$ provides an adequate degree of control over the resolution of non-determinism, whilst still permitting definitions of semantic relations as applicative similarity and bisimilarity (see the discussion on page 19).

3.3 Type Assignment

The type assignment system for $\mathscr{L}$ is adapted from those of PCF and the computational λ-calculus following [Gor94, CG95, Jef99]. We define the system and introduce some useful abbreviations for terms.

A type is assigned to a term with respect to an *environment*, which determines types for a finite collection of variables that may be free in the term. It suffices to consider a finite collection because we are primarily interested in terms that have no free variables, and there is no way to bind an infinite collection of free variables in a term.

Definition 3.3.1 An ***environment*** is a finite partial function from *Var* to types. The symbols Γ and Δ range over environments. When $x \notin Dom(\Gamma)$, the environment that extends Γ by mapping x to σ is denoted $\Gamma, x : \sigma$. The environment Γ, Δ is defined similarly for environments Γ and Δ such that $Dom(\Gamma) \cap Dom(\Delta) = \emptyset$. The empty environment is denoted by $\emptyset$.

Definition 3.3.2 The type assignment judgement, term M has type σ in environment Γ, written $\Gamma \vdash M : \sigma$, is defined inductively by the rules in figure 3.3. The notation $\Gamma \vdash M = N : \sigma$ means that the terms M and N are equal (up to α-conversion), and both $\Gamma \vdash M : \sigma$ and $\Gamma \vdash N : \sigma$ can be derived. Define the set of well-typed terms $\mathscr{L}$, the set of well-typed, closed terms $\mathscr{L}_0$, and the set of well-typed, closed, canonical terms Can_0 by:

$$\begin{aligned} \mathscr{L} &\stackrel{\text{def}}{=} \{M \mid \exists \Gamma, \sigma.\, \Gamma \vdash M : \sigma\} \\ \mathscr{L}_0 &\stackrel{\text{def}}{=} \{M \mid \exists \sigma.\, \emptyset \vdash M : \sigma\} \\ Can_0 &\stackrel{\text{def}}{=} \{K \mid \exists \sigma.\, \emptyset \vdash K : \sigma \wedge K \text{ canonical}\} \end{aligned}$$

Henceforth, terms are assumed to be well-typed. A term M is a ***program*** if $M \in \mathscr{L}_0$, and in this case the empty environment is omitted from the type assignment judgement, i.e., we write $\vdash M : \sigma$ for $\emptyset \vdash M : \sigma$.

The fixed-point and erratic choice term constructors are always assigned a computation type. In sections 3.4 and 3.6, it is shown that there is no way to use the other constructors to introduce non-termination or non-determinism at value types.

The type assignment system satisfies the usual properties such as weakening and contraction.

Lemma 3.3.3 Let Γ and Δ be environments such that $Dom(\Gamma) \cap Dom(\Delta) = \emptyset$. Then:

1. If $\Gamma \vdash M : \sigma$, then $Fv(M) \subseteq Dom(\Gamma)$.
2. If $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$, then $\sigma = \tau$.
3. If $\Gamma \vdash M : \sigma$, then $\Gamma, \Delta \vdash M : \sigma$.
4. If $\Gamma, \Delta \vdash M : \sigma$ and $Fv(M) \subseteq Dom(\Gamma)$, then $\Gamma \vdash M : \sigma$.
5. If $\Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \tau$ and $\Gamma \vdash N_i : \sigma_i$, for $1 \le i \le n$, then $\Gamma \vdash M[N_1, \ldots, N_n / x_1, \ldots, x_n] : \tau$.

Proof In each case, by induction on the type assignment derivation for M. (2) makes essential use of the type annotations. (3) uses the bound variable naming convention to ensure that the bound variables of M do not appear in Δ. □

$$\Gamma \vdash x : \sigma \quad (\Gamma(x) = \sigma)$$

$$\frac{\Gamma \vdash M : \sigma_m}{\Gamma \vdash \mathsf{inj}\, m, \mathsf{sum}\,\langle \sigma_n \mid n < \kappa \rangle\, \mathsf{of}\, M : \mathsf{sum}\,\langle \sigma_n \mid n < \kappa \rangle} \quad (m < \kappa)$$

$$\frac{\Gamma \vdash M : \mathsf{sum}\,\langle \sigma_n \mid n < \kappa \rangle \qquad \{\Gamma, x_n : \sigma_n \vdash N_n : \tau \mid n < \kappa\}}{\Gamma \vdash \mathsf{case}\, M\, \mathsf{of}\, \langle x_n.N_n \mid n < \kappa \rangle : \tau}$$

$$\frac{\{\Gamma \vdash M_n : \sigma_n \mid n < \kappa\}}{\Gamma \vdash \mathsf{tuple}\,\langle M_n \mid n < \kappa \rangle : \mathsf{prod}\,\langle \sigma_n \mid n < \kappa \rangle}$$

$$\frac{\Gamma \vdash M : \mathsf{prod}\,\langle \sigma_n \mid n < \kappa \rangle}{\Gamma \vdash \mathsf{proj}\, m\, \mathsf{of}\, M : \sigma_m} \quad (m < \kappa)$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x{:}\sigma. M : \sigma \to \tau}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash [M] : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash M : \mathsf{P}_\perp(\sigma) \qquad \Gamma, x : \sigma \vdash N : \mathsf{P}_\perp(\tau)}{\Gamma \vdash \mathsf{let}\, x{:}\sigma \Leftarrow M\, \mathsf{in}\, N : \mathsf{P}_\perp(\tau)}$$

$$\frac{\Gamma, x : \mathsf{P}_\perp(\sigma) \vdash M : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \mathsf{fix}\, x{:}\mathsf{P}_\perp(\sigma). M : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\{\Gamma \vdash M_n : \sigma \mid n < \kappa\}}{\Gamma \vdash ?\langle M_n \mid n < \kappa \rangle : \mathsf{P}_\perp(\sigma)}$$

Figure 3.3: Type assignment

Although type annotations within terms are necessary to ensure uniqueness of types, we often omit them for brevity.

Lemma 3.3.4 describes an equivalence between terms with successive substitutions. The type assignment judgements are used only to restrict the free variables appearing in terms, and so a slightly more general statement could be given for untyped terms.

Lemma 3.3.4 Let Γ and $\Delta = x_1 : \sigma_1, \ldots x_n : \sigma_n$ be environments such that $\Gamma \cap \Delta = \emptyset$. If the variables $y_1, \ldots, y_m$ are not in the domains of Γ or Δ, and:

- $\Gamma, \Delta, y_1 : \tau_1, \ldots, y_m : \tau_m \vdash L : \sigma$
- For all $1 \leq i \leq m$, $\Gamma, \Delta \vdash M_i : \tau_i$
- For all $1 \leq i \leq n$, $\Gamma \vdash N_i : \sigma_i$

Then, with $\vec{M} = M_1, \ldots, M_m$ and $\vec{N} = N_1, \ldots, N_n$:

$$\Gamma \vdash L[\vec{M}/\vec{y}][\vec{N}/\vec{x}] = L[\vec{N}/\vec{x}][M_1[\vec{N}/\vec{x}], \ldots, M_m[\vec{N}/\vec{x}]/y_1, \ldots, y_m] : \sigma$$

Proof The equality is proven by induction on the term L. The type assignment follows from lemma 3.3.3. □

We now consider abbreviations for frequently used terms. The abbreviations defined in Figure 3.4 include constants for the ground types unit, bool, and nat, as well as logical and arithmetical operators. Variables that appear only on the right-hand side of definitions are fresh.

In addition, every non-empty set of natural numbers determines a program of type $\mathsf{P}_\bot(\mathsf{nat})$. For $A \subseteq_{\mathrm{ne}} \omega$, let $\langle a_n \mid n < \kappa \rangle$ be the unique strictly increasing sequence of natural numbers that enumerates the elements of A. Then define:

$$?\underline{A} \stackrel{\mathrm{def}}{=} ?\langle \underline{a_n} \mid n < \kappa \rangle$$

For example, $?\underline{\omega} = ?\langle \underline{n} \mid n < \omega \rangle$.

The terms $\star$, false, true, $\underline{n}$ are canonical, for all $n \in \omega$. The boolean terms false and true are distinct from the numerical terms $\underline{0}$ and $\underline{1}$ because their type annotations differ.

The definitions of the arithmetic operators illustrate how functions on ω determine terms. More generally, any function $f : \omega^n \to \omega$ determines a term $x_1 : \mathsf{nat}, \ldots, x_n : \mathsf{nat} \vdash M : \mathsf{nat}$ composed of nested case statements.

With the exception of conditional terms $\mathsf{if}\, L\, \mathsf{then}\, M\, \mathsf{else}\, N$, the third group of definitions in figure 3.4 create or manipulate computations. Terms of the form Ω_σ never terminate. The call-by-value abbreviation $\lambda^v x{:}\sigma.M$ combines the default call-by-name abstraction with strict sequencing to evaluate the argument before it is substituted into M. The "function" versions of the sequencing and fixed-point constructs $\mathsf{flet}_\sigma x{:}\tau \Leftarrow M \,\mathsf{in}\, N$ and $\mathsf{ffix}_\sigma f{:}\tau.M$ are used to define recursive functions with type $\sigma \to \mathsf{P}_\bot(\tau)$. In contrast, a program of the form $\mathsf{fix}\, x.M$ would have type $\mathsf{P}_\bot(\sigma \to \mathsf{P}_\bot(\tau))$. The call-by-value abstractions and the "function" sequencing and fixed-point constructs are always canonical terms.

$$
\begin{aligned}
\star &\stackrel{\text{def}}{=} \mathsf{tuple}\,\langle\rangle \\
\mathsf{false} &\stackrel{\text{def}}{=} \mathsf{inj}\,0,\mathsf{bool\ of}\,\star \\
\mathsf{true} &\stackrel{\text{def}}{=} \mathsf{inj}\,1,\mathsf{bool\ of}\,\star \\
\underline{n} &\stackrel{\text{def}}{=} \mathsf{inj}\,n,\mathsf{nat\ of}\ \star \qquad (n\in\omega)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{not}\,(M) &\stackrel{\text{def}}{=} \mathsf{case}\,M\,\mathsf{of}\ \langle x_0.\mathsf{true},x_1.\mathsf{false}\rangle \\
\mathsf{or}\,(M,N) &\stackrel{\text{def}}{=} \mathsf{case}\,M\,\mathsf{of}\ \langle x_0.\mathsf{case}\,N\,\mathsf{of}\ \langle y_0.\mathsf{false},y_1.\mathsf{true}\rangle,x_1.\mathsf{true}\rangle \\
\mathsf{and}\,(M,N) &\stackrel{\text{def}}{=} \mathsf{case}\,M\,\mathsf{of}\ \langle x_0.\mathsf{false},x_1.\mathsf{case}\,N\,\mathsf{of}\ \langle y_0.\mathsf{false},y_1.\mathsf{true}\rangle\rangle \\
\mathsf{plus}\,(M,N) &\stackrel{\text{def}}{=} \mathsf{case}\,M\,\mathsf{of}\ \langle x_i.\mathsf{case}\,N\,\mathsf{of}\ \langle y_j.\underline{i+j}\mid j<\omega\rangle\mid i<\omega\rangle \\
\mathsf{minus}\,(M,N) &\stackrel{\text{def}}{=} \mathsf{case}\,M\,\mathsf{of}\ \langle x_i.\mathsf{case}\,N\,\mathsf{of}\ \langle y_j.L_{i,j}\mid j<\omega\rangle\mid i<\omega\rangle \\
&\qquad \text{where}\quad L_{i,j}=\begin{cases}\underline{i-j} & \text{if } i\geq j\\ \underline{0} & \text{if } i<j\end{cases} \\
\mathsf{eq}\,(M,N) &\stackrel{\text{def}}{=} \mathsf{case}\,M\,\mathsf{of}\ \langle x_i.\mathsf{case}\,N\,\mathsf{of}\ \langle y_j.L_{i,j}\mid j<\omega\rangle\mid i<\omega\rangle \\
&\qquad \text{where}\quad L_{i,j}=\begin{cases}\mathsf{false} & \text{if } i\neq j\\ \mathsf{true} & \text{if } i=j\end{cases} \\
\mathsf{lt}\,(M,N) &\stackrel{\text{def}}{=} \mathsf{case}\,M\,\mathsf{of}\ \langle x_i.\mathsf{case}\,N\,\mathsf{of}\ \langle y_j.L_{i,j}\mid j<\omega\rangle\mid i<\omega\rangle \\
&\qquad \text{where}\quad L_{i,j}=\begin{cases}\mathsf{false} & \text{if } i\geq j\\ \mathsf{true} & \text{if } i<j\end{cases}
\end{aligned}
$$

$$
\begin{aligned}
\Omega_\sigma &\stackrel{\text{def}}{=} \mathsf{fix}\,x{:}\sigma.\,x \\
\lambda^v x{:}\sigma.\,M &\stackrel{\text{def}}{=} \lambda y{:}\mathsf{P}_\bot(\sigma).\,\mathsf{let}\,x{:}\sigma\Leftarrow y\,\mathsf{in}\,M \\
\mathsf{if}\,L\,\mathsf{then}\,M\,\mathsf{else}\,N &\stackrel{\text{def}}{=} \mathsf{case}\,L\,\mathsf{of}\ \langle x_0.N,x_1.M\rangle \\
\mathsf{flet}_\sigma\,x{:}\tau\Leftarrow M\,\mathsf{in}\,N &\stackrel{\text{def}}{=} \lambda y{:}\sigma.\,\mathsf{let}\,x{:}\tau\Leftarrow M\,\mathsf{in}\,N\,y \\
\mathsf{ffix}_\sigma\,f{:}\tau.\,M &\stackrel{\text{def}}{=} \mathsf{flet}_\sigma\,f{:}\tau\Leftarrow(\mathsf{fix}\,g{:}\mathsf{P}_\bot(\tau).\,[\mathsf{flet}_\sigma\,f{:}\tau\Leftarrow g\,\mathsf{in}\,M])\,\mathsf{in}\,M \\
M\cup N &\stackrel{\text{def}}{=} \mathsf{let}\,x{:}\mathsf{bool}\Leftarrow ?\langle\mathsf{false},\mathsf{true}\rangle\,\mathsf{in}\,\mathsf{if}\,x\,\mathsf{then}\,M\,\mathsf{else}\,N
\end{aligned}
$$

Figure 3.4: Abbreviated terms

$$\Gamma \vdash \star : \mathsf{unit} \qquad \Gamma \vdash \mathsf{false} : \mathsf{bool}$$

$$\Gamma \vdash \mathsf{true} : \mathsf{bool} \qquad \Gamma \vdash \underline{n} : \mathsf{nat} \quad (n \in \omega)$$

$$\frac{\Gamma \vdash M : \mathsf{bool}}{\Gamma \vdash \mathsf{not}\,(M) : \mathsf{bool}}$$

$$\frac{\Gamma \vdash M : \mathsf{bool} \qquad \Gamma \vdash N : \mathsf{bool}}{\Gamma \vdash \mathsf{or}\,(M,N) : \mathsf{bool}} \qquad \frac{\Gamma \vdash M : \mathsf{bool} \qquad \Gamma \vdash N : \mathsf{bool}}{\Gamma \vdash \mathsf{and}\,(M,N) : \mathsf{bool}}$$

$$\frac{\Gamma \vdash M : \mathsf{nat} \qquad \Gamma \vdash N : \mathsf{nat}}{\Gamma \vdash \mathsf{plus}\,(M,N) : \mathsf{nat}} \qquad \frac{\Gamma \vdash M : \mathsf{nat} \qquad \Gamma \vdash N : \mathsf{nat}}{\Gamma \vdash \mathsf{minus}\,(M,N) : \mathsf{nat}}$$

$$\frac{\Gamma \vdash M : \mathsf{nat} \qquad \Gamma \vdash N : \mathsf{nat}}{\Gamma \vdash \mathsf{eq}\,(M,N) : \mathsf{bool}} \qquad \frac{\Gamma \vdash M : \mathsf{nat} \qquad \Gamma \vdash N : \mathsf{nat}}{\Gamma \vdash \mathsf{lt}\,(M,N) : \mathsf{bool}}$$

$$\Gamma \vdash \Omega_{\mathsf{P}_\bot(\sigma)} : \mathsf{P}_\bot(\sigma)$$

$$\frac{\Gamma, x : \sigma \vdash M : \mathsf{P}_\bot(\tau)}{\Gamma \vdash \lambda^v x{:}\sigma.M : \mathsf{P}_\bot(\sigma) \to \mathsf{P}_\bot(\tau)}$$

$$\frac{\Gamma \vdash L : \mathsf{bool} \qquad \Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathsf{if}\,L\,\mathsf{then}\,M\,\mathsf{else}\,N : \sigma}$$

$$\frac{\Gamma \vdash M : \mathsf{P}_\bot(\tau_1) \qquad \Gamma, x : \tau_1 \vdash N : \sigma \to \mathsf{P}_\bot(\tau_2)}{\Gamma \vdash \mathsf{flet}_\sigma\, x{:}\tau_1 \Leftarrow M\,\mathsf{in}\,N : \sigma \to \mathsf{P}_\bot(\tau_2)}$$

$$\frac{\Gamma, f : \sigma \to \mathsf{P}_\bot(\tau) \vdash M : \sigma \to \mathsf{P}_\bot(\tau)}{\Gamma \vdash \mathsf{ffix}_\sigma\, f{:}\sigma \to \mathsf{P}_\bot(\tau).M : \sigma \to \mathsf{P}_\bot(\tau)}$$

$$\frac{\Gamma \vdash M : \mathsf{P}_\bot(\sigma) \qquad \Gamma \vdash N : \mathsf{P}_\bot(\sigma)}{\Gamma \vdash M \cup N : \mathsf{P}_\bot(\sigma)}$$

$$\Gamma \vdash ?\underline{A} : \mathsf{P}_\bot(\mathsf{nat}) \quad (A \subseteq_{\mathrm{ne}} \omega)$$

Figure 3.5: Type assignment for abbreviated terms

The abbreviation $M \cup N$ serves as the binary erratic choice constructor for $\mathscr{L}$. It is also possible to define $M \cup N \overset{\text{def}}{=} \mathsf{let}\, x \Leftarrow ?\langle M,N\rangle\, \mathsf{in}\, x$. However, the latter definition would cause problems when we consider fragments of $\mathscr{L}$, because a fragment that contains $M \cup N$ would also have to contain $?\langle M,N\rangle$. The definition in figure 3.4 is more useful because it only requires a fragment to contain $?\langle \mathsf{false}, \mathsf{true}\rangle$.

Derived type assignment rules for the abbreviations are given in figure 3.5.

3.4 Reduction Semantics

In this section we present a reduction semantics for $\mathscr{L}$ and a novel treatment of reduction contexts, and then discuss some of the properties of reduction. The reduction semantics follows

$$\begin{array}{rcl} E & ::= & \mathsf{case}\,(-)\,\mathsf{of}\,\langle x_n.N_n \mid n < \kappa\rangle \\ & \mid & \mathsf{proj}\,n\,\mathsf{of}\,(-) \\ & \mid & (-)\,N \\ & \mid & \mathsf{let}\,x{:}\sigma \Leftarrow (-)\,\mathsf{in}\,N \end{array}$$

Figure 3.6: Reduction constructors

Plotkin's [Plo81] guidelines for a *structural operational semantics* because the behaviour of a term M depends only upon the behaviour of terms formed from subterms of M.

The reduction semantics consists of a *reduction relation* $\rightarrowtriangle \subseteq \mathscr{L} \times \mathscr{L}$ and a finer *deterministic reduction relation* $\rightarrowtriangle_{\mathrm{det}} \subseteq \rightarrowtriangle$. Reductions are permitted on open terms to facilitate a proof technique used in chapter 5. The relations are defined by induction from a collection of rule schema. Reduction constructors are used to specify where reduction can take place, and this simplifies the rule schema because only one rule schema is needed to express that reduction may take place inside a reduction constructor. Felleisen and Friedman [FF86] introduced this approach to defining reduction relations, as well as the terminology *reduction context* for nested reduction constructors. Reduction contexts do not require a treatment of variable-capturing contextual substitution (see [Pit97]).

Definition 3.4.1 The ***reduction constructors*** are defined in figure 3.6. They contain exactly one occurrence of a distinguished symbol $(-) \notin \mathscr{L}$. When E is a reduction constructor and M is a term, we write $E(M)$ for the term that is obtained by replacing the occurrence of $(-)$ by M. This induces a well-defined function on α-equivalence classes of terms.

Reduction constructors determine a *reduction strategy*, or a path through each abstract syntax tree to the next redex, and so each term has at most one redex. Only one rule schema can apply to a redex, but reduction is non-deterministic because the rule schema for indexed erratic choice may be instantiated in more than one way. The deterministic reduction relation is obtained by excluding the reduction rule for indexed erratic choice.

It is sometimes inconvenient to work with reduction contexts in proofs. For this reason, we define a *blocking relation* $\between \subseteq \mathscr{L} \times \mathit{Var}$ and a *blocked substitution* operation. A term M is blocked at a variable x, written $M \between x$, if there is a reduction context $\vec{E}(-) = E_1(E_2(\ldots E_n(-)\ldots))$ such that $M = \vec{E}(x)$. The blocked substitution of N for the blocked variable x in M satisfies $M[x \mapsto N] = \vec{E}(N)$, i.e., substitution only happens at the occurrence of x corresponding to the distinguished symbol $(-)$.

Definition 3.4.2

- The ***reduction relation*** $\rightarrowtriangle \subseteq \mathscr{L} \times \mathscr{L}$ (also known as a ***transition relation*** or a ***small-step relation***) is defined inductively from the rules in figure 3.7.

- The ***deterministic reduction relation*** $\rightarrowtriangle_{\mathrm{det}} \subseteq \rightarrowtriangle$ is defined inductively from the rules in figure 3.7 with the rule $?\langle M_n \mid n < \kappa\rangle \rightarrowtriangle [M_m]$ removed.

$$\frac{M \rightarrowtriangle N}{E(M) \rightarrowtriangle E(N)}$$

$$\mathsf{case}\,(\mathsf{inj}\,m\,\mathsf{of}\,M)\,\mathsf{of}\,\langle x_n.N_n \mid n < \kappa\rangle \rightarrowtriangle N_m[M/x_m]$$

$$\mathsf{proj}\,m\,\mathsf{of}\,(\mathsf{tuple}\,\langle M_n \mid n < \kappa\rangle) \rightarrowtriangle M_m$$

$$(\lambda x.M)N \rightarrowtriangle M[N/x]$$

$$\mathsf{let}\,x \Leftarrow [M]\,\mathsf{in}\,N \rightarrowtriangle N[M/x]$$

$$\mathsf{fix}\,x.M \rightarrowtriangle M[\mathsf{fix}\,x.M/x]$$

$$?\langle M_n \mid n < \kappa\rangle \rightarrowtriangle [M_m] \quad (m < \kappa)$$

Figure 3.7: Reduction relation

- A term M ***converges*** to a term N if $M \rightarrowtriangle^* N$ and N has no reductions.
- A term M_0 ***diverges***, denoted $M_0 \rightarrowtriangle^\omega$, if there exists an ω-sequence of terms $\langle M_n \mid n < \omega\rangle$ such that $M_n \rightarrowtriangle M_{n+1}$, for all $n \in \omega$.
- The ***blocking relation*** $\lightning \subseteq \mathscr{L} \times \mathit{Var}$ is defined inductively from the rules:

$$x \lightning x \qquad\qquad \frac{M \lightning x}{E(M) \lightning x}$$

  If $M \lightning x$, then we say that the term M is ***blocked*** on x.
- For terms $\Gamma, x : \tau \vdash M : \sigma$ and $\Gamma, x : \tau \vdash N : \tau$ such that $M \lightning x$, define the ***blocked substitution*** of N for the blocked occurrence of x in M, denoted $M[x \mapsto N]$, by induction on the derivation of $M \lightning x$:

$$x[x \mapsto N] \stackrel{\text{def}}{=} N$$
$$E(M)[x \mapsto N] \stackrel{\text{def}}{=} E(M[x \mapsto N])$$

Example 3.4.3 demonstrates the use of the sequencing construct for controlling resolution of non-determinism.

Example 3.4.3 Consider the following programs of type $\mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{nat})$:

$$M \stackrel{\text{def}}{=} \lambda x{:}\mathsf{P}_\perp(\mathsf{nat}).\,\mathsf{let}\,y \Leftarrow x\,\mathsf{in}\,[\mathsf{plus}\,(y,y)]$$
$$N \stackrel{\text{def}}{=} \lambda x{:}\mathsf{P}_\perp(\mathsf{nat}).\,\mathsf{let}\,y \Leftarrow x\,\mathsf{in}\,\mathsf{let}\,z \Leftarrow x\,\mathsf{in}\,[\mathsf{plus}\,(y,z)]$$

When M is applied to $?\langle \underline{0}, \underline{1}\rangle$, a possible reduction sequence is:

$$\begin{aligned} M\,?\langle \underline{0}, \underline{1}\rangle &\rightarrowtriangle \mathsf{let}\,y \Leftarrow ?\langle \underline{0}, \underline{1}\rangle\,\mathsf{in}\,[\mathsf{plus}\,(y,y)] \\ &\rightarrowtriangle \mathsf{let}\,y \Leftarrow [\underline{0}]\,\mathsf{in}\,[\mathsf{plus}\,(y,y)] \\ &\rightarrowtriangle [\mathsf{plus}\,(\underline{0}, \underline{0})] \end{aligned}$$

$$\Omega \rightarrowtriangle_{\mathrm{det}} \Omega$$
$$(\lambda^v x.M)N \rightarrowtriangle_{\mathrm{det}} \mathsf{let}\, x \Leftarrow N \,\mathsf{in}\, M$$
$$\mathsf{if\,false\,then}\, M \,\mathsf{else}\, N \rightarrowtriangle_{\mathrm{det}} N$$
$$\mathsf{if\,true\,then}\, M \,\mathsf{else}\, N \rightarrowtriangle_{\mathrm{det}} M$$
$$(\mathsf{flet}\, x \Leftarrow L \,\mathsf{in}\, M)N \rightarrowtriangle_{\mathrm{det}} \mathsf{let}\, x \Leftarrow L \,\mathsf{in}\, MN$$
$$(\mathsf{ffix}\, f.M)N \rightarrowtriangle^{+}_{\mathrm{det}} (M[\mathsf{ffix}\, f.M/f])N$$
$$M \cup N \rightarrowtriangle^{+} M$$
$$M \cup N \rightarrowtriangle^{+} N$$
$$?\underline{A} \rightarrowtriangle [\underline{n}] \quad (n \in A)$$

Figure 3.8: Derived reduction rules

The only other reduction sequence gives $M\,?\langle\underline{0},\underline{1}\rangle \rightarrowtriangle^{+} [\mathsf{plus}\,(\underline{1},\underline{1})]$. Similarly, for $N\,?\langle\underline{0},\underline{1}\rangle$, we have $N\,?\langle\underline{0},\underline{1}\rangle \rightarrowtriangle^{+} [\mathsf{plus}\,(\underline{0},\underline{0})]$ and $N\,?\langle\underline{0},\underline{1}\rangle \rightarrowtriangle^{+} [\mathsf{plus}\,(\underline{1},\underline{1})]$. In addition, there is a reduction sequence in which two different choices are made:

$$\begin{aligned}
N\,?\langle\underline{0},\underline{1}\rangle &\rightarrowtriangle \mathsf{let}\, y \Leftarrow ?\langle\underline{0},\underline{1}\rangle \,\mathsf{in}\, \mathsf{let}\, z \Leftarrow ?\langle\underline{0},\underline{1}\rangle \,\mathsf{in}\, [\mathsf{plus}\,(y,z)] \\
&\rightarrowtriangle \mathsf{let}\, y \Leftarrow [\underline{0}] \,\mathsf{in}\, \mathsf{let}\, z \Leftarrow ?\langle\underline{0},\underline{1}\rangle \,\mathsf{in}\, [\mathsf{plus}\,(y,z)] \\
&\rightarrowtriangle \mathsf{let}\, z \Leftarrow ?\langle\underline{0},\underline{1}\rangle \,\mathsf{in}\, [\mathsf{plus}\,(\underline{0},z)] \\
&\rightarrowtriangle \mathsf{let}\, z \Leftarrow [\underline{1}] \,\mathsf{in}\, [\mathsf{plus}\,(\underline{0},z)] \\
&\rightarrowtriangle [\mathsf{plus}\,(\underline{0},\underline{1})]
\end{aligned}$$

Reversing the two choices gives a fourth reduction sequence $N\,?\langle\underline{0},\underline{1}\rangle \rightarrowtriangle^{+} [\mathsf{plus}\,(\underline{1},\underline{0})]$.

Appropriate reduction rules can be derived for the arithmetic operators. For example, it can be shown that for all programs $M,N \in \mathscr{L}_0$, if $M \rightarrowtriangle^{*} \underline{m}$ and $N \rightarrowtriangle^{*} \underline{n}$, then $\mathsf{plus}\,(M,N) \rightarrowtriangle^{+} \underline{m+n}$. Note that the arithmetic expressions in example 3.4.3 are not reduced unless they are placed inside another context because, e.g., $[\mathsf{plus}\,(\underline{0},\underline{1})]$ is canonical.

The reduction rules in figure 3.8 can be derived for the other abbreviations in figure 3.4. In addition, the conditional construct is a reduction constructor $\mathsf{if}\,(-)\,\mathsf{then}\, M \,\mathsf{else}\, N$. Call-by-value abstractions and the "function" sequencing and fixed-point constructs do not form reduction constructors because they immediately reduce to a sequencing construct when applied to a term.

The derived reduction rules have the expected properties. For example, Ω always diverges, $\mathsf{ffix}\, f.M$ is a fixed-point for recursively-defined functions, binary erratic choice $M \cup N$ reduces to either M or N, and the erratic choice $?\underline{A}$ of a set of numbers $A \subseteq_{\mathrm{ne}} \omega$ reduces to any number in A.

The reduction relation preserves the type of a term, and, if a term of value type has a reduction, then it must be a deterministic reduction.

Lemma 3.4.4 (Subject Reduction) If $\Gamma \vdash M : \sigma$ and $M \rightarrow N$ then:

1. $\Gamma \vdash N : \sigma$
2. If σ is a value type, then $M \rightarrow_{\text{det}} N$.

Proof By induction on the derivation of $M \rightarrow N$. □

The reduction relation determines a TS $\langle \mathscr{L}, \rightarrow \rangle$ with terms as states (or configurations). The terms are partitioned according to their type, and the subject reduction property implies that the reduction relation does not cross partitions. The TS has a complex structure. For example, there are terms that diverge and terms that have infinitely many successors. However, much of the reduction behaviour of terms is ignored, because only the convergence and divergence properties of programs are considered in chapters 4 and 5. Other approaches based on higher-order weak bisimilarity, such as [FHJ95, Jef99], are more sensitive to reductions (they become τ-labelled transitions).

Lemma 3.4.5 shows that the blocking relation only relates a term to its free variables, and so programs are never blocked. Blocked substitution of a term N for the blocked occurrence of x in M only replaces that occurrence of x. There may be other free occurrences of x in M and free occurrences of x in N.

Lemma 3.4.5 If $\Gamma \vdash M : \tau$ and $M \between x$, then $x \in Dom(\Gamma)$. If in addition $\Gamma(x) = \sigma$ and $\Gamma \vdash N : \sigma$, then $\Gamma \vdash M[x \mapsto N] : \tau$.

Proof Prove $x \in Fv(M)$ by induction on the derivation of $M \between x$, and then apply lemma 3.3.3(1). The second part is also proven by induction on the derivation of $M \between x$. □

The "progression" property for the reduction relation tells us that every program is either canonical or has at least one reduction to another program. More generally, every term is canonical, has at least one reduction, or is blocked on a free variable.

Lemma 3.4.6 If $\Gamma \vdash M : \sigma$, then exactly one of the following holds:

1. M is canonical; or
2. There exists N such that $M \rightarrow N$, and the term N is unique if $M \rightarrow_{\text{det}} N$; or
3. There exists $x \in Dom(\Gamma)$ such that $M \between x$.

Proof By induction on the derivation of $\Gamma \vdash M : \sigma$. □

It follows immediately from lemmas 3.4.4 and 3.4.6 that a term of value type can reduce to at most one other term. The majority of terms that have reductions with respect to the reduction relation, but not the deterministic reduction relation, can reduce to more than one term because they are indexed erratic choice terms inside reduction contexts. The exceptions are singleton erratic choices, e.g., $?\langle\star\rangle$ only reduces to $[\star]$.

By iterating lemma 3.4.6, we can show that every program diverges or converges to some canonical program (possibly both).

Lemma 3.4.7 If M is a program, then M converges to some canonical program or M diverges.

Proof Let $X \subseteq \mathscr{L}_0$ be the set of programs that cannot converge to a canonical program:

$$X \stackrel{\text{def}}{=} \{M \in \mathscr{L}_0 \mid \nexists K \in Can_0 . M \rightarrowtriangle^* K\}$$

Consider a program $M_n \in X$. The program M_n cannot be canonical or blocked, and so, by lemma 3.4.6, there exists M_{n+1} such that $M_n \rightarrowtriangle M_{n+1}$. If M_{n+1} converged to some canonical program, then M_n could as well, contradicting $M_n \in X$. Therefore $M_{n+1} \in X$. We can then construct an ω-sequence of programs $\langle M_n \mid n \in \omega\rangle$ such that $M_n \rightarrowtriangle M_{n+1}$ and $M_n \in X$, for all $n \in \omega$. Therefore a program must diverge whenever it cannot converge to a canonical program.

□

The reduction relation is certainly not Church-Rosser (see [Bar84, HS86]) because of erratic choice terms such as $?\langle \mathsf{false}, \mathsf{true}\rangle$. The deterministic reduction relation is trivially Church-Rosser because each term has at most one deterministic reduction.

In a deterministic setting, programs M and N are said to be *Kleene equivalent* (see [Pit97, Las98b]) if, for all canonical programs K, M converges to K if and only if N converges to K. Non-divergent terms are Kleene equivalent if and only if they are related by the reflexive, transitive closure of the union of the reduction relation and its dual. In contrast, the relation $(\rightarrowtriangle \cup \rightarrowtriangle^{\mathrm{op}})^*$ is not useful for $\mathscr{L}$ because it relates many programs with the same computation type, e.g., $[\mathsf{false}] \leftarrowtriangle ?\langle \mathsf{false}, \mathsf{true}\rangle \rightarrowtriangle [\mathsf{true}]$. The definition of Kleene equivalence can be replayed for $\mathscr{L}$, but it is not sensitive to some of the divergence properties of programs. For example, the programs $\Omega \cup [\star]$ and $[\star]$, both of type $\mathsf{P}_\perp(\mathsf{unit})$, are Kleene equivalent, but only the first program can diverge. Taking the divergence properties of programs into consideration leads to a family of equivalences (and preorders) based upon Kleene equivalence that corresponds to the family of variants of similarity and bisimilarity for LTSWDs (see definition 2.4.11).

Lemma 3.4.8 establishes cardinality bounds upon the set of canonical programs to which a program can converge.

Lemma 3.4.8 Consider a program M.

1. The set of canonical programs $\{K \mid M \text{ converges to } K\}$ is countable.
2. If M does not contain any occurrences of infinite indexed erratic choice term constructors and M does not diverge, then the set of canonical programs $\{K \mid M \text{ converges to } K\}$ is finite.

Proof

1. Each state has a countable set of successors with respect to reduction, because indexed erratic choice is restricted to finite or ω-sequences of terms. So, for all $n \in \omega$, the set of programs related to M by $\rightarrowtriangle^n$ is countable. Therefore $\bigcup\{\{N \mid M \rightarrowtriangle^n N\} \mid n \in \omega\}$ is also countable. It is a superset of $\{K \mid M \text{ converges to } K\}$ and we are done.

2. If $\{K \mid M \text{ converges to } K\}$ is infinite, then the synchronisation tree with root M obtained by unfolding the TS $\langle \mathscr{L}, \twoheadrightarrow \rangle$ has an infinite set of nodes. The synchronisation tree is finitely-branching, and so, by König's lemma (lemma 2.1.10), there exists an infinite sequence of reduction steps from M, i.e., M diverges.

$\square$

Erratic non-determinism is sometimes classified into finite non-determinism and countably infinite non-determinism. For example, $?\langle \mathsf{false}, \mathsf{true} \rangle$ and $?\langle \underline{0}, \underline{1}, \ldots, \underline{n} \rangle$, for $n \in \omega$, are finitely non-deterministic, whereas $?\underline{\omega}$ is countably non-deterministic. This classification is based upon the maximum cardinality of non-divergent programs constructed using the non-deterministic operator. Example 3.4.9 illustrates the importance of the non-divergence condition.

Example 3.4.9 Finite non-determinism can be used to construct programs that converge to a countably infinite set of canonical programs. Lemma 3.4.8 ensures that such programs always diverge. For example, define the program M of type $\mathsf{P}_\perp(\mathsf{nat})$ by:

$$M \stackrel{\text{def}}{=} \mathsf{fix}\, x.\, [\underline{0}] \cup (\mathsf{let}\, y \Leftarrow x\, \mathsf{in}\, [\mathsf{plus}\,(y, \underline{1})])$$

For any $n \in \omega$, there is a program N such that $M \twoheadrightarrow^+ [N]$ and $N \twoheadrightarrow^* \underline{n}$, but M can also diverge. The program M has the same convergence and divergence behaviour as $\Omega \cup ?\underline{\omega}$. If divergence behaviour is ignored, these programs are also identified with $?\underline{\omega}$, in which case the distinction between finite non-determinism and countable non-determinism is lost. Finally, if ambiguous choice is used instead of erratic choice in the definition of M, then the resulting program is equivalent to $?\underline{\omega}$ because of the divergence avoiding properties of ambiguous choice (see [Mor98]).

There are different forms of countable non-determinism arising from countably indexed erratic choice, and $?\underline{\omega}$ is the least expressive. The differences are formalised as relative definability results in section 5.5. For example, if $A \subseteq_{\mathrm{ne}} \omega$ is not recursively enumerable, then $?\underline{\omega}$ can be defined in terms of $?\underline{A}$ using PCF-like programs, but not vice-versa.

This section concludes with two lemmas, used in section 5.7, that concern reduction, substitution, and blocked substitution for a blocked occurrence of a variable. Lemma 3.4.10(1) shows that if an open term reduces to another, we can deduce that performing the same substitution on both terms also gives a valid reduction. In other words, a reduction of an open term can be instantiated in different ways by substituting for the free variables. Part (2) of the lemma is a partial converse. If an open term with a substitution can reduce, then that reduction either involves the substituted term, in which case the open term must be blocked on the substitution variable, or it does not involve the substituted term, in which case the open term can perform the same reduction.

Lemma 3.4.10 If $\Gamma \vdash L : \sigma$ and $\Gamma, x : \sigma \vdash M_1 : \tau$ then:

1. If $M_1 \twoheadrightarrow M_2$ then $M_1[L/x] \twoheadrightarrow M_2[L/x]$.

2. If $M_1[L/x] \twoheadrightarrow N$, then $M_1 \between x$ or there exists a term M_2 such that $M_1 \twoheadrightarrow M_2$ and $M_2[L/x] = N$.

Proof

1. By induction on the derivation of $M_1 \rightarrowtriangle M_2$, making use of lemma 3.3.4.

2. By induction on the derivation of $\Gamma, x : \sigma \vdash M_1 : \tau$, making use of lemmas 3.2.2 and 3.3.4. $\square$

Lemma 3.4.11(2) shows that, with some restrictions on free variables, a substitution into a blocked term can be written as a blocked substitution followed by the original substitution. Part (3) essentially states that reduction takes place inside reduction contexts presented as blocked terms. Part (4) shows that substituting a fixed-point for a blocked variable gives a term with a known reduction in which the fixed-point is only unwound once. This property is used in the proof of the Scott induction principle.

Lemma 3.4.11 Consider terms:

$$\Gamma, x : \sigma \vdash L : \tau \qquad \Gamma, x : \sigma \vdash M_1 : \sigma \qquad \Gamma \vdash N : \sigma$$

such that $L \lightning x$. Then:

1. $L[x \mapsto M_1][N/x] = L[x \mapsto M_1[N/x]][N/x]$

2. $L[N/x] = L[x \mapsto N][N/x]$

3. If $M_1 \rightarrowtriangle M_2$, then $L[x \mapsto M_1] \rightarrowtriangle L[x \mapsto M_2]$.

4. If σ is a computation type, then $L[\mathsf{fix}x.M_1/x] \rightarrowtriangle_{\mathrm{det}} L[x \mapsto M_1][\mathsf{fix}x.M_1/x]$.

Proof

1. By induction on the derivation of $L \lightning x$.

2. Follows from (1) by taking $M_1 = x$ and observing that $L[x \mapsto x] = L$.

3. By induction on the derivation of $L \lightning x$.

4. Using prior results:

$$\begin{array}{rcll} L[\mathsf{fix}x.M_1/x] & = & L[x \mapsto \mathsf{fix}x.M_1][\mathsf{fix}x.M_1/x] & (3.4.11(2)) \\ & \rightarrowtriangle_{\mathrm{det}} & L[x \mapsto M_1[\mathsf{fix}x.M_1/x]][\mathsf{fix}x.M_1/x] & (3.4.11(3), 3.4.10(1)) \\ & = & L[x \mapsto M_1][\mathsf{fix}x.M_1/x] & (3.4.11(1)) \end{array}$$

$\square$

$$K \Downarrow^{\mathrm{may}} K \quad (K \in Can_0)$$

$$\frac{L \Downarrow^{\mathrm{may}} \mathsf{inj}\, m\, \mathsf{of}\, M \qquad N_m[M/x_m] \Downarrow^{\mathrm{may}} K}{\mathsf{case}\, L\, \mathsf{of}\, \langle x_n.N_n \mid n < \kappa \rangle \Downarrow^{\mathrm{may}} K}$$

$$\frac{M \Downarrow^{\mathrm{may}} \mathsf{tuple}\, \langle N_n \mid n < \kappa \rangle \qquad N_m \Downarrow^{\mathrm{may}} K}{\mathsf{proj}\, m\, \mathsf{of}\, M \Downarrow^{\mathrm{may}} K}$$

$$\frac{L \Downarrow^{\mathrm{may}} \lambda x.M \qquad M[N/x] \Downarrow^{\mathrm{may}} K}{LN \Downarrow^{\mathrm{may}} K}$$

$$\frac{L \Downarrow^{\mathrm{may}} [M] \qquad N[M/x] \Downarrow^{\mathrm{may}} K}{\mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, N \Downarrow^{\mathrm{may}} K}$$

$$\frac{M[\mathsf{fix}\, x.M/x] \Downarrow^{\mathrm{may}} K}{\mathsf{fix}\, x.M \Downarrow^{\mathrm{may}} K}$$

$$?\langle M_n \mid n < \kappa \rangle \Downarrow^{\mathrm{may}} [M_m] \quad (m < \kappa)$$

Figure 3.9: May convergence

3.5 Evaluation Semantics

The evaluation semantics describes the convergence and divergence behaviour of programs without mention of the intermediate states in a sequence of reductions. Evaluation semantics are also known as *natural semantics* or *big-step semantics* (see [Kah87, Gun92, Win93]). Convergence behaviour is given by an inductively-defined *may convergence relation*, and divergence behaviour by a coinductively-defined *may divergence predicate*. In contrast to the reduction relation, the may convergence relation and the may divergence predicate are restricted to programs.

Definition 3.5.1 The ***may convergence*** relation $\Downarrow^{\mathrm{may}} \subseteq \mathscr{L}_0 \times \mathscr{L}_0$ is defined inductively from the rules in figure 3.9. For a program M, we write $M \Downarrow^{\mathrm{may}}$ if and only if there exists a program N such that $M \Downarrow^{\mathrm{may}} N$.

Lemma 3.5.2 establishes the connection between the reduction semantics and the may convergence relation. In particular, may convergence only relates program to canonical programs so $\Downarrow^{\mathrm{may}} \subseteq \mathscr{L}_0 \times Can_0$, and, if $M \Downarrow^{\mathrm{may}} K$, then M and K have the same type.

Lemma 3.5.2 For programs M and N:

1. If $M \Downarrow^{\mathrm{may}} N$, then $M \rightharpoonup^* N$ and N is canonical.

2. If $M \rightharpoonup N$ and $N \Downarrow^{\mathrm{may}} K$, then $M \Downarrow^{\mathrm{may}} K$.

3. $M \Downarrow^{\mathrm{may}} N$ if and only if $M \rightharpoonup^* N$ and N is canonical.

$$\frac{M \Uparrow^{\text{may}}}{E(M) \Uparrow^{\text{may}}}$$

$$\frac{L \Downarrow^{\text{may}} \mathsf{inj}\, m\, \mathsf{of}\, M \qquad N_m[M/x_m] \Uparrow^{\text{may}}}{\mathsf{case}\, L\, \mathsf{of}\, \langle x_n.N_n \mid n < \kappa \rangle \Uparrow^{\text{may}}}$$

$$\frac{M \Downarrow^{\text{may}} \mathsf{tuple}\, \langle N_n \mid n < \kappa \rangle \qquad N_m \Uparrow^{\text{may}}}{\mathsf{proj}\, m\, \mathsf{of}\, M \Uparrow^{\text{may}}}$$

$$\frac{L \Downarrow^{\text{may}} \lambda x.M \qquad M[N/x] \Uparrow^{\text{may}}}{LN \Uparrow^{\text{may}}}$$

$$\frac{L \Downarrow^{\text{may}} [M] \qquad N[M/x] \Uparrow^{\text{may}}}{\mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, N \Uparrow^{\text{may}}}$$

$$\frac{M[\mathsf{fix}\, x.M/x] \Uparrow^{\text{may}}}{\mathsf{fix}\, x.M \Uparrow^{\text{may}}}$$

Figure 3.10: May divergence

Proof

1. By induction on the derivation of $M \Downarrow^{\text{may}} N$.
2. By induction on $M \rightharpoonup N$, using case analysis of $N \Downarrow^{\text{may}} K$ in the inductive step.
3. The forward direction is established in (1). The reverse direction is established by an induction on the length of the sequence of reductions, using (2) for the inductive step. □

In a deterministic programming language the may divergence predicate can be defined as the complement of the may convergence predicate. In $\mathscr{L}$ the relationship between the may divergence and may convergence predicates is weaker: by lemma 3.4.7, every program may converge or may diverge. The may divergence predicate is defined coinductively following [CC92, HM95, Las97, IS98].

Definition 3.5.3 The ***may divergence*** predicate $\Uparrow^{\text{may}} \subseteq \mathscr{L}_0$ is defined coinductively from the rules in figure 3.10. The may convergence judgements are side conditions rather than premises.

In section 3.6 it is shown that programs of value type cannot diverge, and consequently we can replace the rule:

$$\frac{M \Uparrow^{\text{may}}}{E(M) \Uparrow^{\text{may}}}$$

with:

$$\frac{L \Uparrow^{\text{may}}}{\mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, N \Uparrow^{\text{may}}}$$

However, the original formulation is more convenient for the statement and proof of the compatibility theorems in chapter 5.

The may divergence predicate coincides with the notion of divergence arising from the reduction semantics.

Lemma 3.5.4 If M is a program, then $M \Uparrow^{\mathrm{may}}$ if and only if $M \rightarrowtriangle^{\omega}$.

Proof We prove the forward direction first. By coinduction it suffices to prove that $M \Uparrow^{\mathrm{may}}$ implies there exists a program N such that $M \rightarrowtriangle^{+} N$ and $N \Uparrow^{\mathrm{may}}$. This is proven by induction on the type assignment derivation of M, making use of lemma 3.5.2. The reverse direction is also proven by coinduction. We need to show that the conclusion of one of the rules in figure 3.10 matches M and that the premise of that rule (not the may convergence side conditions) is contained in $\rightarrowtriangle^{\omega}$. This is proven by case analysis on the derivation of the first reduction. The base cases are straightforward. For the remaining case, suppose that $E(M) \rightarrowtriangle E(N) \rightarrowtriangle^{\omega}$ and $M \rightarrowtriangle N$. If $M \rightarrowtriangle^{\omega}$, then the may divergence rule for $E(M)$ applies. Otherwise, there exists a canonical program K such that $M \rightarrowtriangle^{+} K$ and $E(M) \rightarrowtriangle^{+} E(K) \rightarrowtriangle^{\omega}$. By lemma 3.5.2, $M \Downarrow^{\mathrm{may}} K$, and it can be verified that one of the may divergence rules applies for each of the four forms of reduction constructors. □

The *must convergence predicate* is the complement of the may divergence predicate, so a program must converge if it cannot diverge. By lemma 2.3.5, the complement of the coinductively-defined may divergence predicate is the least fixed-point of some monotone function on programs, and this function can be defined in terms of the monotone function induced by the may divergence rules. In fact, the must convergence predicate is also inductively-defined from a collection of rules.

Definition 3.5.5 The ***must convergence*** predicate $\Downarrow^{\mathrm{must}} \subseteq \mathscr{L}_0$ is defined inductively from the rules in figure 3.11.

As with the may divergence rules, the must convergence rules do not take advantage of the fact that programs of value type cannot diverge.

Example 3.4.9 demonstrates that some programs may converge to infinitely many canonical programs. Consequently, some instances of the must convergence rule schema for programs of the form $\mathsf{let}\, x \Leftarrow M \,\mathsf{in}\, N$ have an infinite collection of premises, and so the derivation trees for must convergence judgements may have nodes with an infinite set of successors. The derivation trees for inductive definitions are always well-founded and so have a rank. In this case, the rank may be greater than ω because of infinite branching. The rank of derivation trees for must convergence judgements is investigated in section 3.8.

Finally, we consider derived rules for binary erratic choice. The binary erratic choice of M and N may converge to a canonical program if either M or N can:

$$\frac{M \Downarrow^{\mathrm{may}} K}{M \cup N \Downarrow^{\mathrm{may}} K} \qquad \frac{N \Downarrow^{\mathrm{may}} K}{M \cup N \Downarrow^{\mathrm{may}} K}$$

$$K \Downarrow^{\text{must}} \qquad (K \in Can_0)$$

$$\frac{L \Downarrow^{\text{must}} \qquad \{N_m[M/x_m] \Downarrow^{\text{must}} \mid L \Downarrow^{\text{may}} \mathsf{inj}\, m\, \mathsf{of}\, M\}}{\mathsf{case}\, L\, \mathsf{of}\, \langle x_n.N_n \mid n < \kappa\rangle \Downarrow^{\text{must}}}$$

$$\frac{M \Downarrow^{\text{must}} \qquad \{N_m \Downarrow^{\text{must}} \mid M \Downarrow^{\text{may}} \mathsf{tuple}\,\langle N_n \mid n < \kappa\rangle\}}{\mathsf{proj}\, m\, \mathsf{of}\, M \Downarrow^{\text{must}}}$$

$$\frac{L \Downarrow^{\text{must}} \qquad \{M[N/x] \Downarrow^{\text{must}} \mid L \Downarrow^{\text{may}} \lambda x.M\}}{LN \Downarrow^{\text{must}}}$$

$$\frac{L \Downarrow^{\text{must}} \qquad \{N[M/x] \Downarrow^{\text{must}} \mid L \Downarrow^{\text{may}} [M]\}}{\mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, N \Downarrow^{\text{must}}}$$

$$\frac{M[\mathsf{fix}\, x.M/x] \Downarrow^{\text{must}}}{\mathsf{fix}\, x.M \Downarrow^{\text{must}}}$$

$$?\langle M_n \mid n < \kappa\rangle \Downarrow^{\text{must}}$$

Figure 3.11: Must convergence

The binary erratic choice of M and N may diverge if either M or N can:

$$\frac{M \Uparrow^{\text{may}}}{M \cup N \Uparrow^{\text{may}}} \qquad \frac{N \Uparrow^{\text{may}}}{M \cup N \Uparrow^{\text{may}}} \qquad \frac{M \Downarrow^{\text{must}} \qquad N \Downarrow^{\text{must}}}{M \cup N \Downarrow^{\text{must}}}$$

The may divergence (equivalently, must convergence) behaviour of binary erratic choice differentiates it from binary ambiguous choice (see [Las98b, Mor98] and section 2.6), because the binary ambiguous choice of M and N may diverge only if both M and N may diverge.

3.6 Normalisation

Programs of computation type can exhibit non-termination or non-determinism. In this section we prove that these behaviours are restricted to computation types because a program of value type always converges to a unique canonical program.

There are a number of techniques for proving normalisation of typed functional programming languages. For example, Girard [GLT89] describes both an elementary proof of weak normalisation and his extension of Tait's method to System F. Gordon [Gor94] proves a normalisation result for a variant of the computational λ-calculus with restricted recursive types via a translation to a strongly normalising λ-calculus.

The normalisation result for $\mathscr{L}$ requires some care because of infinitary terms. Reduction cannot be permitted inside constructors unless they form a reduction context, even for value types, because it may lead to infinite sequences of reductions for canonical programs. For example, each instance of $(\lambda x.x)\star$ in the following program could reduce to $\star$:

$$\mathsf{tuple}\,\langle(\lambda x.x)\star \mid n < \omega\rangle$$

The proof of normalisation given here is based upon Tait's method. We first define a type-indexed family of *reducibility candidates*.

Definition 3.6.1 For each type σ, define a set of programs $Red(\sigma)$ by $M \in Red(\sigma)$ if and only if M is a program of type σ and one of the following holds:

- $\sigma = \mathsf{sum}\,\langle \sigma_n \mid n < \kappa \rangle \wedge \exists m, N.\,(M \Downarrow^{\mathrm{may}} \mathsf{inj}\, m\, \mathsf{of}\, N) \wedge N \in Red(\sigma_m)$
- $\sigma = \mathsf{prod}\,\langle \sigma_n \mid n < \kappa \rangle \wedge \exists \langle N_n \mid n < \kappa \rangle.\, M \Downarrow^{\mathrm{may}} \mathsf{tuple}\,\langle N_n \mid n < \kappa \rangle \wedge \forall n < \kappa.\, N_n \in Red(\sigma_n)$
- $\sigma = \sigma_1 \to \sigma_2 \wedge \forall N \in Red(\sigma_1).\, \exists L.\,(M \Downarrow^{\mathrm{may}} \lambda x.\, L) \wedge L[N/x] \in Red(\sigma_2)$
- σ is a computation type.

Lemma 3.6.2 If M and N are programs such that $M \twoheadrightarrow N$ and $N \in Red(\sigma)$, then $M \in Red(\sigma)$.

Proof Immediate for programs of computation type. For programs of value type, apply lemma 3.5.2(2). □

Tait's method involves showing that the reducibility candidates contain all programs, and so every program of value type must converge to a canonical program.

Proposition 3.6.3 If M is a program of a value type, then $M \Downarrow^{\mathrm{must}}$ and there exists a unique canonical program K such that $M \Downarrow^{\mathrm{may}} K$.

Proof We show that every program of type σ is a member of $Red(\sigma)$, and deduce that there exists a canonical program K such that $M \Downarrow^{\mathrm{may}} K$. The uniqueness of K and the must convergence property follow because reduction is deterministic at value types (see lemma 3.4.4). To show that every program of type σ is a member of $Red(\sigma)$, we prove a more general result. For an environment $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ and a term $\Gamma \vdash M : \tau$ such that, whenever $N_1 \in Red(\sigma_1), \ldots, N_n \in Red(\sigma_n)$, we have $M[\vec{N}/\vec{x}] \in Red(\tau)$. This is a straightforward induction on the derivation of $\Gamma \vdash M : \tau$, making use of lemma 3.6.2 when M has value type and is not canonical.

□

3.7 Fragments of $\mathscr{L}$

The non-deterministic λ-calculus $\mathscr{L}$ is more expressive than most of the non-deterministic λ-calculi in the literature because of countably infinite coproducts, products, and indexed erratic choice (the notable missing features are call-by-need parameter-passing and parallel non-determinism such as ambiguous choice). If we were only to consider semantic relations such as contextual equivalence and bisimilarity for $\mathscr{L}$, then the results may be inapplicable to less expressive calculi because the semantic relations may be too fine for higher-order terms. For example, in chapter 5 it is shown that binary erratic choice cannot distinguish some programs that can be distinguished using $?\underline{\omega}$ (cf. programs that cannot be distinguished by sequential programs but can be distinguished using parallel-or, see [Gun92]). Fortunately, the definition of the syntax and operational semantics of the coproducts, products, and indexed erratic choice permits a straightforward treatment of fragments of $\mathscr{L}$, where each fragment is a non-deterministic λ-calculus in its own right. The compatibility and Scott induction proofs in chapter 5 are parameterised by a fragment of $\mathscr{L}$. In this section, we define the collection of fragments via a number of closure conditions upon sets of terms of $\mathscr{L}$.

The expressiveness of $\mathscr{L}$ can also be exploited for a study of relative definability of different forms of indexed erratic choice in an operational setting (see section 5.5). Previous treatments of relative definability rely upon a denotational model as a source of non-definable elements. For example, the theory of Turing degrees [Rog67, Cut80, Odi89] is concerned with relative definability of the characteristic functions of sets of natural numbers in the space of partial functions $\bigcup\{\omega^n \rightharpoonup \omega \mid n \in \omega\}$. The definable elements of the space are the partial recursive functions. As another example, Sazonov's degrees of parallelism [Saz75, Lic96, Buc97] is based in a space of directed-complete partial orders. The definable elements are the (sequential) denotations of PCF terms.

The fragments of $\mathscr{L}$ that we consider are not arbitrary sets of terms, but must satisfy certain closure conditions. The definition of a fragment must satisfy at least three criteria. A fragment should not constrain the existing operational semantics for $\mathscr{L}$ (see lemma 3.7.2), it should be possible to prove compatibility for variants of applicative similarity and bisimilarity, and each fragment should be at least as expressive as PCF. The first and second criteria can be satisfied by insisting that every fragment is closed under substitution and taking subterms. The third criterion is satisfied by closing fragments under finite constructors and the abbreviated arithmetic operations.

Definition 3.7.1 Consider a set of terms $\mathcal{E} \subseteq \mathscr{L}$. The notation $\Gamma \vdash M \in \mathcal{E} : \sigma$ means that $\Gamma \vdash M : \sigma$ and $M \in \mathcal{E}$. The ***fragment*** containing $\mathcal{E}$, written $\mathscr{L}(\mathcal{E}) \subseteq \mathscr{L}$, is the least set closed under the rules of figures 3.12 and 3.13, and also closed under subterms (including subterms of infinitary terms). The set of programs in $\mathscr{L}(\mathcal{E})$ is defined by $\mathscr{L}_0(\mathcal{E}) \stackrel{\text{def}}{=} \mathscr{L}(\mathcal{E}) \cap \mathscr{L}_0$. We write $\mathscr{L}(M_1, \ldots, M_n)$ for $\mathscr{L}(\{M_1, \ldots, M_n\})$.

The final condition in figure 3.13 forces fragments to be closed under substitutions that replace a subset of variable occurrences. This ensures that fragments are closed under blocked substitution.

Fragments may be smaller than $\mathscr{L}$ because there are no closure rules for indexed erratic choice, and the closure rules for terms involving coproducts and products are restricted to arithmetic operators or finite κ by the side conditions $\kappa < \omega$.

The substitution and subterm closure conditions ensure that the operational semantics for $\mathscr{L}$ is well-behaved within each fragment.

Lemma 3.7.2 For $\mathcal{E} \subseteq \mathscr{L}$ and $M \in \mathscr{L}(\mathcal{E})$:

1. If $M \rightarrowtriangle N$ then $N \in \mathscr{L}(\mathcal{E})$.

2. If $M \Downarrow^{\text{may}} K$ then $K \in \mathscr{L}_0(\mathcal{E})$.

$$\Gamma \vdash x \in \mathscr{L}(\mathcal{E}) : \sigma \quad (\Gamma(x) = \sigma)$$

$$\frac{\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \sigma_m}{\Gamma \vdash \mathsf{inj}\, m\, \mathsf{of}\, M \in \mathscr{L}(\mathcal{E}) : \mathsf{sum}\, \langle \sigma_n \mid n < \kappa \rangle} \quad (\kappa < \omega)$$

$$\frac{\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{sum}\, \langle \sigma_n \mid n < \kappa \rangle \qquad \{\Gamma, x_n : \sigma_n \vdash N_n \in \mathscr{L}(\mathcal{E}) : \tau \mid n < \kappa\}}{\Gamma \vdash \mathsf{case}\, M\, \mathsf{of}\, \langle x_n . N_n \mid n < \kappa \rangle \in \mathscr{L}(\mathcal{E}) : \tau} \quad (\kappa < \omega)$$

$$\frac{\{\Gamma \vdash M_n \in \mathscr{L}(\mathcal{E}) : \sigma_n \mid n < \kappa\}}{\Gamma \vdash \mathsf{tuple}\, \langle M_n \mid n < \kappa \rangle \in \mathscr{L}(\mathcal{E}) : \mathsf{prod}\, \langle \sigma_n \mid n < \kappa \rangle} \quad (\kappa < \omega)$$

$$\frac{\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{prod}\, \langle \sigma_n \mid n < \kappa \rangle}{\Gamma \vdash \mathsf{proj}\, m\, \mathsf{of}\, M \in \mathscr{L}(\mathcal{E}) : \sigma_m} \quad (\kappa < \omega)$$

$$\frac{\Gamma, x : \sigma \vdash M \in \mathscr{L}(\mathcal{E}) : \tau}{\Gamma \vdash \lambda x{:}\sigma . M \in \mathscr{L}(\mathcal{E}) : \sigma \to \tau}$$

$$\frac{\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \sigma \to \tau \qquad \Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \sigma}{\Gamma \vdash M N \in \mathscr{L}(\mathcal{E}) : \tau}$$

$$\frac{\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \sigma}{\Gamma \vdash [M] \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\sigma) \qquad \Gamma, x : \sigma \vdash N \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\tau)}{\Gamma \vdash \mathsf{let}\, x{:}\sigma \Leftarrow M\, \mathsf{in}\, N \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\tau)}$$

$$\frac{\Gamma, x : \mathsf{P}_\perp(\sigma) \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \mathsf{fix}\, x{:}\mathsf{P}_\perp(\sigma) . M \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\sigma)}$$

Figure 3.12: Fragment closure (part 1)

$$\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \sigma \quad (\Gamma \vdash M \in \mathcal{E} : \sigma)$$

$$\Gamma \vdash \underline{n} \in \mathscr{L}(\mathcal{E}) : \mathsf{nat} \quad (n \in \omega)$$

$$\frac{\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{nat} \qquad \Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \mathsf{nat}}{\Gamma \vdash \mathsf{plus}(M,N) \in \mathscr{L}(\mathcal{E}) : \mathsf{nat}}$$

$$\frac{\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{nat} \qquad \Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \mathsf{nat}}{\Gamma \vdash \mathsf{minus}(M,N) \in \mathscr{L}(\mathcal{E}) : \mathsf{nat}}$$

$$\frac{\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{nat} \qquad \Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \mathsf{nat}}{\Gamma \vdash \mathsf{eq}(M,N) \in \mathscr{L}(\mathcal{E}) : \mathsf{bool}}$$

$$\frac{\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{nat} \qquad \Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \mathsf{nat}}{\Gamma \vdash \mathsf{lt}(M,N) \in \mathscr{L}(\mathcal{E}) : \mathsf{bool}}$$

$$\frac{\begin{array}{c}\Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \tau \\ \Gamma \vdash M[\vec{y}/\vec{x}] \in \mathscr{L}(\mathcal{E}) : \tau \\ \{\Gamma \vdash N_i \in \mathscr{L}(\mathcal{E}) : \sigma_i \mid 1 \leq i \leq n\}\end{array}}{\Gamma \vdash M[\vec{N}/\vec{x}] \in \mathscr{L}(\mathcal{E}) : \tau}$$

Figure 3.13: Fragment closure (part 2)

Proof By induction on the derivation of $M \twoheadrightarrow N$, and then apply lemma 3.5.2. □

The arithmetic closure conditions provide enough terms for a translation of PCF (call-by-name or call-by-value) into $\mathscr{L}(\emptyset)$. The call-by-value translation can be used to show that for every partial recursive function $f : \omega^n \rightharpoonup \omega$ there is a program:

$$\vdash M \in \mathscr{L}(\emptyset) : \mathsf{nat} \to \ldots \to \mathsf{nat} \to \mathsf{P}_{\perp}(\mathsf{nat})$$

such that, for all $m, m_1, \ldots, m_n \in \omega$, $f(m_1, \ldots, m_n)$ is defined and equal to m if and only if there exists a program N such that $M\,\underline{m_1}\,\ldots\,\underline{m_n} \Downarrow^{\mathrm{may}} [N]$ and $N \Downarrow^{\mathrm{may}} \underline{m}$.

The set of terms used to define a fragment must be chosen carefully because of the subterm closure condition. For example, any fragment that contains:

$$\mathsf{proj}\,0\,\mathsf{of}\,\mathsf{tuple}\langle\star, ?\langle\mathsf{false}, \mathsf{true}\rangle\rangle$$

must also contain $?\langle\mathsf{false}, \mathsf{true}\rangle$, even though that program will be discarded in every reduction sequence. For this reason, when we consider fragments of $\mathscr{L}$ that are strictly less expressive than those containing binary erratic choice we use, for example:

$$\mathcal{E} = \{\mathsf{let}\,x \Leftarrow ?\langle\Omega, [\star]\rangle\,\mathsf{in}\,x\}$$

rather than $\mathcal{E} = \{\Omega \cup [\star]\}$ because the latter uses binary erratic choice which forces $?\langle\mathsf{false}, \mathsf{true}\rangle$ into the fragment.

$$K \Downarrow^{\text{must}} 0 \quad (K \in Can_0)$$

$$\frac{L \Downarrow^{\text{must}} A \qquad L \Downarrow^{\text{may}} \mathsf{inj}\, m\, \mathsf{of}\, M \qquad N_m[M/x_m] \Downarrow^{\text{must}} B}{\mathsf{case}\, L\, \mathsf{of}\, \langle x_n.N_n \mid n < \kappa\rangle \Downarrow^{\text{must}} (A+1) \cup (B+1)}$$

$$\frac{M \Downarrow^{\text{must}} A \qquad M \Downarrow^{\text{may}} \mathsf{tuple}\, \langle N_n \mid n < \kappa\rangle \qquad N_m \Downarrow^{\text{must}} B}{\mathsf{proj}\, m\, \mathsf{of}\, M \Downarrow^{\text{must}} (A+1) \cup (B+1)}$$

$$\frac{L \Downarrow^{\text{must}} A \qquad L \Downarrow^{\text{may}} \lambda x.M \qquad M[N/x] \Downarrow^{\text{must}} B}{LN \Downarrow^{\text{must}} (A+1) \cup (B+1)}$$

$$\frac{L \Downarrow^{\text{must}} A \qquad \{N[M/x] \Downarrow^{\text{must}} B_M \mid L \Downarrow^{\text{may}} [M]\}}{\mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, N \Downarrow^{\text{must}} (A+1) \cup \bigcup\{B_M + 1 \mid L \Downarrow^{\text{may}} [M]\}}$$

$$\frac{M[\mathsf{fix}\, x.M/x] \Downarrow^{\text{must}} A}{\mathsf{fix}\, x.M \Downarrow^{\text{must}} (A+1)}$$

$$?\langle M_n \mid n < \kappa\rangle \Downarrow^{\text{must}} 0$$

Figure 3.14: Must convergence rank

3.8 Rank of Must Convergence

In this section we investigate the rank of derivation trees for must convergence judgements. The must convergence predicate is defined by induction, so the derivation trees are well-founded and the rank always exists. The rank of the derivation trees can be used to classify programs and provides a measure of the proof-theoretic strength of arguments involving induction on must convergence judgements, such as the compatibility theorem in chapter 5. Lassen [Las98b] gives unwinding and syntactic continuity results for a non-deterministic λ-calculus with an operator equivalent to $?\underline{\omega}$. The results make use of transfinite unwindings of fixed-point terms, and the operational semantics of such terms is based upon the rank of the derivation trees of must convergence judgements.

The rank of a well-founded tree is defined in definition 2.1.16. However, it is convenient to have a direct definition of the rank of a derivation tree for a must convergence judgement.

Definition 3.8.1 Let M be a program such that $M \Downarrow^{\text{must}}$. The ***must convergence rank*** of M is an ordinal defined inductively from the rules in figure 3.14, where $M \Downarrow^{\text{must}} A$ means that M has must convergence rank A.

The must convergence rank of a program is well-defined because each program has at most one must convergence derivation tree.

Proposition 3.8.2 proves that the supremum of the must convergence ranks of programs in $\mathscr{L}$ is ω_1, the least uncountable ordinal. In addition, the supremum is shown to be ω when programs are only drawn from deterministic or finitely non-deterministic fragments.

Proposition 3.8.2

1. If $\mathcal{E} \subseteq \mathscr{L}$ is such that $\mathscr{L}(\mathcal{E})$ does not contain any occurrences of infinite indexed erratic choice term constructors, then:

$$\bigcup\{A \mid \exists M \in \mathscr{L}_0(\mathcal{E}). M \Downarrow^{\text{must}} A\} = \omega$$

2. If $\mathcal{E} \subseteq \mathscr{L}$ is such that $\mathscr{L}(\mathcal{E}) = \mathscr{L}$ then:

$$\bigcup\{A \mid \exists M \in \mathscr{L}_0(\mathcal{E}). M \Downarrow^{\text{must}} A\} = \omega_1$$

Proof

1. With the exception of the rule for the sequencing term constructor, all instances of the must convergence rule schema have a finite number of premises, no matter which fragment of the language is considered. If $\mathscr{L}(\mathcal{E})$ does not contain any infinite erratic choice term constructors, then lemma 3.4.8(2) applies to every must convergent program $M \in \mathscr{L}_0(\mathcal{E})$ and therefore the restriction of the must convergence rule schema for the sequencing term constructor to $\mathscr{L}_0(\mathcal{E})$ must also be finitely-branching. By induction on the proof of $M \Downarrow^{\text{must}} A$ it can be shown that $A < \omega$. To see that the supremum is ω, consider the sequence defined by:

$$\begin{aligned} M_0 &\stackrel{\text{def}}{=} [\star] \\ M_{n+1} &\stackrel{\text{def}}{=} \mathsf{let}\, x \Leftarrow [M_n]\, \mathsf{in}\, x \qquad (n \in \omega) \end{aligned}$$

Then, for all $n \in \omega$, $M_n \Downarrow^{\text{must}} n$.

2. By lemma 3.4.8(1), the instances of the must convergence rule schema for the sequencing term constructor may have at most a countably infinite family of premises. It can be shown by induction on the proof of $M \Downarrow^{\text{must}} A$ that $A < \omega_1$. For the other direction, we would like to define a transfinite sequence of must convergent programs by induction:

$$\begin{aligned} M_0 &\stackrel{\text{def}}{=} [\star] \\ M_A &\stackrel{\text{def}}{=} \mathsf{let}\, x \Leftarrow ?\langle M_B \mid B < A\rangle\, \mathsf{in}\, x \qquad (0 < A < \omega_1) \end{aligned}$$

However, $\mathsf{let}\, x \Leftarrow ?\langle M_B \mid B < A\rangle\, \mathsf{in}\, x$ is not a term when $A > \omega$, and so we have to use the fact that A is countable to obtain an ω-sequence of programs that is a reordering of $?\langle M_B \mid B < A\rangle$. It is then straightforward to prove by induction that, for all $A < \omega_1$, $M_A \Downarrow^{\text{must}} A$. □

There is an analogue of proposition 3.8.2 for countably non-deterministic fragments such as $\mathscr{L}(?\underline{\omega})$. For each fragment, the supremum of the must convergence ranks is the least non-recursive ordinal ω_1^{CK} (see section 2.5). Proposition 3.8.3 is based upon a similar result given by Apt and Plotkin [AP86] for an imperative programming language with an operator equivalent to $?\underline{\omega}$. Apt and Plotkin prove that the reduction sequences of a non-divergent program form a well-founded tree with rank[2] ω_1^{CK}.

[2] Warning: Apt and Plotkin [AP86] use height of a well-founded tree for the ordinal called the rank of a well-founded tree here.

Proposition 3.8.3 Consider $\mathcal{E} \subseteq \mathscr{L}$ such that every term in $\mathcal{E}$ is a program of type $\mathsf{P}_\perp(\mathsf{nat})$ and has the form $?\langle M_n \mid n < \kappa\rangle$, where $\{m \in \omega \mid \exists n < \kappa. M_n \Downarrow^{\text{may}} \underline{m}\}$ is a recursive set. If at least one of these subsets of ω is infinite, then:

$$\bigcup\{A \mid \exists M \in \mathscr{L}_0(\mathcal{E}). M \Downarrow^{\text{must}} A\} = \omega_1^{\text{CK}}$$

Proof To prove that the supremum is less than or equal to ω_1^{CK} we show that, for every program $M \in \mathscr{L}(\mathcal{E})$ such that $M \Downarrow^{\text{must}} A$, we have $A < \omega_1^{\text{CK}}$, i.e. A is a recursive ordinal. By proposition 2.5.10, it suffices to show that each derivation tree is isomorphic to a recursive tree. It is not necessarily possible to encode all terms in $\mathscr{L}(\mathcal{E})$ as natural numbers because there could be infinitely many terms in $\mathcal{E}$ that determine the same recursive set. However, a fixed must convergent program $M \in \mathscr{L}(\mathcal{E})$ uses only a finite subset $\mathcal{E}' \subseteq \mathcal{E}$, so $M \in \mathscr{L}(\mathcal{E}')$. By an argument similar to that of lemma 3.7.2 it can be shown that all terms present in a derivation of must convergence for M (including may convergence side conditions) can be encoded as natural numbers. The terms in $\mathcal{E}'$ are treated as constants rather than infinite terms. Building upon the encoding of terms, derivation trees for may convergence judgements can also be encoded as natural numbers so that they form a recursive set—it is decidable whether a natural number represents a valid may convergence derivation. This requires the hypothesis that all of the terms in the finite set $\mathcal{E}'$ determine recursive sets. Finally, branches in the must convergence derivation tree for M can be encoded as sequences of natural numbers so that it is decidable whether a sequence represents a valid branch. It is necessary to encode the may convergence side conditions to ensure decidability. Therefore, a must convergence derivation is isomorphic to a recursive tree, so its rank is a recursive ordinal, and the supremum of all of the must convergence ranks for programs in $\mathscr{L}_0(\mathcal{E})$ is less than or equal to ω_1^{CK}.

For the other direction, we may assume $?\underline{\omega} \in \mathcal{E}$ without loss of generality, because there is always a program with the same convergence and divergence behaviour. To see this, suppose that $\vdash ?\langle M_n \mid n < \kappa\rangle \in \mathcal{E} : \mathsf{P}_\perp(\mathsf{nat})$ determines an infinite subset of ω. Then there is a program with the same convergence and divergence behaviour as $?\langle \mathsf{false}, \mathsf{true}\rangle$, defined by choosing two numbers and testing whether the first is less than the second:

$$\mathsf{let}\, x \Leftarrow ?\langle M_n \mid n < \kappa\rangle \,\mathsf{in}\, \mathsf{let}\, y \Leftarrow ?\langle M_n \mid n < \kappa\rangle \,\mathsf{in}\, \mathsf{if}\, \mathsf{lt}\,(x,y)\, \mathsf{then}\, [\mathsf{false}]\, \mathsf{else}\, [\mathsf{true}]$$

Hence, the binary erratic choice term constructor can be defined in terms of $?\langle M_n \mid n < \kappa\rangle$. Now define the program N of type $\mathsf{nat} \to \mathsf{P}_\perp(\mathsf{nat})$ by:

$$N \stackrel{\text{def}}{=} \mathsf{ffix}\, f. \lambda x. \,\mathsf{if}\, \mathsf{eq}\,(x, \underline{0})\, \mathsf{then}\, [\underline{0}]\, \mathsf{else}\, ([x] \cup f\,(\mathsf{minus}\,(x, \underline{1})))$$

Then $?\underline{\omega}$ has the same convergence and divergence behaviour as the program:

$$\mathsf{let}\, y \Leftarrow ?\langle M_n \mid n < \kappa\rangle \,\mathsf{in}\, N\, y$$

This program chooses a natural number y from an infinite set, and then $N\,y$ chooses a number between 0 and y. It is an acceptable substitute for $?\underline{\omega}$ in the argument below because its must convergence rank is always greater than or equal to ω, as opposed to $?\underline{\omega} \Downarrow^{\text{must}} 0$, and we intend to show that for every recursive ordinal A there is a program with a must convergence rank greater than or equal to A.

Consider a recursive ordinal A. Suppose that a set of natural numbers $B \subseteq \omega$, a well-order $\preceq \subseteq B \times B$, and a partial recursive function $f : \omega \times \omega \rightharpoonup \omega$ specify a recursive well-order that is order-isomorphic to A, and $g : B \to A$ is a component of that isomorphism. We assume a program $M_A \in \mathscr{L}_0(\emptyset)$ of type $\mathsf{nat} \to \mathsf{nat} \to \mathsf{P}_\perp(\mathsf{bool})$ that implements f. Define a program $\mathsf{slow} \in \mathscr{L}_0(?\underline{\omega})$ with type $(\mathsf{nat} \to \mathsf{nat} \to \mathsf{P}_\perp(\mathsf{bool})) \to \mathsf{nat} \to \mathsf{P}_\perp(\mathsf{unit})$ by:

$$\begin{aligned}\mathsf{slow} \stackrel{\text{def}}{=} \lambda h.\, \lambda x.\, &\mathsf{let}\, y \Leftarrow ?\underline{\omega}\, \mathsf{in} \\ &\qquad \mathsf{let}\, z \Leftarrow h\,y\,x\, \mathsf{in}\, \mathsf{if}\, (\mathsf{and}\, (z, \mathsf{not}\, (\mathsf{eq}\, (x,y))))\, \mathsf{then}\, (\mathsf{slow}\, h\,y)\, \mathsf{else}\, [\star]\end{aligned}$$

The program slow takes as arguments a partial order h on the natural numbers and a natural number x, and then chooses a new number y. If y is strictly less than x, with respect to the partial order h, it recursively calls itself with h and y as arguments. Otherwise, it converges to $[\star]$. Thus all sequences of natural numbers starting with x that are strictly descending with respect to h can occur during the evaluation of $\mathsf{slow}\, h\,x$. Now, if $n \in \omega \setminus B$, then $\mathsf{slow}\, M_A\, \underline{n} \Downarrow^{\mathrm{must}}$ because $M_A\, \underline{m}\, \underline{n} \Downarrow^{\mathrm{may}} \mathsf{false}$, for all $m \in \omega$. Then, by well-founded induction on $n \in B$ with respect to $\preceq$, it can be shown that there exists an ordinal $A' \geq g(n)$ such that $\mathsf{slow}\, M_A\, \underline{n} \Downarrow^{\mathrm{must}} A'$. Therefore, for some ordinal $A' \geq A$ we have that:

$$\mathsf{let}\, x \Leftarrow ?\underline{\omega}\, \mathsf{in}\, \mathsf{slow}\, M_A\, x \Downarrow^{\mathrm{must}} A'$$

Consequently, the supremum of the must convergence ranks of programs in $\mathscr{L}_0(?\underline{\omega})$ is greater than or equal to ω_1^{CK}. $\square$

Infinite coproducts that are not arithmetic operators are excluded from $\mathcal{E}$ for proposition 3.8.3 because it would be possible to use infinitary case statements to construct programs that determine non-recursive sets. For example, if $\{m_n \in \omega \mid n < \omega\}$ is a non-recursive set, then the program:

$$\mathsf{let}\, x \Leftarrow ?\underline{\omega}\, \mathsf{in}\, \mathsf{case}\, x\, \mathsf{of}\, \langle y_n.[\underline{m_n}] \mid n < \omega \rangle$$

has the same convergence and divergence behaviour as $?\langle \underline{m_n} \mid n < \omega \rangle$.

Chapter 4

Typed Transition Systems

We introduce and investigate a class of LTSWDs (see definition 2.2.8), called typed transition systems, that are suitable for studying non-deterministic λ-calculi such as $\mathscr{L}$ and its fragments. Each state of a typed transition system has a unique type from the type system for $\mathscr{L}$. Typed transition systems are a simply-typed variant of Ong's (quasi-)non-deterministic applicative transition systems, which in turn draw upon analyses of process calculi via their LTS structure and λ-calculi via applicative structures, pre-frames, (quasi-)applicative transition systems, and LTSs. The variants of similarity and bisimilarity defined for LTSWDs apply to typed transition systems, and give rise to variants of applicative similarity and applicative bisimilarity on the typed transition systems derived from non-deterministic λ-calculi.

In section 4.1 we define the class of typed transition systems, discuss basic properties of typed transition systems, and review other classes of applicative structures that appear in the literature. In section 4.2 we recall the definitions of the lower, upper, convex, and refinement variants of similarity and bisimilarity for LTSWDs, and consider identifications between the relations that hold in degenerate cases, e.g., for states of a type with P-order 0 or 1. Sections 4.3 and 4.4 provide a case study of a typed transition system $\mathscr{S}$ that extends $\in$-TSWDs with coproduct, product, and function types. The states of $\mathscr{S}$ can be defined by induction on their type because the type system for $\mathscr{L}$ does not include coinductive or recursive types. For example, the states of function type $\sigma \to \tau$ in $\mathscr{S}$ are set-theoretic functions from the states of type σ to the states of type τ. Simple examples that differentiate the variants of similarity and bisimilarity are constructed in $\mathscr{S}$. In section 4.5 we investigate maps between typed transition systems, and show that there is a map from every typed transition system satisfying certain conditions to $\mathscr{S}$, so $\mathscr{S}$ is a weak terminal in a non-trivial subcategory of the category of typed transition systems. In chapter 5, we show that many of the typed transition systems derived from fragments of $\mathscr{L}$ satisfy these conditions.

4.1 Typed Transition Systems

Typed transition systems abstract the computational behaviour of programs of $\mathscr{L}$, and serve the same role for $\mathscr{L}$ and its fragments as LTSs do for CCS. Intuitively, the abstraction combines the evaluation semantics with a decomposition of the canonical programs resulting from evaluation. Individual reduction steps are not part of the structure of a typed transition system.

A typed transition system can be presented as an LTSWD where each state has a unique type. The transitions from a state are restricted by its type because a transition indicates convergent behaviour and information about the result. For example, a state of type $\mathsf{sum}\,\langle\sigma_n \mid n < \kappa\rangle$ must have exactly one transition. The label on the transition identifies the component of the coproduct in which the result lies, and so must be a natural number $m < \kappa$. The target state of the transition must have type σ_m. On the other hand, a state of type $\mathsf{prod}\,\langle\sigma_n \mid n < \kappa\rangle$ must have exactly one transition labelled with n, for each $n < \kappa$. The target state of a transition labelled with n must have type σ_n. A state of function type $\sigma \to \tau$ must have exactly one transition labelled with $@s$ for each state s of type σ, and the target state of the transition must have type τ. States of type $\mathsf{P}_\perp(\sigma)$ may have zero or more transitions, but they must be labelled with $\Downarrow$ and the target states must have type σ. A state may only diverge if it has a computation type, and every state with a computation type must be able to diverge if it has no transitions. This reflects the fact that every program of computation type must have at least one convergent or divergent behaviour.

Definition 4.1.1 A ***typed transition system*** (TTS) is an LTSWD $\mathscr{T} = \langle S, A, \Uparrow^{\mathrm{may}}, \to\rangle$ such that:

1. Every state has a unique type. The set of states with type σ is denoted $\mathscr{T}(\sigma)$. We also write $\mathscr{T}$ for S.
2. If $s \Uparrow^{\mathrm{may}}$, then s has a computation type.
3. The set of labels A satisfies:

$$A = \omega \cup \{@s \mid s \in \mathscr{T}\} \cup \{\Downarrow\}$$

4. If $s \xrightarrow{a} t$, then:
   (a) $s \in \mathscr{T}(\mathsf{sum}\,\langle\sigma_n \mid n < \kappa\rangle) \Longrightarrow (a \in \omega) \wedge (a < \kappa) \wedge (t \in \mathscr{T}(\sigma_a))$
   (b) $s \in \mathscr{T}(\mathsf{prod}\,\langle\sigma_n \mid n < \kappa\rangle) \Longrightarrow (a \in \omega) \wedge (a < \kappa) \wedge (t \in \mathscr{T}(\sigma_a))$
   (c) $s \in \mathscr{T}(\sigma \to \tau) \Longrightarrow (\exists u \in \mathscr{T}(\sigma).\, a = @u) \wedge (t \in \mathscr{T}(\tau))$
   (d) $s \in \mathscr{T}(\mathsf{P}_\perp(\sigma)) \Longrightarrow (a = \Downarrow) \wedge (t \in \mathscr{T}(\sigma))$
5. For all states s:
   (a) $s \in \mathscr{T}(\mathsf{sum}\,\langle\sigma_n \mid n < \kappa\rangle) \Longrightarrow \exists! m < \kappa.\, \exists t \in \mathscr{T}(\sigma_m).\, s \xrightarrow{m} t$
   (b) $s \in \mathscr{T}(\mathsf{prod}\,\langle\sigma_n \mid n < \kappa\rangle) \Longrightarrow \forall n < \kappa.\, \exists t \in \mathscr{T}(\sigma_n).\, s \xrightarrow{n} t$
   (c) $s \in \mathscr{T}(\sigma \to \tau) \Longrightarrow \forall u \in \mathscr{T}(\sigma).\, \exists t \in \mathscr{T}(\tau).\, s \xrightarrow{@u} t$
   (d) $s \in \mathscr{T}(\mathsf{P}_\perp(\sigma)) \Longrightarrow (s \Uparrow^{\mathrm{may}}) \vee (\exists t \in \mathscr{T}(\sigma).\, s \xrightarrow{\Downarrow} t)$
6. If $s \xrightarrow{a} t$, $s \xrightarrow{a} u$ and $t \neq u$, then s has a computation type.

For a state $s \in \mathscr{T}(\mathsf{prod}\,\langle\sigma_n \mid n < \kappa\rangle)$ and $m < \kappa$, we write $s@m$ for the unique state such that $s \xrightarrow{m} s@m$. For states $s \in \mathscr{T}(\sigma \to \tau)$ and $t \in \mathscr{T}(\sigma)$, we write $s@t$ for the unique state such that $s \xrightarrow{@t} s@t$.

The dual $\to^{\mathrm{op}}$ of the transition relation determined by the labelled transition relation of a TTS is always well-founded, so there are no infinite sequences of transitions. This is because the type system provides a well-founded measure upon states, and the target state of a transition always has a type strictly smaller than the type of the source state.

It is useful to pick out the states of a TTS that cannot diverge or are deterministic at computation types, and that have the same property at all successor states. By lemma 2.3.11 and the well-founded measure on the transition relation given by the type system, these sets of states can be defined inductively or coinductively.

Definition 4.1.2 Let $\mathscr{T}$ be a TTS. The ***hereditarily total states*** $\mathit{Total}(\mathscr{T}) \subseteq \mathscr{T}$ and the ***hereditarily deterministic states*** $\mathit{Det}(\mathscr{T}) \subseteq \mathscr{T}$ are the greatest sets such that, for all states s, t:

1. (a) If $s \in \mathit{Total}(\mathscr{T})$ and $s \to t$, then $t \in \mathit{Total}(\mathscr{T})$.

   (b) If $s \in \mathit{Det}(\mathscr{T})$ and $s \to t$, then $t \in \mathit{Det}(\mathscr{T})$.

2. (a) If $s \in \mathit{Total}(\mathscr{T})$ and s has a computation type, then $s \Downarrow^{\mathrm{must}}$.

   (b) If $s \in \mathit{Det}(\mathscr{T})$ and s has a computation type, then $s \Uparrow^{\mathrm{may}}$ and there are no transitions from s, or $s \Downarrow^{\mathrm{must}}$ and there is exactly one transition from s.

Of course, if a state has type σ and $\mathit{POrd}(\sigma) = 0$, then the state must be hereditarily total and hereditarily deterministic. Hereditarily total and hereditarily deterministic states are considered in lemma 4.2.5.

Non-deterministic λ-calculi determine TTSs. In chapter 5 we show that $\mathscr{L}$ determines a TTS, as does each fragment of $\mathscr{L}$. The states of the TTSs are programs, and transitions are defined in terms of the evaluation semantics and a decomposition of the outermost constructor of canonical programs. For example, a program M of type $\mathsf{sum}\,\langle\sigma_n \mid n < \kappa\rangle$ has a transition to N, labelled by m, whenever it may converge to $\mathsf{inj}\,m\,\mathsf{of}\,N$.

However, not all TTSs arise from a programming language with an operational semantics. In sections 4.3 and 4.4 we study a TTS $\mathscr{S}$ that is related to the $\in$-TSWDs described in example 2.2.9. If the set of states of type σ is $\mathscr{S}(\sigma)$, then the set of states of type $\mathsf{P}_\perp(\sigma)$ is defined by $\mathscr{S}(\mathsf{P}_\perp(\sigma)) \stackrel{\mathrm{def}}{=} \mathcal{P}_{\mathrm{ne}}(\mathscr{S}(\sigma)_\perp)$. The states of coproduct, product, and function types are similarly defined in terms of the states of their component types.

TTSs are related to a number of other structures that abstract the application of an object representing a function to an argument. In the remainder of this section, we describe some of those structures. Such structures can be broadly classified according to whether or not they have a well-founded type system, where coinductive or recursive types are considered to be non-well-founded even though their syntax is well-founded. If there is a well-founded type system, then inductive techniques such as logical relations can be employed, otherwise coinductive techniques such as similarity and bisimilarity must be used.

Applicative structures, pre-frames, frames, and Henkin models (see [Gun92, Mit96]) use well-founded type systems. They are defined with reference to a signature consisting of a set of symbols for constants, and a set of ground or base types. The type systems are obtained by closing under the function type constructor (and sometimes product types). An applicative structure is a family of sets indexed by types, say $\langle A(\sigma) \mid \sigma \text{ a type}\rangle$, a family of application functions indexed by pairs of types $\langle App^{\sigma,\tau} : A(\sigma \to \tau) \times A(\sigma) \to A(\tau) \mid \sigma,\tau \text{ types}\rangle$, and an element in $A(\sigma)$ for each constant symbol of type σ. The other structures impose additional constraints. Specifically, each $A(\sigma)$ may be required to be non-empty, the sets $A(\sigma \to \tau)$ may be restricted to subsets of $A(\sigma) \to A(\tau)$ so that elements of $A(\sigma \to \tau)$ are equal if they have the same applicative behaviour (extensionality), or the structure may be required to support an interpretation of the simply-typed λ-calculus. The syntax and operational semantics of a simply-typed λ-calculus determine an applicative structure. In addition, there are many examples of syntax-free applicative structures, e.g., based on interpreting the function type constructor as the set-theoretic function space, the continuous function space between directed-complete partial orders, or as a set of partial recursive functions.

An applicative structure determines an LTS with the disjoint union of the sets $A(\sigma)$ as states. The labelled transition relation replaces the family of application functions because there is a labelled transition $a \xrightarrow{@b} App^{\sigma,\tau}(a,b)$ if and only if $a \in A(\sigma \to \tau)$ and $b \in A(\sigma)$.

Logical relations can be defined upon the elements of an applicative structure. We say that elements $a_1, b_1 \in A(\sigma \to \tau)$ are related if and only if $App^{\sigma,\tau}(a_1,a_2)$ and $App^{\sigma,\tau}(b_1,b_2)$ are related, for all related $a_2, b_2 \in A(\sigma)$. The type system must be well-founded for this definition to make sense because of the (contravariant) requirement that a_2 and b_2 are related.

In contrast to applicative structures and the other structures mentioned above, (quasi-)applicative transition systems, (quasi-)non-deterministic applicative transition systems, σ-transition systems, and LTSs can be used to abstract the application operation in a setting with no types, or coinductive or recursive types. There are also untyped variations of applicative structures or applicative structures with divergence (see [Bar84, HS86, Abr90]).

A quasi-applicative transition system (qATS) consists of a set A and a partial function $Ev : A \rightharpoonup (A \to A)$ (see [Abr90]). Closed terms of the lazy λ-calculus form a qATS. For a closed term M of the lazy λ-calculus, $Ev(M)$ is defined if there exists a term N such that M converges to $\lambda x.N$. In this case, for all closed terms L, $Ev(M)(L) = N[L/x]$. There are also syntax-free examples of qATSs, e.g., those arising from domain-theoretic models of the lazy λ-calculus.

A preorder, *applicative similarity*, and an equivalence, *applicative bisimilarity*, can be defined upon the states of a qATS by coinduction (see [Abr90, AO93]). However, a function similar to the one used to define a logical relation is not monotonic without a well-founded type system because of the need to test states $a_1, b_1 \in A$ by applying them to related states $a_2, b_2 \in A$. Abramsky's definition sidesteps this issue and only tests states $a_1, b_1 \in A$ by applying them both to the same state $c \in A$. This raises the question of whether or not the qATS satisfies a compatibility property: if $a, b, c \in A$ are such that $Ev(a)$ is defined and b and c are related by applicative similarity, then are $Ev(a)(b)$ and $Ev(a)(c)$ also related by applicative similarity? If so, the qATS is called an applicative transition system (ATS), and the Ev function is well-defined on the equivalence classes of A with respect to applicative similarity. Abramsky [Abr90] uses a domain logic to prove that the qATS obtained from the lazy λ-calculus is an ATS. Howe [How89] proves the same result using a syntactic technique, variations of which are used in chapter 5.

A qATS can also be presented as an LTS or an LTSWD (although the may divergence predicate is redundant because a qATS models deterministic behaviour). For $a, b \in A$, there is a transition $a \xrightarrow{@b} Ev(a)(b)$ if and only if $Ev(a)$ is defined. Applicative similarity and applicative bisimilarity on qATSs correspond to similarity and bisimilarity on such LTSs. Gordon [Gor94] introduces this idea by presenting a specific LTS for an FPC-like language (see also [Gor95a]). Note that these LTSs do not have τ-labelled transitions, and so there is no notion of weak similarity or weak bisimilarity.

Ong [Ong92a, Ong92b, Ong93] generalises qATSs to *quasi-non-deterministic applicative transition systems* (qNATS) by incorporating a may divergence predicate and allowing the evaluation operation to be a relation instead of a partial function, so $Ev \subseteq A \times (A \to A)$. Every $a \in A$ is required to either diverge or be related to at least one function by Ev. qNATSs can be used to study untyped non-deterministic λ-calculi as qATSs can be used to study untyped deterministic λ-calculi. Every qNATS determines an LTSWD, and the relation on the qNATS called "applicative bisimulation" in [Ong92a, Ong92b, Ong93] corresponds to convex similarity on the LTSWD.

For consistency with the terminology of qATSs and qNATSs, TTSs should be named quasi-TTSs to indicate that application need not be well-behaved with respect to a coinductively-defined relation. However, that convention is not followed here because several variants of similarity and bisimilarity are used with TTSs, and it is possible that application is well-behaved with respect to some relations but not others.

Ong and Pitts [OP93] also propose σ-*evaluation systems*, with coinductively-defined relations σ-*simulation*, to unify LTSWDs, qATSs, and qNATSs. For each type σ, they generate a class of structures called σ-evaluation systems, and a preorder, σ-simulation, on the states of those structures. The type σ can be chosen so that LTSWDs, qATSs, and qNATSs are σ-evaluation systems.

4.2 Similarity and Bisimilarity

The lower, upper, convex, and refinement variants of similarity and bisimilarity are defined in section 2.4 for LTSWDs. In this section we review and study these relations in more detail, and focus on the special case of TTSs. The discussion covers the inclusions between variants of similarity and bisimilarity on LTSWDs, and conditions upon the states of a TTS under which the inclusions collapse to equalities. We conclude with a definition of applicative compatibility for TTSs that permits a quotient structure to be constructed.

Recall that the variants of similarity and bisimilarity are defined in terms of a lower simulation function $\langle\cdot\rangle_{\mathrm{LS}}$ and an upper simulation function $\langle\cdot\rangle_{\mathrm{US}}$ (see definition 2.4.10). When these functions, or the variants of similarity and bisimilarity, are used with TTSs we annotate them with the TTS, e.g., $\langle\cdot\rangle^{\mathscr{T}}_{\mathrm{LS}}$ and $\langle\cdot\rangle^{\mathscr{T}}_{\mathrm{US}}$, because the TTSs arising from fragments of $\mathscr{L}$ can have common states of function type that behave differently in different TTSs. For example, in the TTS determined by the fragment $\mathscr{L}(\emptyset)$, a program M of type $\mathsf{P}_{\perp}(\mathsf{bool}) \to \mathsf{P}_{\perp}(\mathsf{bool})$ can only be applied to programs from $\mathscr{L}(\emptyset)$, which are always deterministic. However, in the TTS determined by $\mathscr{L}(?\langle \mathsf{false}, \mathsf{true}\rangle)$, M has transitions labelled with non-deterministic terms such as $?\langle \mathsf{false}, \mathsf{true}\rangle$. Consequently, the variants of similarity and bisimilarity need not be conservative between fragments $\mathscr{L}(\mathcal{E}_1)$ and $\mathscr{L}(\mathcal{E}_2)$, even when $\mathcal{E}_1 \subseteq \mathcal{E}_2$. That is, if the TTSs determined by those fragments are denoted by $\mathscr{L}_0(\mathcal{E}_1)$ and $\mathscr{L}(\mathcal{E}_2)$, there may be programs $M, N \in \mathscr{L}(\mathcal{E}_1)$ such that $M \simeq_{\mathrm{CB}}^{\mathscr{L}_0(\mathcal{E}_1)} N$ and $M \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(\mathcal{E}_2)} N$. The relationship between relations on overlapping TTSs is examined in sections 4.5 and 5.6.

The definitions of $\langle\cdot\rangle_{\mathrm{LS}}$ and $\langle\cdot\rangle_{\mathrm{US}}$ are modified slightly for TTSs because states should only be related if they have the same type. In definition 4.2.1 the functions are presented for TTSs using the labelled transitions specific to each type. The two functions have the same behaviour upon states of value types. States of a coproduct type are related if they are in the same component of the coproduct and their target states are related. States of a product or function type are related if every pair of projections is related. Using this definition means that we have to prove that function application in a TTS behaves reasonably with respect to the variants of similarity and bisimilarity. This is the purpose of the compatibility theorems for TTSs determined by fragments of $\mathscr{L}$ (see chapter 5).

It is tempting to make use of the fact that $\mathscr{L}$ has a well-founded type system by defining a logical relation. However, we do not have a proof of the fundamental theorem of logical relations for a non-deterministic programming language. Operationally-based proofs for deterministic languages are usually based upon a syntactic unwinding lemma, and the status of such properties in the presence of non-determinism is not yet clear. In addition, the logical relations approach would not extend to recursive types.

Definition 4.2.1 Let $\mathscr{T}$ be a TTS and $\mathcal{R} \subseteq \mathscr{T} \times \mathscr{T}$. The relations $\langle \mathcal{R}\rangle_{\mathrm{LS}}^{\mathscr{T}} \subseteq \mathscr{T} \times \mathscr{T}$ and $\langle \mathcal{R}\rangle_{\mathrm{US}}^{\mathscr{T}} \subseteq \mathscr{T} \times \mathscr{T}$ are defined, for $s_1, t_1 \in \mathscr{T}$, by:

1. If s_1 and t_1 have the same value type, then $\langle s_1, t_1\rangle \in \langle \mathcal{R}\rangle_{\mathrm{LS}}^{\mathscr{T}}$ and $\langle s_1, t_1\rangle \in \langle \mathcal{R}\rangle_{\mathrm{US}}^{\mathscr{T}}$ if and only if:

   (a) $s_1, t_1 \in \mathscr{T}(\mathsf{sum}\,\langle \sigma_n \mid n < \kappa\rangle) \wedge$
   $\exists m < \kappa.\, \exists s_2, t_2 \in \mathscr{T}(\sigma_m).\, s_1 \xrightarrow{m} s_2 \wedge t_1 \xrightarrow{m} t_2 \wedge \langle s_2, t_2\rangle \in \mathcal{R}$

   (b) $s_1, t_1 \in \mathscr{T}(\mathsf{prod}\,\langle \sigma_n \mid n < \kappa\rangle) \wedge \forall m < \kappa.\, \langle s_1 @ m, t_1 @ m\rangle \in \mathcal{R}$

   (c) $s_1, t_1 \in \mathscr{T}(\sigma \to \tau) \wedge \forall u \in \mathscr{T}(\sigma).\, \langle s_1 @ u, t_1 @ u\rangle \in \mathcal{R}$

2. If s_1 and t_1 have the same type $\mathsf{P}_\perp(\sigma)$, then:

   (a) $\langle s_1, t_1\rangle \in \langle \mathcal{R}\rangle_{\mathrm{LS}}^{\mathscr{T}}$ if and only if:
   $\forall s_2 \in \mathscr{T}(\sigma).\, s_1 \xrightarrow{\Downarrow} s_2 \Longrightarrow \exists t_2 \in \mathscr{T}(\sigma).\, t_1 \xrightarrow{\Downarrow} t_2 \wedge \langle s_2, t_2\rangle \in \mathcal{R}$

   (b) $\langle s_1, t_1\rangle \in \langle \mathcal{R}\rangle_{\mathrm{US}}^{\mathscr{T}}$ if and only if:
   $s_1 \Downarrow^{\mathrm{must}} \Longrightarrow$
   $(t_1 \Downarrow^{\mathrm{must}} \wedge \forall t_2 \in \mathscr{T}(\sigma).\, t_1 \xrightarrow{\Downarrow} t_2 \Longrightarrow \exists s_2 \in \mathscr{T}(\sigma).\, s_1 \xrightarrow{\Downarrow} s_2 \wedge \langle s_2, t_2\rangle \in \mathcal{R})$

In definition 4.2.1, the clause for $\langle\cdot\rangle^{\mathscr{T}}_{\mathrm{US}}$ at a computation type can be rewritten to emphasise that every divergent or convergent behaviour of t_1 is matched by a suitable behaviour of s_1. For $s_1, t_1 \in \mathscr{T}(\mathsf{P}_\perp(\sigma))$, we have that $\langle s_1, t_1\rangle \in \langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{US}}$ if and only if:

$$
\begin{aligned}
&(t_1 \Uparrow^{\mathrm{may}} \Longrightarrow s_1 \Uparrow^{\mathrm{may}}) \wedge \\
&(\forall t_2 \in \mathscr{T}(\sigma).\, t_1 \xrightarrow{\Downarrow} t_2 \Longrightarrow (s_1 \Uparrow^{\mathrm{may}}) \vee (\exists s_2 \in \mathscr{T}(\sigma).\, s_1 \xrightarrow{\Downarrow} s_2 \wedge \langle s_2, t_2\rangle \in \mathcal{R}))
\end{aligned}
$$

The functions $\langle\cdot\rangle^{\mathscr{T}}_{\mathrm{LS}}$ and $\langle\cdot\rangle^{\mathscr{T}}_{\mathrm{US}}$ are monotone with respect to the inclusion partial order. In addition, they satisfy the following property with respect to relational composition:

Lemma 4.2.2 For a TTS $\mathscr{T}$ and relations $\mathcal{R}, \mathcal{S} \subseteq \mathscr{T} \times \mathscr{T}$:

1. $\langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{LS}};\langle\mathcal{S}\rangle^{\mathscr{T}}_{\mathrm{LS}} \subseteq \langle\mathcal{R};\mathcal{S}\rangle^{\mathscr{T}}_{\mathrm{LS}}$

2. $\langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{US}};\langle\mathcal{S}\rangle^{\mathscr{T}}_{\mathrm{US}} \subseteq \langle\mathcal{R};\mathcal{S}\rangle^{\mathscr{T}}_{\mathrm{US}}$

Proof Straightforward. □

In definition 2.4.11, lower, upper, convex, and refinement variants of similarity, mutual similarity, and bisimilarity are defined for LTSWDs in terms of $\langle\cdot\rangle_{\mathrm{LS}}$ and $\langle\cdot\rangle_{\mathrm{US}}$. The same variants can be defined for a TTS $\mathscr{T}$ using $\langle\cdot\rangle^{\mathscr{T}}_{\mathrm{LS}}$ and $\langle\cdot\rangle^{\mathscr{T}}_{\mathrm{US}}$. We use the same names for the relations defined on TTSs, even though they may be finer than the corresponding relations when a TTS is considered as an LTSWD because the latter can relate states of different type. In practice, this should cause no confusion because we only ever consider whether states of the same type are related.

Definition 4.2.3 uses coinduction to define the relations, but the well-founded type system and the fact that every transition decreases the size of the types of the states, in conjunction with lemma 2.3.11, implies that the relations are the unique fixed-points. However, in the sequel, coinductive methods are used whenever possible to make it easier to extend results to structures with recursive types.

Definition 4.2.3 For a TTS $\mathscr{T}$, the lower, upper, convex, and refinement variants of similarity, mutual similarity, and bisimilarity are the binary relations on $\mathscr{T}$ defined by:

$\lesssim^{\mathscr{T}}_{\mathrm{LS}} \stackrel{\mathrm{def}}{=} \nu\mathcal{R}.\,\langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{LS}}$	(lower similarity)
$\simeq^{\mathscr{T}}_{\mathrm{LS}} \stackrel{\mathrm{def}}{=} \lesssim^{\mathscr{T}}_{\mathrm{LS}} \cap (\lesssim^{\mathscr{T}}_{\mathrm{LS}})^{\mathrm{op}}$	(mutual lower similarity)
$\simeq^{\mathscr{T}}_{\mathrm{LB}} \stackrel{\mathrm{def}}{=} \nu\mathcal{R}.\,\langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{LS}} \cap (\langle\mathcal{R}^{\mathrm{op}}\rangle^{\mathscr{T}}_{\mathrm{LS}})^{\mathrm{op}}$	(lower bisimilarity)
$\lesssim^{\mathscr{T}}_{\mathrm{US}} \stackrel{\mathrm{def}}{=} \nu\mathcal{R}.\,\langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{US}}$	(upper similarity)
$\simeq^{\mathscr{T}}_{\mathrm{US}} \stackrel{\mathrm{def}}{=} \lesssim^{\mathscr{T}}_{\mathrm{US}} \cap (\lesssim^{\mathscr{T}}_{\mathrm{US}})^{\mathrm{op}}$	(mutual upper similarity)
$\simeq^{\mathscr{T}}_{\mathrm{UB}} \stackrel{\mathrm{def}}{=} \nu\mathcal{R}.\,\langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{US}} \cap (\langle\mathcal{R}^{\mathrm{op}}\rangle^{\mathscr{T}}_{\mathrm{US}})^{\mathrm{op}}$	(upper bisimilarity)
$\lesssim^{\mathscr{T}}_{\mathrm{CS}} \stackrel{\mathrm{def}}{=} \nu\mathcal{R}.\,\langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{LS}} \cap \langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{US}}$	(convex similarity)
$\simeq^{\mathscr{T}}_{\mathrm{CS}} \stackrel{\mathrm{def}}{=} \lesssim^{\mathscr{T}}_{\mathrm{CS}} \cap (\lesssim^{\mathscr{T}}_{\mathrm{CS}})^{\mathrm{op}}$	(mutual convex similarity)
$\simeq^{\mathscr{T}}_{\mathrm{CB}} \stackrel{\mathrm{def}}{=} \nu\mathcal{R}.\,\langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{LS}} \cap \langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{US}} \cap (\langle\mathcal{R}^{\mathrm{op}}\rangle^{\mathscr{T}}_{\mathrm{LS}})^{\mathrm{op}} \cap (\langle\mathcal{R}^{\mathrm{op}}\rangle^{\mathscr{T}}_{\mathrm{US}})^{\mathrm{op}}$	(convex bisimilarity)
$\lesssim^{\mathscr{T}}_{\mathrm{RS}} \stackrel{\mathrm{def}}{=} \nu\mathcal{R}.\,(\langle\mathcal{R}^{\mathrm{op}}\rangle^{\mathscr{T}}_{\mathrm{LS}})^{\mathrm{op}} \cap \langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{US}}$	(refinement similarity)
$\simeq^{\mathscr{T}}_{\mathrm{RS}} \stackrel{\mathrm{def}}{=} \lesssim^{\mathscr{T}}_{\mathrm{RS}} \cap (\lesssim^{\mathscr{T}}_{\mathrm{RS}})^{\mathrm{op}}$	(mutual refinement similarity)
$\simeq^{\mathscr{T}}_{\mathrm{RB}} \stackrel{\mathrm{def}}{=} \nu\mathcal{R}.(\langle\mathcal{R}^{\mathrm{op}}\rangle^{\mathscr{T}}_{\mathrm{LS}})^{\mathrm{op}} \cap \langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{US}} \cap \langle\mathcal{R}\rangle^{\mathscr{T}}_{\mathrm{LS}} \cap (\langle\mathcal{R}^{\mathrm{op}}\rangle^{\mathscr{T}}_{\mathrm{US}})^{\mathrm{op}}$	(refinement bisimilarity)

The names of the relations are summarised in the table below:

	Lower	Upper	Convex	Refinement
Similarity	$\lesssim^{\mathscr{T}}_{\mathrm{LS}}$	$\lesssim^{\mathscr{T}}_{\mathrm{US}}$	$\lesssim^{\mathscr{T}}_{\mathrm{CS}}$	$\lesssim^{\mathscr{T}}_{\mathrm{RS}}$
Mutual Similarity	$\simeq^{\mathscr{T}}_{\mathrm{LS}}$	$\simeq^{\mathscr{T}}_{\mathrm{US}}$	$\simeq^{\mathscr{T}}_{\mathrm{CS}}$	$\simeq^{\mathscr{T}}_{\mathrm{RS}}$
Bisimilarity	$\simeq^{\mathscr{T}}_{\mathrm{LB}}$	$\simeq^{\mathscr{T}}_{\mathrm{UB}}$	$\simeq^{\mathscr{T}}_{\mathrm{CB}}$	$\simeq^{\mathscr{T}}_{\mathrm{RB}}$

Refinement bisimilarity and convex bisimilarity are identical by definition.

By lemma 2.3.6, the variants of similarity are preorders and the variants of mutual similarity and bisimilarity are equivalences.

Definition 4.2.3 is concise, but it is useful to have expansions of each combination of the simulation functions. For value types, all of the variants of similarity, mutual similarity, and bisimilarity have the same expansion. If $\mathcal{R}$ is any variant of similarity, mutual similarity, or bisimilarity on $\mathscr{T}$ and $s_1, t_1 \in \mathscr{T}$ have a value type, then:

1. If $s_1, t_1 \in \mathscr{T}(\mathsf{sum}\,\langle\sigma_n \mid n < \kappa\rangle)$, then $\langle s_1, t_1\rangle \in \mathcal{R}$ if and only if:
   $\exists m < \kappa.\, \exists s_2, t_2.\, (s_1 \xrightarrow{m} s_2) \wedge (t_1 \xrightarrow{m} t_2) \wedge \langle s_2, t_2\rangle \in \mathcal{R}$

2. If $s_1, t_1 \in \mathscr{T}(\mathsf{prod}\,\langle\sigma_n \mid n < \kappa\rangle)$, then $\langle s_1, t_1\rangle \in \mathcal{R}$ if and only if:
   $\forall n < \kappa.\, \langle s_1 @ n, t_1 @ n\rangle \in \mathcal{R}$

3. If $s_1, t_1 \in \mathscr{T}(\sigma \to \tau)$, then $\langle s_1, t_1 \rangle \in \mathcal{R}$ if and only if:
   $\forall u \in \mathscr{T}(\sigma).\, \langle s_1 @ u, t_1 @ u \rangle \in \mathcal{R}$

In general, the relations are different at computation types. The expansions are given in figure 4.1 for states $s_1, t_1 \in \mathscr{T}(\mathsf{P}_\perp(\sigma))$.

We now examine the relationships between the variants of similarity and bisimilarity. Lemma 4.2.4 identifies inclusions that hold in all TTSs. In example 4.4.5, we show that some of the inclusions are strict.

Lemma 4.2.4 The inclusions between the lower, upper, convex, and refinement variants of similarity, mutual similarity, and bisimilarity depicted in figure 4.2 hold in any TTS $\mathscr{T}$.

Proof By corollary 2.3.9, and the fact that the mutual similarities and bisimilarities are equivalences. □

Under certain conditions on states, there are identifications between the relations. For example, if the P-order of a type is 0, then the restrictions of the variants of similarity, mutual similarity, and bisimilarity to states of that type are all the same. Also, if the P-order of a type is 1, then the restrictions of the mutual similarities coincide with the restrictions of the bisimilarities. Lemma 4.2.5 proves these identifications, as well as some for hereditarily deterministic and hereditarily total states.

Lemma 4.2.5 For a TTS $\mathscr{T}$ and $s_1, t_1 \in \mathscr{T}(\sigma)$:

1. If $POrd(\sigma) = 0$, then:

$$\begin{aligned}
&s_1 \lesssim^{\mathscr{T}}_{\mathrm{LS}} t_1 \iff s_1 \lesssim^{\mathscr{T}}_{\mathrm{US}} t_1 \iff s_1 \lesssim^{\mathscr{T}}_{\mathrm{CS}} t_1 \iff s_1 \lesssim^{\mathscr{T}}_{\mathrm{RS}} t_1 \iff \\
&s_1 \simeq^{\mathscr{T}}_{\mathrm{LS}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{US}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{CS}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{RS}} t_1 \iff \\
&s_1 \simeq^{\mathscr{T}}_{\mathrm{LB}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{UB}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_1
\end{aligned}$$

2. If $POrd(\sigma) = 1$, then:
   (a) $s_1 \simeq^{\mathscr{T}}_{\mathrm{LS}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{LB}} t_1$
   (b) $s_1 \simeq^{\mathscr{T}}_{\mathrm{US}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{UB}} t_1$
   (c) $s_1 \simeq^{\mathscr{T}}_{\mathrm{CS}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{RS}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_1$
3. If $s_1, t_1 \in Total(\mathscr{T})$, then:
   (a) $s_1 \lesssim^{\mathscr{T}}_{\mathrm{LS}} t_1 \iff t_1 \lesssim^{\mathscr{T}}_{\mathrm{US}} s_1 \iff t_1 \lesssim^{\mathscr{T}}_{\mathrm{RS}} s_1$
   (b) $s_1 \simeq^{\mathscr{T}}_{\mathrm{LS}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{US}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{RS}} t_1$
   (c) $s_1 \lesssim^{\mathscr{T}}_{\mathrm{CS}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{CS}} t_1 \iff$
   $s_1 \simeq^{\mathscr{T}}_{\mathrm{LB}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{UB}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_1$

$$
\begin{aligned}
& s_1 \lesssim^{\mathscr{T}}_{\mathrm{LS}} t_1 \iff \\
& \forall s_2.\, s_1 \xrightarrow{\Downarrow} s_2 \implies \exists t_2.\, t_1 \xrightarrow{\Downarrow} t_2 \wedge s_2 \lesssim^{\mathscr{T}}_{\mathrm{LS}} t_2 \\[1em]
& s_1 \lesssim^{\mathscr{T}}_{\mathrm{US}} t_1 \iff \\
& s_1 \Downarrow^{\mathrm{must}} \implies (t_1 \Downarrow^{\mathrm{must}} \wedge \forall t_2.\, t_1 \xrightarrow{\Downarrow} t_2 \implies \exists s_2.\, s_1 \xrightarrow{\Downarrow} s_2 \wedge s_2 \lesssim^{\mathscr{T}}_{\mathrm{US}} t_2) \\[1em]
& s_1 \lesssim^{\mathscr{T}}_{\mathrm{CS}} t_1 \iff \\
& (\forall s_2.\, s_1 \xrightarrow{\Downarrow} s_2 \implies \exists t_2.\, t_1 \xrightarrow{\Downarrow} t_2 \wedge s_2 \lesssim^{\mathscr{T}}_{\mathrm{CS}} t_2) \wedge \\
& (s_1 \Downarrow^{\mathrm{must}} \implies (t_1 \Downarrow^{\mathrm{must}} \wedge \forall t_2.\, t_1 \xrightarrow{\Downarrow} t_2 \implies \exists s_2.\, s_1 \xrightarrow{\Downarrow} s_2 \wedge s_2 \lesssim^{\mathscr{T}}_{\mathrm{CS}} t_2)) \\[1em]
& s_1 \lesssim^{\mathscr{T}}_{\mathrm{RS}} t_1 \iff \\
& (s_1 \Downarrow^{\mathrm{must}} \implies t_1 \Downarrow^{\mathrm{must}}) \wedge \\
& (\forall t_2.\, t_1 \xrightarrow{\Downarrow} t_2 \implies \exists s_2.\, s_1 \xrightarrow{\Downarrow} s_2 \wedge s_2 \lesssim^{\mathscr{T}}_{\mathrm{RS}} t_2) \\[1em]
& s_1 \simeq^{\mathscr{T}}_{\mathrm{LB}} t_1 \iff \\
& (\forall s_2.\, s_1 \xrightarrow{\Downarrow} s_2 \implies \exists t_2.\, t_1 \xrightarrow{\Downarrow} t_2 \wedge s_2 \simeq^{\mathscr{T}}_{\mathrm{LB}} t_2) \wedge \\
& (\forall t_2.\, t_1 \xrightarrow{\Downarrow} t_2 \implies \exists s_2.\, s_1 \xrightarrow{\Downarrow} s_2 \wedge s_2 \simeq^{\mathscr{T}}_{\mathrm{LB}} t_2) \\[1em]
& s_1 \simeq^{\mathscr{T}}_{\mathrm{UB}} t_1 \iff \\
& (s_1 \Uparrow^{\mathrm{may}} \wedge t_1 \Uparrow^{\mathrm{may}}) \vee \\
& ((s_1 \Downarrow^{\mathrm{must}} \wedge t_1 \Downarrow^{\mathrm{must}}) \wedge \\
& (\forall s_2.\, s_1 \xrightarrow{\Downarrow} s_2 \implies \exists t_2.\, t_1 \xrightarrow{\Downarrow} t_2 \wedge s_2 \simeq^{\mathscr{T}}_{\mathrm{UB}} t_2) \wedge \\
& (\forall t_2.\, t_1 \xrightarrow{\Downarrow} t_2 \implies \exists s_2.\, s_1 \xrightarrow{\Downarrow} s_2 \wedge s_2 \simeq^{\mathscr{T}}_{\mathrm{UB}} t_2)) \\[1em]
& s_1 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_1 \iff \\
& (s_1 \Downarrow^{\mathrm{must}} \iff t_1 \Downarrow^{\mathrm{must}}) \wedge \\
& (\forall s_2.\, s_1 \xrightarrow{\Downarrow} s_2 \implies \exists t_2.\, t_1 \xrightarrow{\Downarrow} t_2 \wedge s_2 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_2) \wedge \\
& (\forall t_2.\, t_1 \xrightarrow{\Downarrow} t_2 \implies \exists s_2.\, s_1 \xrightarrow{\Downarrow} s_2 \wedge s_2 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_2)
\end{aligned}
$$

Figure 4.1: Unfoldings of similarity and bisimilarity for a TTS at $\mathsf{P}_\perp(\sigma)$

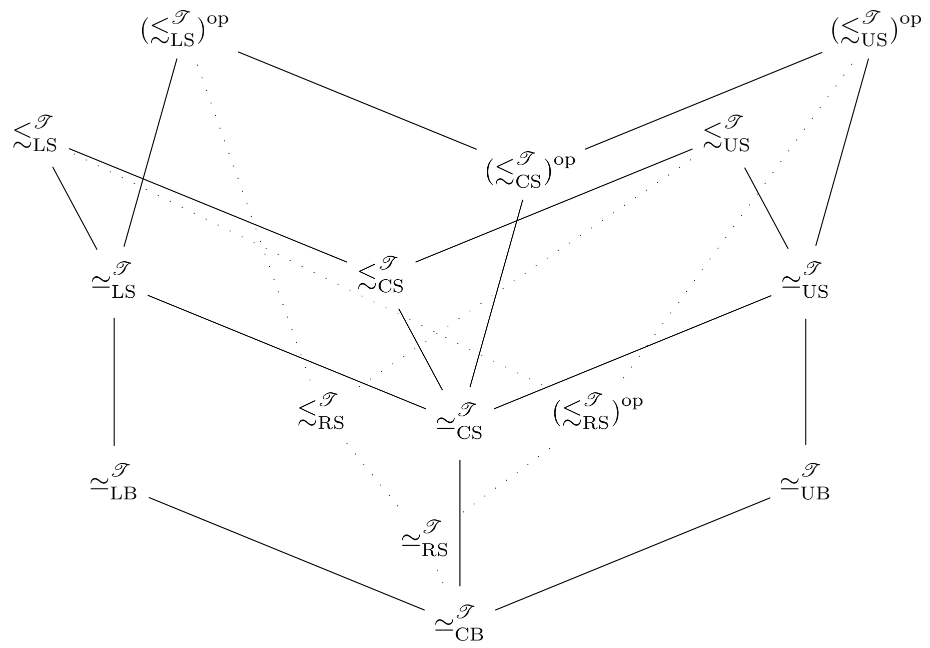


Figure 4.2: Inclusions between similarities and bisimilarities

Note that the order of s_1 and t_1 is reversed for upper similarity and refinement similarity.

4. If $s_1, t_1 \in \mathit{Det}(\mathscr{T})$, then:

   (a) $s_1 \lesssim^{\mathscr{T}}_{\mathrm{LS}} t_1 \iff s_1 \lesssim^{\mathscr{T}}_{\mathrm{US}} t_1 \iff s_1 \lesssim^{\mathscr{T}}_{\mathrm{CS}} t_1$

   (b) $s_1 \lesssim^{\mathscr{T}}_{\mathrm{RS}} t_1 \iff$
   $s_1 \simeq^{\mathscr{T}}_{\mathrm{LS}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{US}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{CS}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{RS}} t_1 \iff$
   $s_1 \simeq^{\mathscr{T}}_{\mathrm{LB}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{UB}} t_1 \iff s_1 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_1$

Proof

1. If $\mathcal{R} \subseteq \mathscr{T} \times \mathscr{T}$ and $s, t \in \mathscr{T}(\sigma)$, where σ is a value type, the following equivalences hold: $\langle s,t \rangle \in \langle \mathcal{R} \rangle^{\mathscr{T}}_{\mathrm{LS}} \iff \langle s,t \rangle \in (\langle \mathcal{R}^{\mathrm{op}} \rangle^{\mathscr{T}}_{\mathrm{LS}})^{\mathrm{op}} \iff \langle s,t \rangle \in \langle \mathcal{R} \rangle^{\mathscr{T}}_{\mathrm{US}} \iff \langle s,t \rangle \in (\langle \mathcal{R}^{\mathrm{op}} \rangle^{\mathscr{T}}_{\mathrm{US}})^{\mathrm{op}}$

   The result follows by coinduction.

2. By lemma 4.2.4, it suffices to prove by coinduction that the mutual similarities are contained in the bisimilarities. We show that $s_1 \simeq^{\mathscr{T}}_{\mathrm{CS}} t_1$ implies $s_1 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_1$. The other cases are similar. If s_1 and t_1 have computation type $\mathsf{P}_\perp(\sigma)$, then, by $s_1 \simeq^{\mathscr{T}}_{\mathrm{CS}} t_1$, we have that $s_1 \Downarrow^{\mathrm{must}}$ if and only if $t_1 \Downarrow^{\mathrm{must}}$. In addition, if there exists s_2 such that $s_1 \xrightarrow{\Downarrow} s_2$, then, using $s_1 \lesssim^{\mathscr{T}}_{\mathrm{CS}} t_1$, there exists t_2 such that $t_1 \xrightarrow{\Downarrow} t_2$ and $s_2 \lesssim^{\mathscr{T}}_{\mathrm{CS}} t_2$. But s_2 and t_2 have type σ with P-order 0, and so $s_2 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_2$, by (1). The other direction is similar, so $s_1 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_1$. Otherwise s_1 and t_1

have a value type with P-order 1. It is straightforward to show that whenever $s_1 \xrightarrow{a} s_2$ and $t_1 \xrightarrow{a} t_2$, we have $s_2 \simeq^{\mathcal{T}}_{\mathrm{CS}} t_2$, and s_2 and t_2 have the same type with P-order 1. Therefore, by coinduction, we are done.

3. The proofs are by coinduction and are straightforward at value types. For computation types, the results follow easily from $s_1 \Downarrow^{\mathrm{must}}$ and $t_1 \Downarrow^{\mathrm{must}}$.

4. The proof is by coinduction and is straightforward at value types. For computation types, recall that if s_1 has a computation type, then $s_1 \Uparrow^{\mathrm{may}}$ and s_1 has no transitions, or $s_1 \Downarrow^{\mathrm{must}}$ and s_1 has exactly one transition. We give the case for $s_1 \lesssim^{\mathcal{T}}_{\mathrm{RS}} t_1$ implies $s_1 \simeq^{\mathcal{T}}_{\mathrm{CB}} t_1$. If $s_1 \Downarrow^{\mathrm{must}}$, then $s_1 \lesssim^{\mathcal{T}}_{\mathrm{RS}} t_1$ implies $t_1 \Downarrow^{\mathrm{must}}$. If $t_1 \Downarrow^{\mathrm{must}}$, then, because $t_1 \in \mathit{Det}(\mathcal{T})$, there exists t_2 such that $t_1 \xrightarrow{\Downarrow} t_2$. By $s_1 \lesssim^{\mathcal{T}}_{\mathrm{RS}} t_1$, there exists s_2 such that $s_1 \xrightarrow{\Downarrow} s_2$ and $s_2 \lesssim^{\mathcal{T}}_{\mathrm{RS}} t_2$, so $s_1 \Downarrow^{\mathrm{must}}$, because $s_1 \in \mathit{Det}(\mathcal{T})$. Therefore, $s_1 \Downarrow^{\mathrm{must}}$ if and only if $t_1 \Downarrow^{\mathrm{must}}$. Now whenever there is a state s_2 such that $s_1 \xrightarrow{\Downarrow} s_2$, we have that $s_1 \Downarrow^{\mathrm{must}}$, so $t_1 \Downarrow^{\mathrm{must}}$ and there is a unique t_2 such that $t_1 \xrightarrow{\Downarrow} t_2$. By $s_1 \lesssim^{\mathcal{T}}_{\mathrm{RS}} t_1$ and the fact that the transition from s_1 is unique, we have $s_2 \lesssim^{\mathcal{T}}_{\mathrm{RS}} t_2$. For the other direction, suppose that there is a state t_2 such that $t_1 \xrightarrow{\Downarrow} t_2$. By $s_1 \lesssim^{\mathcal{T}}_{\mathrm{RS}} t_1$, we immediately get a state s_2 such that $s_1 \xrightarrow{\Downarrow} s_2$ and $s_2 \lesssim^{\mathcal{T}}_{\mathrm{RS}} t_2$. Applying coinduction, we have that $s_1 \simeq^{\mathcal{T}}_{\mathrm{CB}} t_1$. □

Representatives of the maximal equivalence classes with respect to the upper, convex, and refinement variants of similarity can be identified via the hereditarily total and hereditarily deterministic properties.

Lemma 4.2.6 Let $\mathcal{T}$ be a TTS and $s,t \in \mathcal{T}$ states with the same type. Then:

1. If $s \lesssim^{\mathcal{T}}_{\mathrm{US}} t$ and $s \in \mathit{Total}(\mathcal{T}) \cap \mathit{Det}(\mathcal{T})$, then $s \simeq^{\mathcal{T}}_{\mathrm{UB}} t$.

2. If $s \lesssim^{\mathcal{T}}_{\mathrm{CS}} t$ and $s \in \mathit{Total}(\mathcal{T})$, then $s \simeq^{\mathcal{T}}_{\mathrm{CB}} t$.

3. If $s \lesssim^{\mathcal{T}}_{\mathrm{RS}} t$ and $s \in \mathit{Det}(\mathcal{T})$, then $s \simeq^{\mathcal{T}}_{\mathrm{CB}} t$.

Proof

1. By coinduction. Define $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{T}$ by:

$$\mathcal{R} \stackrel{\mathrm{def}}{=} \{\langle s,t\rangle \in \mathcal{T} \times \mathcal{T} \mid \exists \sigma.\, (s,t \in \mathcal{T}(\sigma)) \wedge (s \in \mathit{Total}(\mathcal{T}) \cap \mathit{Det}(\mathcal{T})) \wedge (s \lesssim^{\mathcal{T}}_{\mathrm{US}} t)\}$$

We need to show that, for all $\langle s_1,t_1\rangle \in \mathcal{R}$, we have $\langle s_1,t_1\rangle \in \langle \mathcal{R}\rangle^{\mathcal{T}}_{\mathrm{US}} \cap (\langle \mathcal{R}^{\mathrm{op}}\rangle^{\mathcal{T}}_{\mathrm{US}})^{\mathrm{op}}$. The cases when s_1 and t_1 have a value type are straightforward. If s_1 and t_1 have type $\mathsf{P}_\perp(\sigma)$, then $s_1 \in \mathit{Total}(\mathcal{T})$ and $s_1 \lesssim^{\mathcal{T}}_{\mathrm{US}} t_1$ imply that $s_1 \Downarrow^{\mathrm{must}}$ and $t_1 \Downarrow^{\mathrm{must}}$. Now consider any $t_2 \in \mathcal{T}(\sigma)$ such that $t_1 \xrightarrow{\Downarrow} t_2$. By $s_1 \Downarrow^{\mathrm{must}}$ and $s_1 \lesssim^{\mathcal{T}}_{\mathrm{US}} t_1$, we have that there exists $s_2 \in$

$\mathscr{T}(\sigma)$ such that $s_1 \stackrel{\Downarrow}{\rightarrow} s_2$ and $s_2 \lesssim^{\mathscr{T}}_{\mathrm{US}} t_2$. However, s_2 is the unique successor of s_1 because $s_1 \in \mathit{Det}(\mathscr{T})$, so $\langle s_1, t_1 \rangle \in \langle \mathcal{R} \rangle^{\mathscr{T}}_{\mathrm{US}} \cap (\langle \mathcal{R}^{\mathrm{op}} \rangle^{\mathscr{T}}_{\mathrm{US}})^{\mathrm{op}}$, and we are done.

2. Proceed as in (1), redefining $\mathcal{R}$ for the premises of (2). If $s_1, t_1 \in \mathcal{R}$ have type $\mathsf{P}_\perp(\sigma)$, then $s_1 \in \mathit{Total}(\mathscr{T})$ and $s_1 \lesssim^{\mathscr{T}}_{\mathrm{CS}} t_1$ imply that $s_1 \Downarrow^{\mathrm{must}}$ and $t_1 \Downarrow^{\mathrm{must}}$. By $s_1 \lesssim^{\mathscr{T}}_{\mathrm{CS}} t_1$, we have, for all $s_2 \in \mathscr{T}(\sigma)$ such that $s_1 \stackrel{\Downarrow}{\rightarrow} s_2$, there exists $t_2 \in \mathscr{T}(\sigma)$ such that $t_1 \stackrel{\Downarrow}{\rightarrow} t_2$ and $s_2 \lesssim^{\mathscr{T}}_{\mathrm{CS}} t_2$. The opposite direction also holds, because $s_1 \Downarrow^{\mathrm{must}}$, so, for all $t_2 \in \mathscr{T}(\sigma)$ such that $t_1 \stackrel{\Downarrow}{\rightarrow} t_2$, there exists $s_2 \in \mathscr{T}(\sigma)$ such that $s_1 \stackrel{\Downarrow}{\rightarrow} s_2$ and $s_2 \lesssim^{\mathscr{T}}_{\mathrm{CS}} t_2$. Therefore:

$$\langle s_1, t_1 \rangle \in \langle \mathcal{R} \rangle^{\mathscr{T}}_{\mathrm{LS}} \cap \langle \mathcal{R} \rangle^{\mathscr{T}}_{\mathrm{US}} \cap (\langle \mathcal{R}^{\mathrm{op}} \rangle^{\mathscr{T}}_{\mathrm{LS}})^{\mathrm{op}} \cap (\langle \mathcal{R}^{\mathrm{op}} \rangle^{\mathscr{T}}_{\mathrm{US}})^{\mathrm{op}}$$

3. Proceed as in (1), redefining $\mathcal{R}$ for the premises of (3). Suppose that $s_1, t_1 \in \mathcal{R}$ have type $\mathsf{P}_\perp(\sigma)$. If $s_1 \Uparrow^{\mathrm{may}}$, then $s_1 \in \mathit{Det}(\mathscr{T})$ implies that s_1 has no successors. Moreover, t_1 cannot have any successors either and so $t_1 \Uparrow^{\mathrm{may}}$, because $s_1 \lesssim^{\mathscr{T}}_{\mathrm{RS}} t_1$. Otherwise $s_1 \Downarrow^{\mathrm{must}}$, so $t_1 \Downarrow^{\mathrm{must}}$, also because $s_1 \lesssim^{\mathscr{T}}_{\mathrm{RS}} t_1$. Now consider any $t_2 \in \mathscr{T}(\sigma)$ such that $t_1 \stackrel{\Downarrow}{\rightarrow} t_2$. By $s_1 \lesssim^{\mathscr{T}}_{\mathrm{RS}} t_1$, we have that there exists $s_2 \in \mathscr{T}(\sigma)$ such that $s_1 \stackrel{\Downarrow}{\rightarrow} s_2$ and $s_2 \lesssim^{\mathscr{T}}_{\mathrm{RS}} t_2$. However, s_2 is the unique successor of s_1 because $s_1 \in \mathit{Det}(\mathscr{T})$, so:

$$\langle s_1, t_1 \rangle \in \langle \mathcal{R} \rangle^{\mathscr{T}}_{\mathrm{LS}} \cap \langle \mathcal{R} \rangle^{\mathscr{T}}_{\mathrm{US}} \cap (\langle \mathcal{R}^{\mathrm{op}} \rangle^{\mathscr{T}}_{\mathrm{LS}})^{\mathrm{op}} \cap (\langle \mathcal{R}^{\mathrm{op}} \rangle^{\mathscr{T}}_{\mathrm{US}})^{\mathrm{op}}$$

□

If the application operation in a TTS is well-behaved with respect to one of the variants of similarity and bisimilarity, then it is possible to define a quotient structure with a well-defined application operation (see section 4.5). Consider a TTS $\mathscr{T}$, a relation $\mathcal{R}$ which is one of the variants of similarity or bisimilarity on $\mathscr{T}$, and states $s_1, t_1 \in \mathscr{T}(\sigma \to \tau)$ and $s_2, t_2 \in \mathscr{T}(\sigma)$ such that $\langle s_1, t_1 \rangle \in \mathcal{R}$ and $\langle s_2, t_2 \rangle \in \mathcal{R}$. In this case, we can deduce that $\langle s_1 @ s_2, t_1 @ s_2 \rangle \in \mathcal{R}$ and $\langle s_1 @ t_2, t_1 @ t_2 \rangle \in \mathcal{R}$, but not $\langle s_1 @ s_2, t_1 @ t_2 \rangle \in \mathcal{R}$ (although, by transitivity of $\mathcal{R}$, we do have that if $\langle s_1 @ s_2, s_1 @ t_2 \rangle \in \mathcal{R}$ or $\langle t_1 @ s_2, t_1 @ t_2 \rangle \in \mathcal{R}$, then $\langle s_1 @ s_2, t_1 @ t_2 \rangle \in \mathcal{R}$). This is analogous to the property that must hold for a qATS to be an ATS (or for a qNATS to be a NATS). However, it is possible that the property holds for one of the variants of similarity and bisimilarity but not another. For this reason *applicative compatibility* is defined with respect to a particular relation.

Definition 4.2.7 Consider a TTS $\mathscr{T}$ and a relation $\mathcal{R}$ which is one of the variants of similarity, mutual similarity, or bisimilarity on $\mathscr{T}$. The relation $\mathcal{R}$ is ***applicatively compatible*** for a state $s \in \mathscr{T}(\sigma \to \tau)$ if, for all $t, u \in \mathscr{T}(\sigma)$, $\langle t, u \rangle \in R$ implies $\langle s @ t, s @ u \rangle \in \mathcal{R}$. The relation $\mathcal{R}$ is ***applicatively compatible*** with respect to $\mathscr{T}$ if $\mathcal{R}$ is applicatively compatible for every state of function type in $\mathscr{T}$.

Example 4.2.8 We define a finite TTS $\mathscr{T}$ for which convex bisimilarity is not applicatively compatible. The states of $\mathscr{T}$ are defined by:

$$\mathscr{T}(\sigma) \stackrel{\text{def}}{=} \begin{cases} \{\star_1, \star_2\} & \text{if } \sigma = \mathsf{unit} \\ \{\mathit{ff}, \mathit{tt}\} & \text{if } \sigma = \mathsf{bool} \\ \{s\} & \text{if } \sigma = \mathsf{unit} \to \mathsf{bool} \\ \emptyset & \text{otherwise} \end{cases}$$

The states $\star_1$ are $\star_2$ have no transitions because they have type $\mathsf{unit} = \mathsf{prod}\,\langle\rangle$. The transitions of the other states are:

$$s \xrightarrow{@\star_1} \mathit{ff} \xrightarrow{0} \star_1$$
$$s \xrightarrow{@\star_2} \mathit{tt} \xrightarrow{1} \star_1$$

Then $\simeq^{\mathscr{T}}_{\text{CB}}$ is not applicatively compatible because $\star_1 \simeq^{\mathscr{T}}_{\text{CB}} \star_2$, but:

$$s@\star_1 = \mathit{ff} \not\simeq^{\mathscr{T}}_{\text{CB}} \mathit{tt} = s@\star_2$$

If the states of a TTS have the property that no distinct states have the same behaviour, then the behaviour is called *extensional*. Behaviour is measured using the variants of mutual similarity and bisimilarity, and so extensionality of an equivalence relation simply means that every equivalence class contains exactly one state.

Definition 4.2.9 Consider a TTS $\mathscr{T}$ and a relation $\mathcal{R}$ which is one of the variants of mutual similarity or bisimilarity on $\mathscr{T}$. The relation $\mathcal{R}$ is ***extensional*** if, for all states $s, t \in \mathscr{T}(\sigma)$, $\langle s, t\rangle \in \mathcal{R}$ implies $s = t$.

Convex bisimilarity for the TTS $\mathscr{T}$ in example 4.2.8 is not extensional because $\star_1 \simeq^{\mathscr{T}}_{\text{CB}} \star_2$. In fact, it could not be extensional because extensionality implies applicative compatibility.

Lemma 4.2.10 Consider a TTS $\mathscr{T}$ and a relation $\mathcal{R}$ which is one of the variants of mutual similarity or bisimilarity on $\mathscr{T}$. If $\mathcal{R}$ is extensional, then it is also applicatively compatible.

Proof Consider $s \in \mathscr{T}(\sigma \to \tau)$ and $t, u \in \mathscr{T}(\sigma)$ such that $\langle t, u\rangle \in \mathcal{R}$. By extensionality, $t = u$, so $s@t = s@u$. Therefore $\langle s@t, s@u\rangle \in \mathcal{R}$, because $\mathcal{R}$ is reflexive. $\square$

In chapter 5 it is shown that the variants of mutual similarity and bisimilarity upon the TTSs determined by (some of) the fragments of $\mathscr{L}$ are applicatively compatible but not extensional, and so the converse of lemma 4.2.10 does not hold.

4.3 The TTS $\mathscr{S}$ and Bisimilarity

In this section we define a syntax-free TTS $\mathscr{S}$ for which convex bisimilarity is extensional. The construction of $\mathscr{S}$ relies upon the type system of $\mathscr{L}$ being well-founded because the set of states for each type is defined in terms of the sets of states for smaller types. For example, $\mathscr{S}(\sigma \to \tau)$ is the set-theoretic function space $\mathscr{S}(\sigma) \to \mathscr{S}(\tau)$, and $\mathscr{S}(\mathsf{P}_\perp(\sigma))$ is $\mathscr{P}_{\mathrm{ne}}(\mathscr{S}(\sigma)_\perp)$, the set of non-empty subsets of $\mathscr{S}(\sigma)_\perp$. The TTS $\mathscr{S}$ resembles full set-theoretic Henkin models (see [Mit96]) and the $\in$-TSWDs defined in section 2.2. In section 4.4, a number of examples are given in $\mathscr{S}$ to show that the inclusions between the variants of similarity, mutual similarity, and bisimilarity can be strict. Using an embedding result proved in section 4.5, these examples can be pulled back to the TTSs arising from the fragments of $\mathscr{L}$.

Definition 4.3.1 The states of the TTS $\mathscr{S}$ are defined by induction on types:

$$\begin{aligned}
\mathscr{S}(\mathsf{sum}\,\langle\sigma_n \mid n<\kappa\rangle) &\stackrel{\text{def}}{=} \{\mathsf{sum}\,\langle\sigma_n \mid n<\kappa\rangle\} \times \Big(\sum_{n<\kappa} \mathscr{S}(\sigma_n)\Big) \\
\mathscr{S}(\mathsf{prod}\,\langle\sigma_n \mid n<\kappa\rangle) &\stackrel{\text{def}}{=} \{\mathsf{prod}\,\langle\sigma_n \mid n<\kappa\rangle\} \times \Big(\prod_{n<\kappa} \mathscr{S}(\sigma_n)\Big) \\
\mathscr{S}(\sigma \to \tau) &\stackrel{\text{def}}{=} \{\sigma \to \tau\} \times (\mathscr{S}(\sigma) \to \mathscr{S}(\tau)) \\
\mathscr{S}(\mathsf{P}_\perp(\sigma)) &\stackrel{\text{def}}{=} \{\mathsf{P}_\perp(\sigma)\} \times \mathscr{P}_{\mathrm{ne}}(\mathscr{S}(\sigma)_\perp)
\end{aligned}$$

The purpose of the first component of each state is to ensure that the sets $\langle \mathscr{S}(\sigma) \mid \sigma \text{ a type}\rangle$ are pairwise disjoint, and is omitted when it can be inferred from the context. A state $A \in \mathscr{S}$ may diverge if and only if it has a computation type and $\perp \in A$. The labelled transition relation is defined by:

$$\begin{array}{rcll}
\langle m, A\rangle \in \mathscr{S}(\mathsf{sum}\,\langle\sigma_n \mid n<\kappa\rangle) & \xrightarrow{m} & A \in \mathscr{S}(\sigma_m) & \\
\langle A_n \mid n<\kappa\rangle \in \mathscr{S}(\mathsf{prod}\,\langle\sigma_n \mid n<\kappa\rangle) & \xrightarrow{m} & A_m \in \mathscr{S}(\sigma_m) & (m<\kappa) \\
f \in \mathscr{S}(\sigma \to \tau) & \xrightarrow{@A} & f(A) \in \mathscr{S}(\tau) & (A \in \mathscr{S}(\sigma)) \\
A \in \mathscr{S}(\mathsf{P}_\perp(\sigma)) & \xrightarrow{\Downarrow} & B \in \mathscr{S}(\sigma) & (B \in A \setminus \{\perp\})
\end{array}$$

We write $\star \in \mathscr{S}(\mathsf{unit})$ for the unique element of the unit type, and $\mathit{ff}, \mathit{tt} \in \mathscr{S}(\mathsf{bool})$ for the elements of the boolean type such that $\mathit{ff} \xrightarrow{0} \star$ and $\mathit{tt} \xrightarrow{1} \star$.

In the remainder of this section, we consider properties of the variants of bisimilarity on $\mathscr{S}$. Convex bisimilarity is straightforward because it is extensional.

Lemma 4.3.2 Convex bisimilarity $\simeq^{\mathscr{S}}_{\mathrm{CB}}$ on $\mathscr{S}$ is extensional, and hence applicatively compatible.

Proof By induction on the type of states. We give the case for states of a computation type $\mathsf{P}_\perp(\sigma)$. Consider $A, B \in \mathscr{S}(\mathsf{P}_\perp(\sigma))$ such that $A \simeq^{\mathscr{S}}_{\mathrm{CB}} B$. Now $\perp \in A$ if and only $\perp \in B$, because $A \Uparrow^{\mathrm{may}}$ if and only if $B \Uparrow^{\mathrm{may}}$. For $C \in A$ such that $C \neq \perp$, there exists $D \in B$ such that $D \neq \perp$ and $C \simeq^{\mathscr{S}}_{\mathrm{CB}} D$. By the induction hypothesis, $C = D$, so $A \subseteq B$. Similarly, $B \subseteq A$. Therefore $A = B$. Applicative compatibility follows by lemma 4.2.10. $\square$

In contrast, neither lower bisimilarity and upper bisimilarity are extensional or applicatively compatible with respect to $\mathscr{S}$. The failure of extensionality is due to identifications that are almost identical to those considered in section 2.6 for the $\in$-LTSWD for $\mathcal{P}_{\mathrm{ne}}(\omega_\perp)$. For example, the states $\{\perp\},\{\perp,\star\},\{\star\} \in \mathscr{S}(\mathsf{P}_\perp(\mathsf{unit}))$ satisfy:

$$\{\perp\} \not\simeq^{\mathscr{S}}_{\mathrm{LB}} \{\perp,\star\} \simeq^{\mathscr{S}}_{\mathrm{LB}} \{\star\}$$
$$\{\perp\} \simeq^{\mathscr{S}}_{\mathrm{UB}} \{\perp,\star\} \not\simeq^{\mathscr{S}}_{\mathrm{UB}} \{\star\}$$

More generally, for all $A \subseteq_{\mathrm{ne}} \mathscr{S}(\sigma)$, we have $A \simeq^{\mathscr{S}}_{\mathrm{LB}} A \cup \{\perp\}$. Also, for all $A,B \subseteq \mathscr{S}(\sigma)$, we have $A \cup \{\perp\} \simeq^{\mathscr{S}}_{\mathrm{UB}} B \cup \{\perp\}$. In contrast to the $\in$-LTSWD for $\mathcal{P}_{\mathrm{ne}}(\omega_\perp)$, these are not the only identifications, because other computation types may appear in σ. For example, the states $\{\{\perp,\star\}\},\{\{\star\}\} \in \mathscr{S}(\mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{unit})))$ do not fit the schema above, but $\{\{\perp,\star\}\} \simeq^{\mathscr{S}}_{\mathrm{LB}} \{\{\star\}\}$.

The examples above can also be used to demonstrate the failure of applicative compatibility. For lower bisimilarity, consider the function $f : \mathscr{S}(\mathsf{P}_\perp(\mathsf{unit})) \to \mathscr{S}(\mathsf{P}_\perp(\mathsf{unit}))$ that maps $\{\perp\}$ and $\{\perp,\star\}$ to $\{\perp\}$, and $\{\star\}$ to $\{\star\}$. The function f is a state of type $\mathscr{S}(\mathsf{P}_\perp(\mathsf{unit}) \to \mathsf{P}_\perp(\mathsf{unit}))$ and lower bisimilarity is not applicatively compatible for f because of the states $\{\perp,\star\} \simeq^{\mathscr{S}}_{\mathrm{LB}} \{\star\}$ that are mapped to $\{\perp\} \not\simeq^{\mathscr{S}}_{\mathrm{LB}} \{\star\}$. A similar function can be used to show that upper bisimilarity is not applicatively compatible.

It is possible to construct a TTS for which both lower bisimilarity and upper bisimilarity are extensional by modifying the definition of the set of states at a computation type. Using the approach taken in section 2.6, the states of a computation type $\mathsf{P}_\perp(\sigma)$ are either $\{\perp\}$ or a non-empty subset of the set of states of σ. Of course, there are states of $\mathscr{S}$, such as the function f above, that do not correspond to any states in this TTS.

4.4 The TTS $\mathscr{S}$ and Similarity

In this section we investigate the variants of similarity and mutual similarity for the TTS $\mathscr{S}$ defined in section 4.3. Extensionality and applicative compatibility fail for all variants of similarity and mutual similarity because there are non-trivial equivalence classes. We give examples at finite types to show that the general inclusions of lemma 4.2.4 can be strict, and investigate when meets and joins exist with respect to lower similarity and upper similarity. The examples are elementary but generic, and have analogues in the TTSs determined by the fragments of the programming language $\mathscr{L}$. This methodology has provided new examples to distinguish some of the variants of similarity, mutual similarity, and bisimilarity for $\mathscr{L}$, as well Lassen's [Las98b] non-deterministic λ-calculi.

The types $\mathsf{P}_\perp(\mathsf{unit})$ and $\mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{unit}))$ have P-orders of 1 and 2 respectively, and there are only finitely many states with one of these types in $\mathscr{S}$. With respect to either lower similarity or upper similarity, the equivalence classes of the states $\mathscr{S}(\mathsf{P}_\perp(\mathsf{unit}))$ form chains of length 2:

$$\{\perp\} \lesssim^{\mathscr{S}}_{\mathrm{LS}} \{\perp,\star\} \simeq^{\mathscr{S}}_{\mathrm{LS}} \{\star\}$$
$$\{\perp\} \simeq^{\mathscr{S}}_{\mathrm{US}} \{\perp,\star\} \lesssim^{\mathscr{S}}_{\mathrm{US}} \{\star\}$$

Similarly, with respect to either lower similarity or upper similarity, the equivalence classes of $\mathscr{S}(\mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{unit})))$ form chains of length 3.

Convex similarity and refinement similarity are more complex than lower similarity and upper similarity. The partial orders on their equivalence classes are depicted in figures 4.3 and 4.4 (only one representative from each equivalence class is given)[1].

The partial orders on the equivalence classes of $\mathscr{S}(\mathsf{P}_\perp(\mathsf{bool}))$ with respect to all of the variants of similarity are depicted in figure 4.5. Figure 4.6 contains the partial orders for $\mathscr{S}(\mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{bool})))$ with respect to lower similarity and upper similarity, and figure 4.7 contains the considerably more complex partial order for convex similarity equivalence classes.

The members of the equivalence classes in figures 4.5 and 4.6 have been chosen to illustrate lemmas 4.2.5(3)(a) and 4.2.6, i.e., lower similarity is the converse of upper similarity and refinement similarity when restricted to hereditarily total states, and there are representatives of the maximal equivalence classes that are hereditarily total and/or hereditarily deterministic.

Lemma 4.2.5(2)(c) ensures that mutual convex similarity and mutual refinement similarity coincide with convex bisimilarity on states of type $\mathsf{P}_\perp(\mathsf{unit})$ or $\mathsf{P}_\perp(\mathsf{bool})$, because those types have P-order 1. However, this property does not hold for the states of type $\mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{unit}))$ or $\mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{bool}))$. The non-trivial equivalence classes of states of type $\mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{unit}))$ with respect to convex similarity and refinement similarity are:

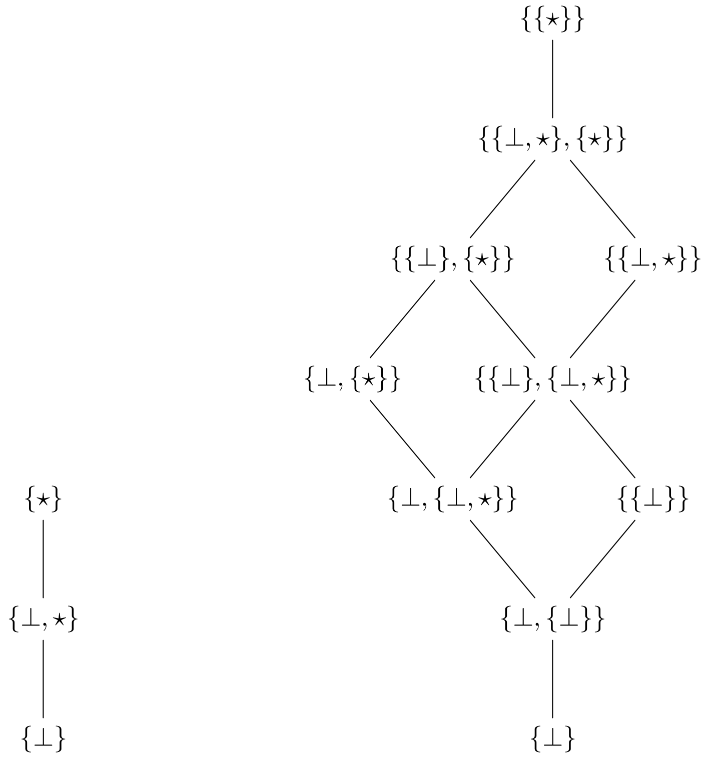
1. Mutual convex similarity:
   (a) $\{\perp,\{\perp,\star\}\} \simeq^{\mathscr{S}}_{\mathrm{CS}} \{\perp,\{\perp\},\{\perp,\star\}\}$
   (b) $\{\perp,\{\star\}\} \simeq^{\mathscr{S}}_{\mathrm{CS}} \{\perp,\{\perp\},\{\star\}\} \simeq^{\mathscr{S}}_{\mathrm{CS}} \{\perp,\{\perp,\star\},\{\star\}\} \simeq^{\mathscr{S}}_{\mathrm{CS}} \{\perp,\{\perp\},\{\perp,\star\},\{\star\}\}$
   (c) $\{\{\perp\},\{\star\}\} \simeq^{\mathscr{S}}_{\mathrm{CS}} \{\{\perp\},\{\perp,\star\},\{\star\}\}$
2. Mutual refinement similarity:
   (a) $\{\perp,\{\perp,\star\}\} \simeq^{\mathscr{S}}_{\mathrm{RS}} \{\perp,\{\perp,\star\},\{\perp\},\{\star\}\} \simeq^{\mathscr{S}}_{\mathrm{RS}} \{\perp,\{\perp,\star\},\{\perp\}\} \simeq^{\mathscr{S}}_{\mathrm{RS}} \{\perp,\{\perp,\star\},\{\star\}\}$
   (b) $\{\{\perp,\star\}\} \simeq^{\mathscr{S}}_{\mathrm{RS}} \{\{\perp,\star\},\{\perp\},\{\star\}\} \simeq^{\mathscr{S}}_{\mathrm{RS}} \{\{\perp,\star\},\{\perp\}\} \simeq^{\mathscr{S}}_{\mathrm{RS}} \{\{\perp,\star\},\{\star\}\}$

Example 4.4.1 gives more general examples to show that the variants of bisimilarity are strictly finer than the variants of mutual similarity.

Example 4.4.1 If $A,B \in \mathscr{S}(\sigma)$ are such that $A \lesssim^{\mathscr{S}}_{\mathrm{LS}} B$ and $B \not\lesssim^{\mathscr{S}}_{\mathrm{LS}} A$, then $\{A,B\} \simeq^{\mathscr{S}}_{\mathrm{LS}} \{B\}$ and $\{A,B\} \not\simeq^{\mathscr{S}}_{\mathrm{LB}} \{B\}$ at $\mathscr{S}(\mathsf{P}_\perp(\sigma))$. Similarly, if $A \lesssim^{\mathscr{S}}_{\mathrm{US}} B$ and $B \not\lesssim^{\mathscr{S}}_{\mathrm{US}} A$, then $\{A\} \simeq^{\mathscr{S}}_{\mathrm{US}} \{A,B\}$ and $\{A\} \not\simeq^{\mathscr{S}}_{\mathrm{UB}} \{A,B\}$. The assignment $A = \{\perp\}$ and $B = \{\star\}$ satisfies both conditions for $\sigma = \mathsf{P}_\perp(\mathsf{unit})$. It is also possible to give assignments that rely only on non-determinism instead of non-termination:

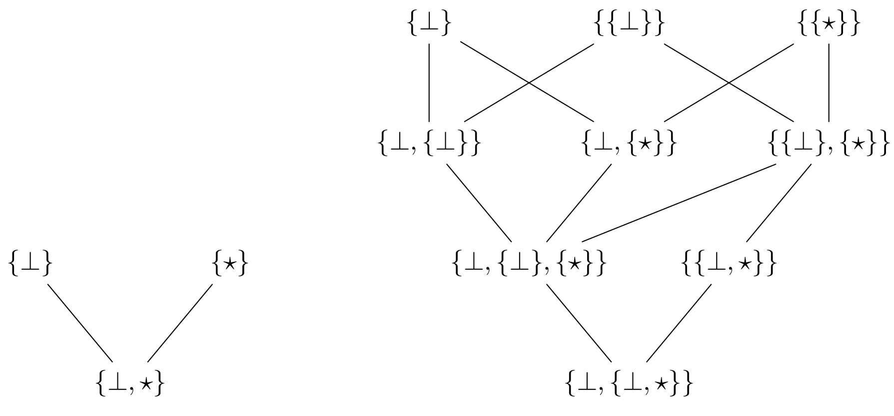
$$\{\{\mathit{ff}\},\{\mathit{ff},\mathit{tt}\}\} \simeq^{\mathscr{S}}_{\mathrm{LS}} \{\{\mathit{ff},\mathit{tt}\}\} \quad \text{and} \quad \{\{\mathit{ff}\},\{\mathit{ff},\mathit{tt}\}\} \not\simeq^{\mathscr{S}}_{\mathrm{LB}} \{\{\mathit{ff},\mathit{tt}\}\}$$
$$\{\{\mathit{ff},\mathit{tt}\}\} \simeq^{\mathscr{S}}_{\mathrm{US}} \{\{\mathit{ff},\mathit{tt}\},\{\mathit{ff}\}\} \quad \text{and} \quad \{\{\mathit{ff},\mathit{tt}\}\} \not\simeq^{\mathscr{S}}_{\mathrm{UB}} \{\{\mathit{ff},\mathit{tt}\},\{\mathit{ff}\}\}$$

[1] The diagrams in this section were generated using the Possum [Ear97] system.



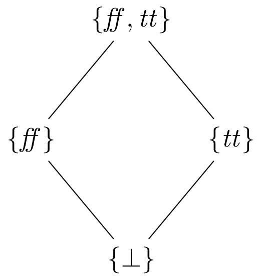
(a) $\mathscr{S}(\mathsf{P}_\bot(\mathsf{unit}))$ (b) $\mathscr{S}(\mathsf{P}_\bot(\mathsf{P}_\bot(\mathsf{unit})))$

Figure 4.3: Equivalence classes of $\mathscr{S}(\mathsf{P}_\bot(\mathsf{unit}))$ and $\mathscr{S}(\mathsf{P}_\bot(\mathsf{P}_\bot(\mathsf{unit})))$ w.r.t. $\lesssim^{\mathscr{S}}_{\mathrm{CS}}$

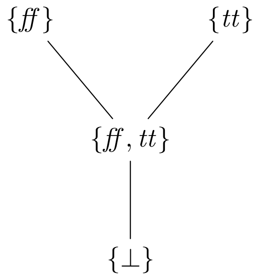


(a) $\mathscr{S}(\mathsf{P}_\bot(\mathsf{unit}))$ (b) $\mathscr{S}(\mathsf{P}_\bot(\mathsf{P}_\bot(\mathsf{unit})))$

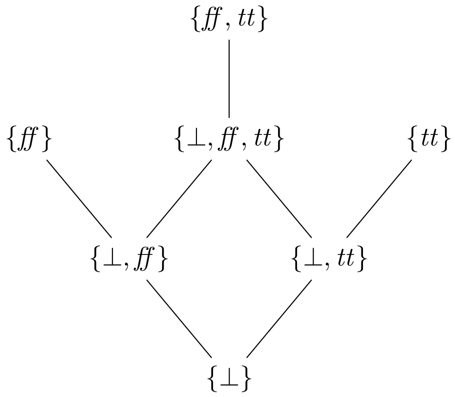
Figure 4.4: Equivalence classes of $\mathscr{S}(\mathsf{P}_\bot(\mathsf{unit}))$ and $\mathscr{S}(\mathsf{P}_\bot(\mathsf{P}_\bot(\mathsf{unit})))$ w.r.t. $\lesssim^{\mathscr{S}}_{\mathrm{RS}}$



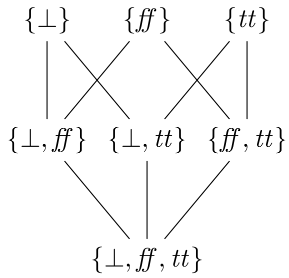
(a) $\lesssim^{\mathscr{S}}_{\mathrm{LS}}$



(b) $\lesssim^{\mathscr{S}}_{\mathrm{US}}$

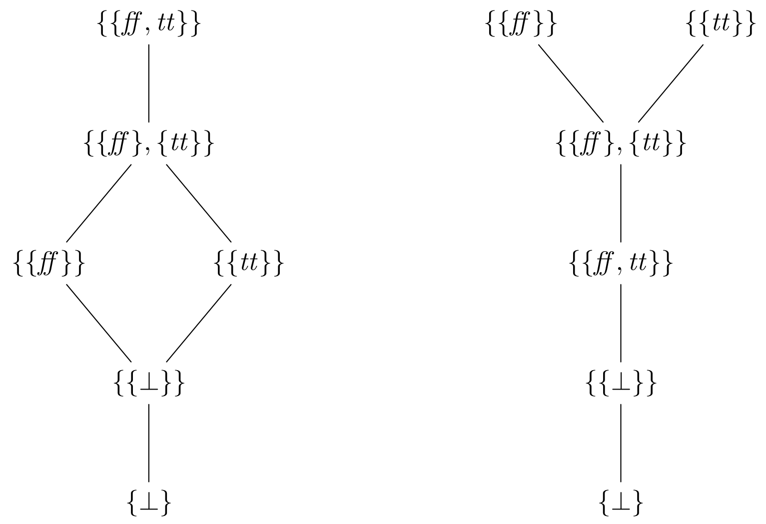


(c) $\lesssim^{\mathscr{S}}_{\mathrm{CS}}$



(d) $\lesssim^{\mathscr{S}}_{\mathrm{RS}}$

Figure 4.5: Equivalence classes of $\mathscr{S}(\mathsf{P}_\perp(\mathsf{bool}))$ w.r.t. $\lesssim^{\mathscr{S}}_{\mathrm{LS}}$, $\lesssim^{\mathscr{S}}_{\mathrm{US}}$, $\lesssim^{\mathscr{S}}_{\mathrm{CS}}$, and $\lesssim^{\mathscr{S}}_{\mathrm{RS}}$



(a) $\lesssim^{\mathscr{S}}_{\mathrm{LS}}$ (b) $\lesssim^{\mathscr{S}}_{\mathrm{US}}$

Figure 4.6: Equivalence classes of $\mathscr{S}(\mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{bool})))$ w.r.t. $\lesssim^{\mathscr{S}}_{\mathrm{LS}}$ and $\lesssim^{\mathscr{S}}_{\mathrm{US}}$

Mutual refinement similarity behaves in the same way as upper similarity. If $A \lesssim^{\mathscr{S}}_{\mathrm{RS}} B$ and $B \not\lesssim^{\mathscr{S}}_{\mathrm{RS}} A$, then $\{A\} \simeq^{\mathscr{S}}_{\mathrm{RS}} \{A,B\}$ and $\{A\} \not\simeq^{\mathscr{S}}_{\mathrm{CB}} \{A,B\}$. The assignment $A = \{ff, tt\}$ and $B = \{ff\}$ satisfies the conditions. Finally, if $A,B,C \in \mathscr{S}(\sigma)$ are such that $A \lesssim^{\mathscr{S}}_{\mathrm{CS}} B \lesssim^{\mathscr{S}}_{\mathrm{CS}} C$ and $C \not\lesssim^{\mathscr{S}}_{\mathrm{CS}} B \not\lesssim^{\mathscr{S}}_{\mathrm{CS}} A$, then:

$$\{A,B,C\} \simeq^{\mathscr{S}}_{\mathrm{CS}} \{A,C\} \qquad \text{and} \qquad \{A,B,C\} \not\simeq^{\mathscr{S}}_{\mathrm{CB}} \{A,C\}$$

A suitable assignment is $A = \{\perp\}$, $B = \{\perp, \star\}$, and $C = \{\star\}$.

The identifications made by mutual lower similarity and mutual upper similarity in example 4.4.1 can be extended to lower and upper sets. The lower and upper sets turn out to be useful for proving the existence of certain meets and joins, with respect to lower similarity and upper similarity, in $\mathscr{S}$ at computation types. First we show that the result of taking a lower set is related to the original set by mutual lower similarity, and similarly for upper sets and mutual upper similarity.

Lemma 4.4.2 For $A \in \mathscr{S}(\mathsf{P}_\perp(\sigma))$, extend the partial orders $\lesssim^{\mathscr{S}}_{\mathrm{LS}}$ and $\lesssim^{\mathscr{S}}_{\mathrm{US}}$ from $\mathscr{S}(\sigma)$ to $\mathscr{S}(\sigma)_\perp$ in the usual way, and then define $\downarrow_{\mathrm{LS}}(A), \uparrow_{\mathrm{US}}(A) \in \mathscr{S}(\mathsf{P}_\perp(\sigma))$ by:

$$\downarrow_{\mathrm{LS}}(A) \stackrel{\mathrm{def}}{=} \{C \in \mathscr{S}(\sigma)_\perp \mid \exists B \in A.\, C \lesssim^{\mathscr{S}}_{\mathrm{LS}} B\}$$
$$\uparrow_{\mathrm{US}}(A) \stackrel{\mathrm{def}}{=} \{C \in \mathscr{S}(\sigma)_\perp \mid \exists B \in A.\, B \lesssim^{\mathscr{S}}_{\mathrm{US}} C\}$$

Then $A \simeq^{\mathscr{S}}_{\mathrm{LS}} \downarrow_{\mathrm{LS}}(A)$ and $A \simeq^{\mathscr{S}}_{\mathrm{US}} \uparrow_{\mathrm{US}}(A)$.

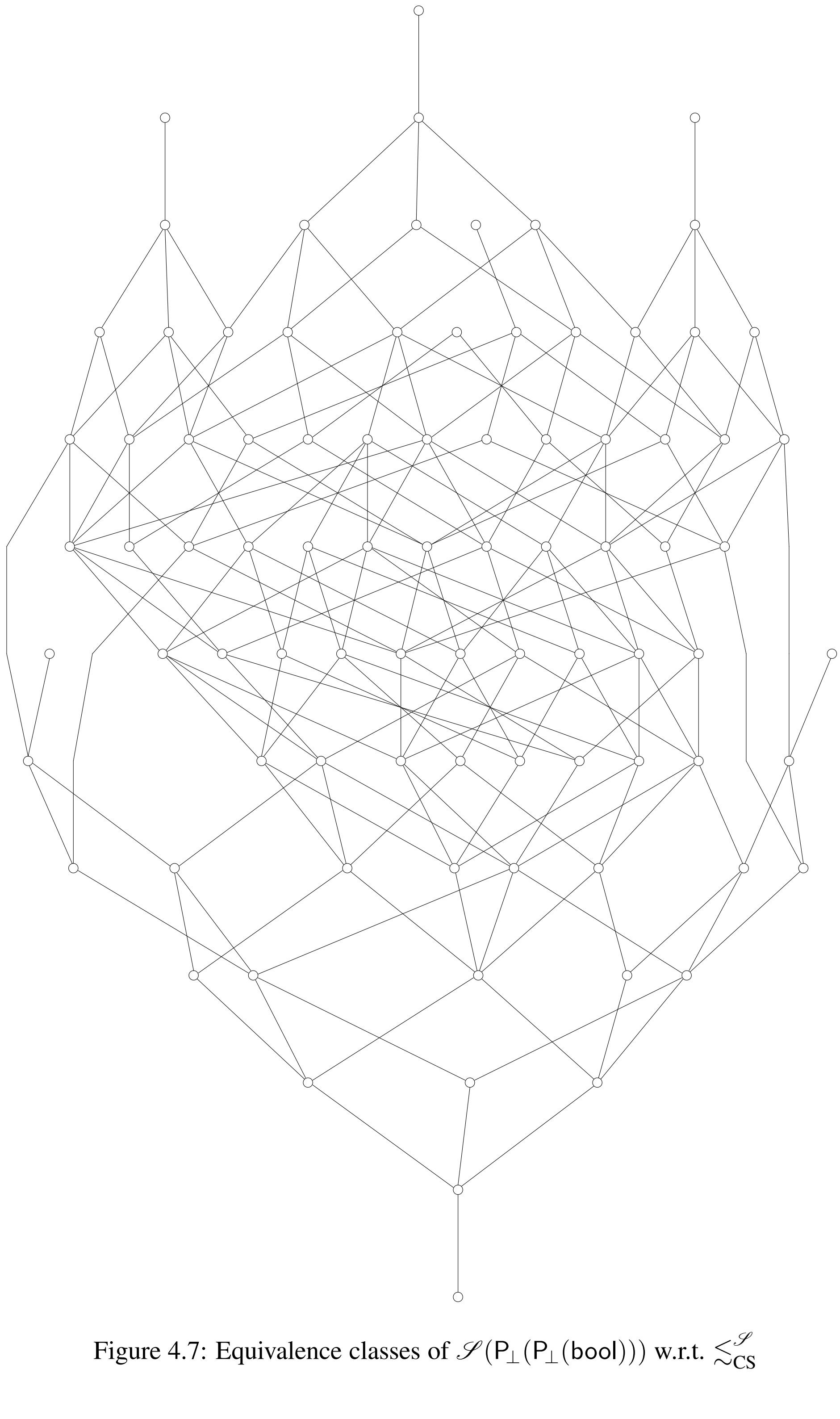


Figure 4.7: Equivalence classes of $\mathscr{S}(\mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{bool})))$ w.r.t. $\lesssim_{\mathrm{CS}}^{\mathscr{S}}$

Proof We have $A \lesssim^{\mathscr{S}}_{\mathrm{LS}} \downarrow_{\mathrm{LS}}(A)$ because lower similarity is reflexive, and if $C \in \downarrow_{\mathrm{LS}}(A)$ such that $C \neq \perp$, so there exists $B \neq \perp$ such that $C \lesssim^{\mathscr{S}}_{\mathrm{LS}} B \in A$, then $\downarrow_{\mathrm{LS}}(A) \lesssim^{\mathscr{S}}_{\mathrm{LS}} A$. Therefore $A \simeq^{\mathscr{S}}_{\mathrm{LS}} \downarrow_{\mathrm{LS}}(A)$. For upper similarity, first note that $\perp \in A$ if and only if $\perp \in \uparrow_{\mathrm{US}}(A)$. If $\perp \in A$, there is nothing to prove, so suppose that $\perp \notin A$. The reflexivity of upper similarity implies $\uparrow_{\mathrm{US}}(A) \lesssim^{\mathscr{S}}_{\mathrm{US}} A$. For the other direction, consider $C \in \uparrow_{\mathrm{US}}(A)$, so there exists $B \in A$ such that $B \lesssim^{\mathscr{S}}_{\mathrm{US}} C$, and we are done. Therefore $A \simeq^{\mathscr{S}}_{\mathrm{US}} \uparrow_{\mathrm{US}}(A)$. □

Lower similarity and upper similarity are partial orders on the images of the $\downarrow_{\mathrm{LS}}(\cdot)$ and $\uparrow_{\mathrm{US}}(\cdot)$ functions, and they agree with the inclusion partial order and its dual respectively. These partial orders are order-isomorphic to the partial orders induced by the preorders lower similarity and upper similarity (respectively) on $\mathscr{S}(\mathsf{P}_\perp(\sigma))$. To prove this it suffices to show that states of computation type are related by lower similarity or upper similarity if their images under $\downarrow_{\mathrm{LS}}(\cdot)$ or $\uparrow_{\mathrm{US}}(\cdot)$ are related by the inclusion order or its dual respectively.

Lemma 4.4.3 For all $A_1, A_2 \in \mathscr{S}(\mathsf{P}_\perp(\sigma))$:

1. $A_1 \lesssim^{\mathscr{S}}_{\mathrm{LS}} A_2$ if and only if $\downarrow_{\mathrm{LS}}(A_1) \subseteq \downarrow_{\mathrm{LS}}(A_2)$.

2. $A_1 \lesssim^{\mathscr{S}}_{\mathrm{US}} A_2$ if and only if $\uparrow_{\mathrm{US}}(A_1) \supseteq \uparrow_{\mathrm{US}}(A_2)$.

Proof

1. Suppose $A_1 \lesssim^{\mathscr{S}}_{\mathrm{LS}} A_2$ and $C_1 \in \downarrow_{\mathrm{LS}}(A_1)$. If $C_1 = \perp$, we are done because every lower set is non-empty and contains $\perp$. Otherwise $C_1 \neq \perp$ and there exists $B_1 \in A_1$ such that $C_1 \lesssim^{\mathscr{S}}_{\mathrm{LS}} B_1$, so $B_1 \neq \perp$. By $A_1 \lesssim^{\mathscr{S}}_{\mathrm{LS}} A_2$, there exists $B_2 \in A_2$ such that $B_1 \lesssim^{\mathscr{S}}_{\mathrm{LS}} B_2$ and $B_2 \neq \perp$. Hence, $C_1 \lesssim^{\mathscr{S}}_{\mathrm{LS}} B_2$, which implies that $C_1 \in \downarrow_{\mathrm{LS}}(A_2)$. For the other direction, suppose $\downarrow_{\mathrm{LS}}(A_1) \subseteq \downarrow_{\mathrm{LS}}(A_2)$ and $B_1 \in A_1$ such that $B_1 \neq \perp$. By reflexivity of lower similarity, $B_1 \in \downarrow_{\mathrm{LS}}(A_1) \subseteq \downarrow_{\mathrm{LS}}(A_2)$, so there exists $B_2 \in A_2$ such that $B_1 \lesssim^{\mathscr{S}}_{\mathrm{LS}} B_2$ and $B_2 \neq \perp$. Therefore $A_1 \lesssim^{\mathscr{S}}_{\mathrm{LS}} A_2$.

2. Suppose $A_1 \lesssim^{\mathscr{S}}_{\mathrm{US}} A_2$ and $C_2 \in \uparrow_{\mathrm{US}}(A_2)$, so there exists $B_2 \in A_2$ such that $B_2 \lesssim^{\mathscr{S}}_{\mathrm{US}} C_2$. If $\perp \in A_1$, then $C_2 \in \uparrow_{\mathrm{US}}(A_1)$, because $\perp \lesssim^{\mathscr{S}}_{\mathrm{US}} C_2$, and we are done. Otherwise, $\perp \notin A_1$. By $A_1 \lesssim^{\mathscr{S}}_{\mathrm{US}} A_2$, we have $\perp \notin A_2$. Consequently, there exists $B_1 \in A_1$ such that $B_1 \lesssim^{\mathscr{S}}_{\mathrm{US}} B_2$ and $B_1 \neq \perp$, so $B_1 \lesssim^{\mathscr{S}}_{\mathrm{US}} C_2$. Therefore $C_2 \in \uparrow_{\mathrm{US}}(A_1)$. For the other direction, suppose $\uparrow_{\mathrm{US}}(A_1) \supseteq \uparrow_{\mathrm{US}}(A_2)$ and $\perp \notin A_1$. By $\uparrow_{\mathrm{US}}(A_1) \supseteq \uparrow_{\mathrm{US}}(A_2)$, we have $\perp \notin A_2$. Now consider $B_2 \in A_2$ such that $B_2 \neq \perp$, so $B_2 \in \uparrow_{\mathrm{US}}(A_2) \subseteq \uparrow_{\mathrm{US}}(A_1)$. Hence, there exists $B_1 \in A_1$ such that $B_1 \lesssim^{\mathscr{S}}_{\mathrm{US}} B_2$ and $B_1 \neq \perp$. Therefore $A_1 \lesssim^{\mathscr{S}}_{\mathrm{US}} A_2$. □

Results on the existence of meets and joins at computation types with respect to lower similarity and upper similarity are obtained via the inclusion partial order and its dual on the images of the $\downarrow_{\mathrm{LS}}(\cdot)$ and $\uparrow_{\mathrm{US}}(\cdot)$ functions.

Proposition 4.4.4 For all types σ:

1. The partial order induced by the preorder $\langle \mathscr{S}(\mathsf{P}_\perp(\sigma)), \lesssim^{\mathscr{S}}_{\mathrm{LS}}\rangle$ is a complete lattice.
2. The partial order induced by the preorder $\langle \mathscr{S}(\mathsf{P}_\perp(\sigma)), \lesssim^{\mathscr{S}}_{\mathrm{US}}\rangle$ has meets of all non-empty subsets and joins of all subsets with an upper bound.

Proof We show that the partial orders:

$$\langle\{\downarrow_{\mathrm{LS}}(A) \mid A \in \mathscr{S}(\mathsf{P}_\perp(\sigma))\}, \subseteq\rangle \qquad \text{and} \qquad \langle\{\uparrow_{\mathrm{US}}(A) \mid A \in \mathscr{S}(\mathsf{P}_\perp(\sigma))\}, \supseteq\rangle$$

satisfy the same properties.

1. The meet of the empty set is $\mathscr{S}(\mathsf{P}_\perp(\sigma))$ and the join of the empty set is $\{\perp\}$, both of which map to themselves by $\downarrow_{\mathrm{LS}}(\cdot)$ and so are in the image of that function. Now consider $X \subseteq_{\mathrm{ne}} \mathscr{S}(\mathsf{P}_\perp(\sigma))$. We claim that the meet and join of $\{\downarrow_{\mathrm{LS}}(A) \mid A \in X\}$ are $\bigcap\{\downarrow_{\mathrm{LS}}(A) \mid A \in X\}$ and $\bigcup\{\downarrow_{\mathrm{LS}}(A) \mid A \in X\}$ respectively. Both sets are members of $\mathscr{S}(\mathsf{P}_\perp(\sigma))$, and in the image of $\downarrow_{\mathrm{LS}}(\cdot)$.They are the meet and join because intersection and union are the meet and join operations respectively on the complete lattice $\langle \mathcal{P}(\mathscr{S}(\sigma)_\perp), \subseteq\rangle$.
2. The meet of the empty set need not exist in general. The join of the empty set is $\mathscr{S}(\mathsf{P}_\perp(\sigma))$, which is equal to $\uparrow_{\mathrm{US}}(\{\perp\})$. Now consider $X \subseteq_{\mathrm{ne}} \mathscr{S}(\mathsf{P}_\perp(\sigma))$. Using the fact that union and intersection are the meet and join operations respectively on the complete lattice $\langle \mathcal{P}(\mathscr{S}(\sigma)_\perp), \supseteq\rangle$, we need only show that $\bigcup\{\uparrow_{\mathrm{US}}(A) \mid A \in X\}$ is always a member of $\{\uparrow_{\mathrm{US}}(A) \mid A \in \mathscr{S}(\mathsf{P}_\perp(\sigma))\}$, and that $\bigcap\{\uparrow_{\mathrm{US}}(A) \mid A \in X\}$ is a member of $\{\uparrow_{\mathrm{US}}(A) \mid A \in \mathscr{S}(\mathsf{P}_\perp(\sigma))\}$ whenever X is bounded above. The first case is straightforward. For the second case, suppose that X is bounded above, so there exists $B \in \mathscr{S}(\mathsf{P}_\perp(\sigma))$ such that, for all $A \in X$, $\uparrow_{\mathrm{US}}(A) \supseteq \uparrow_{\mathrm{US}}(B)$. In this case, $\bigcap\{\uparrow_{\mathrm{US}}(A) \mid A \in X\} \supseteq \uparrow_{\mathrm{US}}(B)$, so the candidate for the join is a non-empty set, and we have $\bigcap\{\uparrow_{\mathrm{US}}(A) \mid A \in X\} \in \{\uparrow_{\mathrm{US}}(A) \mid A \in \mathscr{S}(\mathsf{P}_\perp(\sigma))\}$.

□

Example 4.4.5 shows that the intersection of the lower and upper variants of similarity, mutual similarity, and bisimilarity are strictly coarser than their convex variants. This completes the series of examples that demonstrate that the inclusions in figure 4.2 are strict in $\mathscr{S}$.

Example 4.4.5 For $A \in \mathscr{S}(\sigma)$, consider the sets $\{\perp, \{A\}\}$ and $\{\perp, \{\perp, A\}\}$ that are states of type $\mathsf{P}_\perp(\mathsf{P}_\perp(\sigma))$ in $\mathscr{S}$. They are related by each of:

$$(\simeq^{\mathscr{S}}_{\mathrm{LB}} \cap \simeq^{\mathscr{S}}_{\mathrm{UB}}) \subseteq (\simeq^{\mathscr{S}}_{\mathrm{LS}} \cap \simeq^{\mathscr{S}}_{\mathrm{US}}) \subseteq (\lesssim^{\mathscr{S}}_{\mathrm{LS}} \cap \lesssim^{\mathscr{S}}_{\mathrm{US}}) \qquad \text{and} \qquad (\lesssim^{\mathscr{S}}_{\mathrm{LS}})^{\mathrm{op}} \cap \lesssim^{\mathscr{S}}_{\mathrm{US}}$$

but not by any of:

$$\simeq^{\mathscr{S}}_{\mathrm{CB}} \subseteq \simeq^{\mathscr{S}}_{\mathrm{CS}} \subseteq \lesssim^{\mathscr{S}}_{\mathrm{CS}} \qquad \text{or} \qquad \simeq^{\mathscr{S}}_{\mathrm{RS}} \subseteq \lesssim^{\mathscr{S}}_{\mathrm{RS}}$$

We conclude with examples to show that there are no other general inclusions that could be added to figure 4.2.

Example 4.4.6 In figure 4.8 each row contains examples that demonstrate that two relations are incomparable. For example, the first row shows that mutual convex similarity and lower bisimilarity are incomparable. The states related by mutual convex similarity in the first column are not related by lower bisimilarity, and the states related by lower bisimilarity in the second column are not related by mutual convex similarity.

$\{\bot,\{\mathit{ff},\mathit{tt}\}\} \simeq^{\mathscr{S}}_{\mathrm{CS}} \{\bot,\{\bot,\mathit{ff}\},\{\mathit{ff},\mathit{tt}\}\}$	$\{\star\} \simeq^{\mathscr{S}}_{\mathrm{LB}} \{\bot,\star\}$
$\{\{\bot\},\{\{\star\}\}\} \simeq^{\mathscr{S}}_{\mathrm{CS}} \{\{\bot\},\{\{\bot\},\{\star\}\},\{\{\star\}\}\}$	$\{\bot,\star\} \simeq^{\mathscr{S}}_{\mathrm{UB}} \{\bot\}$
$\{\bot\} \lesssim^{\mathscr{S}}_{\mathrm{CS}} \{\star\}$	$\{\star\} \simeq^{\mathscr{S}}_{\mathrm{LS}} \{\bot,\star\}$
$\{\bot\} \lesssim^{\mathscr{S}}_{\mathrm{CS}} \{\star\}$	$\{\bot,\star\} \simeq^{\mathscr{S}}_{\mathrm{US}} \{\bot\}$
$\{\star\} \simeq^{\mathscr{S}}_{\mathrm{LB}} \{\bot,\star\}$	$\{\bot,\star\} \simeq^{\mathscr{S}}_{\mathrm{UB}} \{\bot\}$
$\{\star\} \simeq^{\mathscr{S}}_{\mathrm{LS}} \{\bot,\star\}$	$\{\bot,\star\} \simeq^{\mathscr{S}}_{\mathrm{US}} \{\bot\}$
$\{\star\} \lesssim^{\mathscr{S}}_{\mathrm{LS}} \{\bot,\star\}$	$\{\bot,\star\} \lesssim^{\mathscr{S}}_{\mathrm{US}} \{\bot\}$
$\{\{\mathit{ff},\mathit{tt}\}\} \simeq^{\mathscr{S}}_{\mathrm{RS}} \{\{\mathit{ff},\mathit{tt}\},\{\mathit{ff}\}\}$	$\{\star\} \simeq^{\mathscr{S}}_{\mathrm{LB}} \{\bot,\star\}$
$\{\{\mathit{ff},\mathit{tt}\}\} \simeq^{\mathscr{S}}_{\mathrm{RS}} \{\{\mathit{ff},\mathit{tt}\},\{\mathit{ff}\}\}$	$\{\bot,\star\} \simeq^{\mathscr{S}}_{\mathrm{UB}} \{\bot\}$
$\{\bot,\{\bot,\star\},\{\star\}\} \simeq^{\mathscr{S}}_{\mathrm{RS}} \{\bot,\{\bot\},\{\bot,\star\}\}$	$\{\bot,\{\star\}\} \simeq^{\mathscr{S}}_{\mathrm{CS}} \{\bot,\{\bot\},\{\star\}\}$
$\{\bot,\{\bot,\star\},\{\star\}\} \lesssim^{\mathscr{S}}_{\mathrm{RS}} \{\bot,\{\bot\},\{\bot,\star\}\}$	$\{\bot,\{\star\}\} \lesssim^{\mathscr{S}}_{\mathrm{CS}} \{\bot,\{\bot\},\{\star\}\}$

Figure 4.8: Incomparable relations

4.5 A Category of TTSs

We investigate the relationship between TTSs and their associated convex bisimilarity relations via a category *PR* of maps between TTSs. Maps are functions between the sets of states that preserve and reflect the structure of TTSs, including the type of a state, labelled transitions, and may divergence. The existence of a map $\phi \in PR(\mathscr{T},\mathscr{U})$ between TTSs $\mathscr{T}$ and $\mathscr{U}$ means that the relation induced on states of $\mathscr{T}$ by pulling convex bisimilarity in $\mathscr{U}$ back along ϕ is finer than convex bisimilarity in $\mathscr{T}$. In particular, there is a map from a TTS to its *extensional collapse* whenever convex bisimilarity is applicatively compatible with respect to the TTS. A map can also arise when $\mathscr{U}$ contains "more" states than $\mathscr{T}$. This leads to a large family of inclusion maps derived by (carefully) removing states from TTSs. The TTSs determined by fragments of the programming language $\mathscr{L}$ in chapter 5 have inclusion maps from smaller fragments to larger fragments.

We also show that there is at least one map to the TTS $\mathscr{S}$ from a TTS where convex bisimilarity is applicatively compatible and an additional constraint is satisfied. The constraint is satisfied by every TTS obtained from a fragment of $\mathscr{L}$.

The conditions upon functions that form the *PR*-maps are similar to, but more restrictive than, those imposed on logical relations. They must be functions instead of relations, and the relationship between states of, for example, natural number type are restricted because it is a coproduct type not a base type. To illustrate this, suppose $\phi : \mathscr{T} \to \mathscr{U}$ and there are states $s \in \mathscr{T}(\mathtt{nat})$, $t \in \mathscr{T}(\mathtt{unit})$, a natural number $m \in \omega$, and a labelled transition $s \xrightarrow{m}^{\mathscr{T}} t$. Then it must be the case that $\phi(s) \in \mathscr{U}(\mathtt{nat})$, $\phi(t) \in \mathscr{U}(\mathtt{unit})$, and there is a labelled transition $\phi(s) \xrightarrow{m}^{\mathscr{U}} \phi(t)$.

Definition 4.5.1 Let $\mathscr{T}$ and $\mathscr{U}$ be TTSs with labelled transition relations $\to^{\mathscr{T}}$ and $\to^{\mathscr{U}}$ respectively, and $\phi : \mathscr{T} \to \mathscr{U}$ a function from the states of $\mathscr{T}$ to the states of $\mathscr{U}$. Define a map ϕ from the labels of $\mathscr{T}$ to those of $\mathscr{U}$ by:

$$\begin{aligned} \phi(n) &\stackrel{\text{def}}{=} n && (n \in \omega) \\ \phi(@t\,) &\stackrel{\text{def}}{=} @(\phi(t)) && (t \in \mathscr{T}) \\ \phi(\Downarrow) &\stackrel{\text{def}}{=} \Downarrow \end{aligned}$$

The function ϕ ***preserves and reflects labelled transitions and may divergence*** if it satisfies all of the following conditions:

1. The type of a state is preserved by ϕ, i.e., if $s \in \mathscr{T}(\sigma)$, then $\phi(s) \in \mathscr{U}(\sigma)$.
2. For all states $s, t \in \mathscr{T}$ and labels a, if $s \xrightarrow{a}^{\mathscr{T}} t$, then $\phi(s) \xrightarrow{\phi(a)}^{\mathscr{U}} \phi(t)$.
3. For all states $s \in \mathscr{T}$, $u \in \mathscr{U}$, and labels a, if $\phi(s) \xrightarrow{\phi(a)}^{\mathscr{U}} u$, then there exists a state $t \in \mathscr{T}$ such that $s \xrightarrow{a}^{\mathscr{T}} t$ and $\phi(t) = u$.
4. May divergence is preserved and reflected by ϕ, i.e., for all states $s \in \mathscr{T}$ of computation type, $s \Uparrow^{\text{may}}$ if and only if $\phi(s) \Uparrow^{\text{may}}$.

It is straightforward to show that functions that preserve and reflect labelled transitions and may divergence form a category. However, it is worth noting that composition is not well-defined if definition 4.5.1(3) is weakened to require only that $\phi(t)$ and u are related by convex bisimilarity.

Definition 4.5.2 The objects of the category *PR* are TTSs. For TTSs $\mathscr{T}$ and $\mathscr{U}$, the members of the homset $PR(\mathscr{T}, \mathscr{U})$ are functions from the states of $\mathscr{T}$ to the states of $\mathscr{U}$ that preserve and reflect labelled transitions and may divergence.

Maps $\phi \in PR(\mathscr{T}, \mathscr{U})$ have the following important property. If the images under ϕ of states of $\mathscr{T}$ are related by convex bisimilarity in $\mathscr{U}$, then the states of $\mathscr{T}$ must also be related by convex bisimilarity in $\mathscr{T}$. Intuitively, this holds because the existence of a map $\phi \in PR(\mathscr{T}, \mathscr{U})$ forces $\mathscr{U}$ to have at least as many states as $\mathscr{T}$, and so $\mathscr{U}$ has at least as much discriminative power for states of function type as $\mathscr{T}$ does. In fact, $\mathscr{U}$ may have fewer states than $\mathscr{T}$ in some cases, but this only arises when states of $\mathscr{T}$ that are in the same convex bisimilarity equivalence class are identified in $\mathscr{U}$.

Proposition 4.5.3 Consider $\phi \in PR(\mathscr{T},\mathscr{U})$. For states $s,t \in \mathscr{T}$, if $\phi(s) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t)$, then $s \simeq^{\mathscr{T}}_{\mathrm{CB}} t$.

Proof By coinduction. Define $\mathcal{R} \subseteq \mathscr{T} \times \mathscr{T}$ by:

$$\mathcal{R} \stackrel{\text{def}}{=} \{\langle s,t\rangle \in \mathscr{T} \times \mathscr{T} \mid \phi(s) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t)\}$$

The cases for states of a value type are straightforward because every state of a value type has at most one transition for each label. For example, if $s,t \in \mathscr{T}(\sigma \to \tau)$, $u \in \mathscr{T}(\sigma)$, and $\phi(s) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t)$, then $s \xrightarrow{@u}^{\mathscr{T}} s@u$ and $t \xrightarrow{@u}^{\mathscr{T}} t@u$. By the labelled transition preservation property of ϕ, we have $\phi(s) \xrightarrow{@\phi(u)}^{\mathscr{U}} \phi(s@u)$ and $\phi(t) \xrightarrow{@\phi(u)}^{\mathscr{U}} \phi(t@u)$. However, $\phi(s)$ and $\phi(t)$ each have one transition labelled with $@\phi(u)$ because $\mathscr{U}$ is a TTS, so $\phi(s@u) = \phi(s)@\phi(u)$ and $\phi(t@u) = \phi(t)@\phi(u)$. Then $\phi(s@u) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t@u)$, because $\phi(s) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t)$. Therefore $\langle s@u, t@u\rangle \in \mathcal{R}$, which completes the case for states of function type. For computation types, consider states $s_1,t_1 \in \mathscr{T}(\mathsf{P}_\perp(\sigma))$ such that $\phi(s_1) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t_1)$. Combining the fact that ϕ preserves and reflects divergence with $\phi(s_1) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t_1)$ gives $s_1 \Uparrow^{\mathrm{may}}$ if and only if $t_1 \Uparrow^{\mathrm{may}}$. Now consider any state $s_2 \in \mathscr{T}(\sigma)$ such that $s_1 \xrightarrow{\Downarrow}^{\mathscr{T}} s_2$, so $\phi(s_1) \xrightarrow{\Downarrow}^{\mathscr{U}} \phi(s_2)$. By $\phi(s_1) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t_1)$, there exists $u \in \mathscr{U}(\sigma)$ such that $\phi(t_1) \xrightarrow{\Downarrow}^{\mathscr{U}} u$ and $\phi(s_2) \simeq^{\mathscr{U}}_{\mathrm{CB}} u$. By the transition reflecting property of ϕ, there exists $t_2 \in \mathscr{T}(\sigma)$ such that $t_1 \xrightarrow{\Downarrow}^{\mathscr{T}} t_2$ and $\phi(t_2) = u$. Therefore $\langle s_2,t_2\rangle \in \mathcal{R}$, because $\phi(s_2) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t_2)$. The other direction is similar. $\square$

There is a partial converse to proposition 4.5.3. It is partial because $\mathscr{U}$ may have states of type σ that are not in the image of ϕ. Such states have no effect on $\mathscr{T}(\sigma \to \tau)$, but can be used to distinguish their images in $\mathscr{U}(\sigma \to \tau)$.

Lemma 4.5.4 Consider $\phi \in PR(\mathscr{T},\mathscr{U})$. Let A be the set of types for which convex bisimilarity in $\mathscr{U}$ pulled back in ϕ coincides with convex bisimilarity in $\mathscr{T}$:

$$A \stackrel{\text{def}}{=} \{\sigma \mid \forall s,t \in \mathscr{T}(\sigma).\, s \simeq^{\mathscr{T}}_{\mathrm{CB}} t \Longrightarrow \phi(s) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t)\}$$

Then:

1. A is closed under the formation of coproducts, products, and computation types.

2. For all types σ and τ, if every state in $\mathscr{U}(\sigma)$ is in the image of ϕ, and $\tau \in A$, then $\sigma \to \tau \in A$.

Proof

1. We give the case for computation types. The cases for coproduct and product types are similar. Consider a type $\sigma \in A$ and states $s_1,t_1 \in \mathscr{T}(\mathsf{P}_\perp(\sigma))$ such that $s_1 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_1$. We want

to show that $\phi(s_1) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t_1)$. Using $s_1 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_1$ and the fact that ϕ preserves and reflects divergence, we see that $\phi(s_1) \Uparrow^{\mathrm{may}}$ if and only if $\phi(t_1) \Uparrow^{\mathrm{may}}$. Now consider $u \in \mathscr{U}(\sigma)$ such that $\phi(s_1) \xrightarrow{\Downarrow}^{\mathscr{U}} u$. By the transition reflecting property of ϕ, there exists $s_2 \in \mathscr{T}(\sigma)$ such that $s_1 \xrightarrow{\Downarrow}^{\mathscr{T}} s_2$ and $\phi(s_2) = u$. By $s_1 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_1$, there exists $t_2 \in \mathscr{T}(\sigma)$ such that $t_1 \xrightarrow{\Downarrow}^{\mathscr{T}} t_2$ and $s_2 \simeq^{\mathscr{T}}_{\mathrm{CB}} t_2$. It follows that $\phi(t_1) \xrightarrow{\Downarrow}^{\mathscr{U}} \phi(t_2)$, and $\phi(s_2) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t_2)$ because $\sigma \in A$. The other direction is similar. Therefore $\phi(s_1) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t_1)$.

2. Consider $s,t \in \mathscr{T}(\sigma \to \tau)$ such that $s \simeq^{\mathscr{T}}_{\mathrm{CB}} t$. We want to show that $\phi(s) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t)$. It suffices to show that, for all $u \in \mathscr{U}(\sigma)$, $\phi(s) @ u \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t) @ u$. By assumption, there exists a state $v \in \mathscr{T}(\sigma)$ such that $u = \phi(v)$. Now $s \xrightarrow{@v}^{\mathscr{T}} s@v$, so $\phi(s) \xrightarrow{@\phi(v)}^{\mathscr{U}} \phi(s@v)$, but $\phi(s)$ has a unique transition labelled with $@\phi(v)$, so $\phi(s) @ \phi(v) = \phi(s@v)$. Similarly, $\phi(t) @ \phi(v) = \phi(t @ v)$. In addition, $s @ v \simeq^{\mathscr{T}}_{\mathrm{CB}} t @ v$, because $s \simeq^{\mathscr{T}}_{\mathrm{CB}} t$. With $\tau \in A$, we get $\phi(s) @ u = \phi(s @ v) \simeq^{\mathscr{U}}_{\mathrm{CB}} \phi(t @ v) = \phi(t) @ u$ as required. □

We now define the *extensional collapse* of a TTS. The states of the extensional collapse are equivalence classes of states with respect to convex bisimilarity. Convex bisimilarity must be applicatively compatible with respect to the TTS so that the application transitions of the extensional collapse are well-defined, for the same reason that it is not always possible to define the quotients of a qATS or a qNATS.

Definition 4.5.5 Let $\mathscr{T}$ be a TTS such that convex bisimilarity is applicatively compatible. The ***extensional collapse*** of $\mathscr{T}$ is a TTS denoted $Ext(\mathscr{T})$. The states of $Ext(\mathscr{T})$ are defined, for each type σ, by:

$$Ext(\mathscr{T})(\sigma) \stackrel{\text{def}}{=} \{\{t \in \mathscr{T}(\sigma) \mid s \simeq^{\mathscr{T}}_{\mathrm{CB}} t\} \mid s \in \mathscr{T}(\sigma)\}$$

A state $A \in Ext(\mathscr{T})$ may diverge if and only if, for all $s \in A$, $s \Uparrow^{\mathrm{may}}$. To define the labelled transition relation, consider states $A,B \in Ext(\mathscr{T})$. If A has a coproduct, product, or computation type, then $A \xrightarrow{a}^{Ext(\mathscr{T})} B$ if and only if, for all $s \in A$, there exists $t \in B$ such that $s \xrightarrow{a}^{\mathscr{T}} t$. If A has a function type, then $A \xrightarrow{@C}^{Ext(\mathscr{T})} B$ if and only if, for all $s \in A$ and $u \in C$, $s@u \in B$.

Convex bisimilarity is extensional on the extensional collapse of a TTS and there is a map from the extensional collapse to the original TTS. In addition, every state of the extensional collapse is in the image of the map, so, by lemma 4.5.4, the images of states related by convex bisimilarity in the original TTS are related by convex bisimilarity in the extensional collapse, i.e., they are equal.

Proposition 4.5.6 Let $\mathscr{T}$ be a TTS such that convex bisimilarity is applicatively compatible. Then $Ext(\mathscr{T})$ is a TTS upon which convex bisimilarity is extensional. In addition, there is a map $\phi \in PR(\mathscr{T}, Ext(\mathscr{T}))$ such that states of $\mathscr{T}$ are related by convex bisimilarity in $\mathscr{T}$ if and only if their images under ϕ are related by convex bisimilarity in $Ext(\mathscr{T})$.

Proof It is straightforward to show that $Ext(\mathscr{T})$ is a TTS. Applicative compatibility of convex bisimilarity is necessary for the existence of transitions from states of $Ext(\mathscr{T})$ with function type. For $A \in Ext(\mathscr{T})(\sigma \to \tau)$ and $C \in Ext(\mathscr{T})(\sigma)$, we must show that there exists $B \in Ext(\mathscr{T})(\tau)$ such that $A \xrightarrow{@C}{}^{Ext(\mathscr{T})} B$. Now A and C are non-empty, so there are states $s_1 \in A$ and $u_1 \in C$. For all $s_2 \in A$ and $u_2 \in C$, applicative compatibility implies that $s_1 @ u_1 \simeq^{\mathscr{T}}_{\mathrm{CB}} s_2 @ u_2$. Therefore we take B to be the convex bisimilarity equivalence class of $s_1 @ u_1$, and we have that $Ext(\mathscr{T})$ is a TTS. Now define the function $\phi : \mathscr{T} \to Ext(\mathscr{T})$, for $s \in \mathscr{T}(\sigma)$, by:

$$\phi(s) \stackrel{\text{def}}{=} \{t \in \mathscr{T}(\sigma) \mid s \simeq^{\mathscr{T}}_{\mathrm{CB}} t\}$$

Clearly every state of $Ext(\mathscr{T})$ is in the image of ϕ. We first show that $\phi \in PR(\mathscr{T}, Ext(\mathscr{T}))$. The function ϕ preserves the type of states, and preserves and reflects may divergence. Preservation and reflection of labelled transitions by ϕ is straightforward, making use of the applicative compatibility of convex bisimilarity on $\mathscr{T}$ and the fact that every state of $Ext(\mathscr{T})$ is the image of a state of $\mathscr{T}$. Combining the latter fact with proposition 4.5.3 and lemma 4.5.4 implies that, for all types σ and states $s, t \in \mathscr{T}(\sigma)$, $s \simeq^{\mathscr{T}}_{\mathrm{CB}} t$ if and only if $\phi(s) \simeq^{Ext(\mathscr{T})}_{\mathrm{CB}} \phi(t)$. Moreover, $s \simeq^{\mathscr{T}}_{\mathrm{CB}} t$ is equivalent to $\phi(s) = \phi(t)$, so convex bisimilarity is extensional in $Ext(\mathscr{T})$. □

In general, there need not be a map from $Ext(\mathscr{T})$ to $\mathscr{T}$ because definition 4.5.1(3) requires that $\phi(t) = u$ rather than $\phi(t) \simeq^{\mathscr{U}}_{\mathrm{CB}} u$.

Example 4.5.7 We define a TTS $\mathscr{T}$ such that convex bisimilarity is applicatively compatible, but for which there are no maps from $Ext(\mathscr{T})$ to $\mathscr{T}$. The states of type unit are $\star_1$ and $\star_2$. The unique state of type $\mathsf{P}_\perp(\mathsf{unit})$ is s, and there are no other states. The state s has two transitions, $s \xrightarrow{\Downarrow}{}^{\mathscr{T}} \star_1$ and $s \xrightarrow{\Downarrow}{}^{\mathscr{T}} \star_2$. Convex bisimilarity is trivially applicatively compatible with respect to the TTS $\mathscr{T}$ because there are no states of function type. The extensional collapse $Ext(\mathscr{T})$ has one state $\{\star_1, \star_2\}$ of type unit, one state $\{s\}$ of type $\mathsf{P}_\perp(\mathsf{unit})$, and a single transition $\{s\} \xrightarrow{\Downarrow}{}^{Ext(\mathscr{T})} \{\star_1, \star_2\}$. For a contradiction, suppose that there is a map $\phi \in PR(Ext(\mathscr{T}), \mathscr{T})$ and, without loss of generality, that $\phi(\{\star_1, \star_2\}) = \star_1$. Such a map does not reflect labelled transitions, because $\phi(\{s\}) = s \xrightarrow{\phi(\Downarrow)}{}^{\mathscr{T}} \star_2$ and $\star_2 \notin Im(\phi)$. Therefore the homset $PR(Ext(\mathscr{T}), \mathscr{T})$ is empty.

We now consider a *restriction* operation upon TTSs. The restriction operation is motivated by the family of TTSs determined by the fragments of the programming language $\mathscr{L}$ (see chapter 5), and the mismatch between applicative similarity for PCF and its directed-complete partial order model caused by the non-definability of parallel elements (see [Gun92]). A TTS is obtained from a TTS $\mathscr{T}$ by restricting to a set of states $X \subseteq \mathscr{T}$ that is closed under labelled transitions of $\mathscr{T}$, with the exception of function application transitions that are labelled with elements not in X. Convex bisimilarity for $\mathscr{T}$ is always finer, sometimes strictly finer, than convex bisimilarity upon the TTS obtained by restriction, and we formalise this by showing that the inclusion function from the states of the restricted TTS is a map.

Definition 4.5.8 Let $\mathscr{T}$ be a TTS and $X \subseteq \mathscr{T}$ a set of states such that, for all states $s \in X$, $t \in \mathscr{T}$, and labels a, such that $s \xrightarrow{a}{}^{\mathscr{T}} t$, either $t \in X$, or there exists a state $u \in \mathscr{T} \setminus X$ such that $a = @u$.

The ***restriction*** of $\mathscr{T}$ to X is a TTS denoted $\mathscr{T} \restriction X$. The states of $\mathscr{T} \restriction X$ are defined, for each type σ, by:

$$(\mathscr{T} \restriction X)(\sigma) \stackrel{\text{def}}{=} \mathscr{T}(\sigma) \cap X$$

The may divergence predicate for $\mathscr{T} \restriction X$ is the restriction to X of the may divergence predicate for $\mathscr{T}$. For states $s, t \in \mathscr{T} \restriction X$, and a label a, there is a labelled transition $s \xrightarrow{a}^{\mathscr{T} \restriction X} t$ if and only if $s \xrightarrow{a}^{\mathscr{T}} t$ and there does not exist $u \in \mathscr{T} \setminus X$ such that $a = @u$.

Proposition 4.5.9 Let $\mathscr{T}$ be a TTS and $X \subseteq \mathscr{T}$ a set of states such that, for all states $s \in X$, $t \in \mathscr{T}$, and labels a, such that $s \xrightarrow{a}^{\mathscr{T}} t$, either $t \in X$, or there exists a state $u \in \mathscr{T} \setminus X$ such that $a = @u$. Then $\mathscr{T} \restriction X$ is a TTS and the inclusion function ϕ from the states of $\mathscr{T} \restriction X$ to the states of $\mathscr{T}$ is a map $\phi \in PR(\mathscr{T} \restriction X, \mathscr{T})$.

Proof For $\mathscr{T} \restriction X$ to be a TTS it suffices to show that $\mathscr{T} \restriction X$ has enough transitions to satisfy definition 4.1.1(5). For a state $s \in \mathscr{T} \restriction X$ of coproduct, product, or computation type, the closure conditions on X ensure that if there is a state $t \in \mathscr{T}$ and a label a such that $s \xrightarrow{a}^{\mathscr{T}} t$, then $s \xrightarrow{a}^{\mathscr{T} \restriction X} t$. If $s \in (\mathscr{T} \restriction X)(\sigma \to \tau)$ and $u \in (\mathscr{T} \restriction X)(\sigma)$, then there exists $t \in \mathscr{T}(\tau)$ such that $s \xrightarrow{@u}^{\mathscr{T}} t$. The closure conditions on X imply that $t \in X$ and $s \xrightarrow{@u}^{\mathscr{T} \restriction X} t$, because $u \in X$. Therefore $\mathscr{T} \restriction X$ is a TTS. The inclusion function ϕ is easily seen to preserve types, preserve and reflect may divergence, and preserve labelled transitions. The closure conditions on X also imply that labelled transitions are reflected by ϕ. For example, if $s \in (\mathscr{T} \restriction X)(\sigma \to \tau)$, $u \in (\mathscr{T} \restriction X)(\sigma)$, and $t \in \mathscr{T}(\tau)$ such that $s \xrightarrow{@u}^{\mathscr{T}} t$, then $t \in X$ because $u \in X$, so $s \xrightarrow{@u}^{\mathscr{T} \restriction X} t$ and $\phi(t) = t$. Therefore $\phi \in PR(\mathscr{T} \restriction X, \mathscr{T})$. □

Example 4.5.10 shows that convex bisimilarity on the restriction of a TTS can be extensional when it is not extensional upon the original TTS, because a duplicate state in an equivalence class can be removed.

Example 4.5.10 Consider the TTS $\mathscr{T}$ described in example 4.2.8, and let $X \subseteq \mathscr{T}$ be the set of states $\{\star_1, \mathit{ff}, \mathit{tt}, s\}$. Then convex bisimilarity on $\mathscr{T} \restriction X$ is extensional although it is not on $\mathscr{T}$.

Conversely, convex bisimilarity may not be extensional on $\mathscr{T} \restriction X$ even though convex bisimilarity is extensional on $\mathscr{T}$. This is because a state $u \in \mathscr{T}(\sigma)$ that can distinguish the states $s, t \in (\mathscr{T} \restriction X)(\sigma \to \tau)$ may not be in $\mathscr{T} \restriction X$.

Example 4.5.11 We define a TTS $\mathscr{T}$ with an extensional convex bisimilarity. There is a restriction of $\mathscr{T}$ where convex bisimilarity is not extensional. The states of $\mathscr{T}$ are defined by:

$$\mathscr{T}(\sigma) \stackrel{\text{def}}{=} \begin{cases} \{\star\} & \text{if } \sigma = \mathsf{unit} \\ \{\mathit{ff}, \mathit{tt}\} & \text{if } \sigma = \mathsf{bool} \\ \{s, t\} & \text{if } \sigma = \mathsf{bool} \to \mathsf{bool} \\ \emptyset & \text{otherwise} \end{cases}$$

The transitions are determined by:

$$\begin{array}{lll} \mathit{ff} \xrightarrow{0} \star & s \xrightarrow{@\mathit{ff}} \mathit{ff} & s \xrightarrow{@\mathit{tt}} \mathit{ff} \\ \mathit{tt} \xrightarrow{1} \star & t \xrightarrow{@\mathit{ff}} \mathit{ff} & t \xrightarrow{@\mathit{tt}} \mathit{tt} \end{array}$$

Then $\simeq_{\mathrm{CB}}^{\mathscr{T}}$ is extensional on $\mathscr{T}$, but if $X \subseteq \mathscr{T}$ is $\{\star, \mathit{ff}, s, t\}$, then $\simeq_{\mathrm{CB}}^{\mathscr{T}\restriction X}$ is not extensional on $\mathscr{T} \restriction X$ because $s \simeq_{\mathrm{CB}}^{\mathscr{T}\restriction X} t$ and $s \neq t$. Note that $s \not\simeq_{\mathrm{CB}}^{\mathscr{T}} t$.

Proposition 4.5.12 shows that the maps of PR are essentially identifications composed with inclusions. Every map $\phi \in PR(\mathscr{T}, \mathscr{U})$ factors through the restriction $\mathscr{U} \restriction Im(\phi)$ and the inclusion map from $\mathscr{U} \restriction Im(\phi)$ to $\mathscr{U}$. Moreover, $\mathscr{U} \restriction Im(\phi)$ has no more discriminative power than $\mathscr{T}$.

Proposition 4.5.12 Consider TTSs $\mathscr{T}$ and $\mathscr{U}$, and a map $\phi \in PR(\mathscr{T}, \mathscr{U})$. Then $Im(\phi) \subseteq \mathscr{U}$ is a set of states such that, for all states $s \in Im(\phi)$, $t \in \mathscr{U}$, and labels a such that $s \xrightarrow{a}^{\mathscr{U}} t$, either $t \in Im(\phi)$ or there exists $u \in \mathscr{U} \setminus Im(\phi)$ such that $a = @u$. The map ϕ factors through $\mathscr{U} \restriction Im(\phi)$ as $\phi = \phi_1 ; \phi_2$, where $\phi_1 \in PR(\mathscr{T}, \mathscr{U} \restriction Im(\phi))$, and $\phi_2 \in PR(\mathscr{U} \restriction Im(\phi), \mathscr{U})$ is the inclusion function. Moreover, for all states $s, t \in \mathscr{T}$, $s \simeq_{\mathrm{CB}}^{\mathscr{T}} t$ if and only if $\phi_1(s) \simeq_{\mathrm{CB}}^{\mathscr{U}\restriction Im(\phi)} \phi_1(t)$.

Proof Without loss of generality, consider $s \in \mathscr{T}$ (so $\phi(s) \in Im(\phi)$), $t \in \mathscr{U}$, and a label a such that $\phi(s) \xrightarrow{a}^{\mathscr{U}} t$. If there exists a label b from $\mathscr{T}$ such that $a = \phi(b)$, then by the labelled transition reflecting property of ϕ, there exists $v \in \mathscr{T}$ such that $s \xrightarrow{b}^{\mathscr{T}} v$ and $t = \phi(v)$, so $t \in Im(\phi)$. Otherwise there must be a state $u \in \mathscr{U}$ such that $a = @u$ and $u \notin Im(\phi)$, and we are done with the first part. Now define $\phi_1 = \phi$, so $\phi_1 : \mathscr{T} \to \mathscr{U} \restriction Im(\phi)$, and let $\phi_2 : \mathscr{U} \restriction Im(\phi) \to \mathscr{U}$ be the inclusion map. We have $\phi = \phi_1 ; \phi_2$. The function ϕ_1 preserves and reflects labelled transitions and may divergence because ϕ does, so $\phi_1 \in PR(\mathscr{T}, \mathscr{U} \restriction Im(\phi))$. By definition, every state of $\mathscr{U} \restriction Im(\phi)$ is in the image of ϕ, so by lemma 4.5.4, for all states $s, t \in \mathscr{T}$, $s \simeq_{\mathrm{CB}}^{\mathscr{T}} t$ if and only if $\phi_1(s) \simeq_{\mathrm{CB}}^{\mathscr{U}\restriction Im(\phi)} \phi_1(t)$. $\square$

We now consider the special role of the TTS $\mathscr{S}$ (see sections 4.3 and 4.4) in the category PR. There is a family of TTSs each of which has an applicatively compatible convex bisimilarity and a map to $\mathscr{S}$. This reinforces the idea that $\mathscr{S}$ contains as many states as possible with different behaviours whilst retaining an extensional convex bisimilarity. However, example 4.5.13 shows that there is a TTS with no maps to $\mathscr{S}$ where convex bisimilarity is applicatively compatible.

Example 4.5.13 We define a TTS $\mathscr{T}$ with extensional convex bisimilarity and a single state $s \in \mathscr{T}(\mathsf{unit} \to \mathsf{sum}\,\langle\rangle)$. Note that no TTS can have a state of $\mathsf{sum}\,\langle\rangle$, so $s \in \mathscr{T}(\mathsf{unit} \to \mathsf{sum}\,\langle\rangle)$ implies that there are no states of type unit in $\mathscr{T}$. Now $\mathscr{S}(\mathsf{unit}) = \{\star\}$, and so $\mathscr{S}(\mathsf{unit} \to \mathsf{sum}\,\langle\rangle)$ is empty. Therefore there are no maps in $PR(\mathscr{T}, \mathscr{S})$, because s cannot be mapped to any state of $\mathscr{S}$.

The problem demonstrated in example 4.5.13 is that a TTS with an applicatively compatible convex bisimilarity may have states of function type even though $\mathscr{S}$ has no states of the same type. Theorem 4.5.15 shows that for every TTS (with an applicatively compatible convex bisimilarity) that has no states of a given type whenever $\mathscr{S}$ has no states of that type, there is a map to $\mathscr{S}$. Before proving this, we define a partial function ξ that picks out one state of $\mathscr{S}$ for each type whenever there is a state of that type.

Lemma 4.5.14 Define a partial function ξ that maps each type σ to a state of $\mathscr{S}(\sigma)$ by induction:

$$\xi(\mathsf{sum}\,\langle\sigma_n \mid n<\kappa\rangle) \stackrel{\text{def}}{=} \begin{cases} \text{undefined} & \text{if } \forall n<\kappa.\,\xi(\sigma_n) \text{ is undefined} \\ \langle m,\xi(\sigma_m)\rangle & \text{if } m<\kappa \text{ is the least natural number} \\ & \text{such that } \xi(\sigma_m) \text{ is defined} \end{cases}$$

$$\xi(\mathsf{prod}\,\langle\sigma_n \mid n<\kappa\rangle) \stackrel{\text{def}}{=} \begin{cases} \text{undefined} & \text{if } \exists m<\kappa.\,\xi(\sigma_m) \text{ is undefined} \\ \langle\xi(\sigma_n) \mid n<\kappa\rangle & \text{otherwise} \end{cases}$$

$$\xi(\sigma\to\tau) \stackrel{\text{def}}{=} \begin{cases} \text{undefined} & \text{if } \xi(\sigma) \text{ is defined and} \\ & \xi(\tau) \text{ is undefined} \\ \{\langle A,\xi(\tau)\rangle \mid A\in\mathscr{S}(\sigma)\} & \text{otherwise} \end{cases}$$

$$\xi(\mathsf{P}_\perp(\sigma)) \stackrel{\text{def}}{=} \{\perp\}$$

Then, whenever $\xi(\sigma)$ is defined, $\xi(\sigma)\in\mathscr{S}(\sigma)$, and whenever $\xi(\sigma)$ is undefined, $\mathscr{S}(\sigma)$ is empty.

Proof The result must be proved by induction on types with the definition to show that the case for function types is well-defined. Consider $\mathsf{sum}\,\langle\sigma_n \mid n<\kappa\rangle$. If $\xi(\mathsf{sum}\,\langle\sigma_n \mid n<\kappa\rangle)$ is defined and is equal to $\langle m,\xi(\sigma_m)\rangle$, then $\xi(\sigma_m)$ is defined, and so by the induction hypothesis, $\xi(\sigma_m)\in\mathscr{S}(\sigma_m)$. Therefore $\langle m,\xi(\sigma_m)\rangle\in\mathscr{S}(\mathsf{sum}\,\langle\sigma_n \mid n<\kappa\rangle)$. If $\xi(\mathsf{sum}\,\langle\sigma_n \mid n<\kappa\rangle)$ is undefined, then suppose $\langle m,A\rangle\in\mathscr{S}(\mathsf{sum}\,\langle\sigma_n \mid n<\kappa\rangle)$, where $m<\kappa$. In this case, $A\in\mathscr{S}(\sigma_m)$ and $\xi(\sigma_m)$ is undefined, which contradicts the induction hypothesis. Therefore $\mathscr{S}(\mathsf{sum}\,\langle\sigma_n \mid n<\kappa\rangle)$ is empty. The case for product types is similar, and the case for computation types is trivial. Finally, consider the function type $\sigma\to\tau$. If $\xi(\sigma\to\tau)$ is defined, then either $\xi(\tau)$ is defined or $\xi(\sigma)$ is undefined. In the former case, by the induction hypothesis, $\xi(\tau)\in\mathscr{S}(\tau)$, and $\xi(\sigma\to\tau)$ is a constant function from $\mathscr{S}(\sigma)$ to $\mathscr{S}(\tau)$. In the latter case, $\mathscr{S}(\sigma)$ is empty, and so f is the unique function from $\mathscr{S}(\sigma)$ to $\mathscr{S}(\tau)$. In either case, $\xi(\sigma\to\tau)\in\mathscr{S}(\sigma\to\tau)$. If $\xi(\sigma\to\tau)$ is undefined, suppose for a contradiction that $f\in\mathscr{S}(\sigma\to\tau)$. By the induction hypothesis, $\xi(\sigma)\in\mathscr{S}(\sigma)$, so $f(\xi(\sigma))\in\mathscr{S}(\tau)$, which contradicts the induction hypothesis at τ. Therefore $\mathscr{S}(\sigma\to\tau)$ is empty. $\square$

Now we can show that maps to $\mathscr{S}$ exist. For a map from a TTS $\mathscr{T}$ to $\mathscr{S}$, a state of $\mathscr{T}$ with a function type $\sigma\to\tau$ is mapped to a function $f:\mathscr{S}(\sigma)\to\mathscr{S}(\tau)$. The function f maps a state of $\mathscr{S}(\sigma)$ to $\xi(\sigma)$ if it is not the image of a state from $\mathscr{T}(\sigma)$. This causes the restriction of $\mathscr{S}$ to the image of the map from $\mathscr{T}$ to $\mathscr{S}$ to have an extensional convex bisimilarity.

Theorem 4.5.15 Consider a TTS $\mathscr{T}$ with an applicatively compatible convex bisimilarity and such that, for all types σ and τ, if $\xi(\sigma\to\tau)$ is undefined (equivalently $\mathscr{S}(\sigma\to\tau)$ is empty) then $\mathscr{T}(\sigma\to\tau)$ is empty. Then there exists a map $\phi\in PR(\mathscr{T},\mathscr{S})$ such that, for all states $s,t\in\mathscr{T}$, $s\simeq^{\mathscr{T}}_{\mathrm{CB}}t$ if and only if $\phi(s)\simeq^{\mathscr{S}}_{\mathrm{CB}}\phi(t)$. In addition, convex bisimilarity is extensional in $\mathscr{S}\restriction Im(\phi)$.

Proof The function ϕ is defined by induction on the type of states. Simultaneously, we argue that ϕ preserves and reflects labelled transitions and may divergence, and, for all states $s,t\in\mathscr{T}$,

$s \simeq_{\mathrm{CB}}^{\mathscr{T}} t$ if and only if $\phi(s) \simeq_{\mathrm{CB}}^{\mathscr{S}} \phi(t)$. At states of coproduct, product, or computation type, ϕ is defined by:

- If $s \in \mathscr{T}(\mathsf{sum}\,\langle \sigma_n \mid n < \kappa \rangle)$ and there exists $m < \kappa$ and $t \in \mathscr{T}(\sigma_m)$ such that $s \xrightarrow{m} t$, then $\phi(s) \stackrel{\text{def}}{=} \langle m, \phi(t) \rangle$.
- If $s \in \mathscr{T}(\mathsf{prod}\,\langle \sigma_n \mid n < \kappa \rangle)$, then $\phi(s) \stackrel{\text{def}}{=} \langle \phi(s@n) \mid n < \kappa \rangle$.
- If $s \in \mathscr{T}(\mathsf{P}_\perp(\sigma))$, then $\phi(s) \stackrel{\text{def}}{=} \{\perp \mid s \Uparrow^{\text{may}}\} \cup \{\phi(t) \mid s \xrightarrow{\Downarrow} t\}$.

It is straightforward to show that for all states of coproduct, product, or function type, ϕ preserves and reflects labelled transitions and may divergence. Moreover, the arguments of proposition 4.5.3 and lemma 4.5.4(1) show that $s \simeq_{\mathrm{CB}}^{\mathscr{T}} t$ if and only if $\phi(s) \simeq_{\mathrm{CB}}^{\mathscr{S}} \phi(t)$ for states $s, t \in \mathscr{T}$ of those types. For function types, consider $\sigma \to \tau$. If $\xi(\sigma)$ is defined and $\xi(\tau)$ is undefined, then $\xi(\sigma \to \tau)$ is undefined, and, by hypothesis, $\mathscr{T}(\sigma \to \tau)$ is empty, so we are done. Otherwise, we must define $\phi(s)$ for every state $s \in \mathscr{T}(\sigma \to \tau)$. If $\xi(\sigma)$ is undefined, then $\mathscr{S}(\sigma)$ is empty by lemma 4.5.14, and so $\phi(s)$ can be defined as the unique function from the empty set to $\mathscr{S}(\tau)$. The remaining case is that both $\xi(\sigma)$ and $\xi(\tau)$ are defined, so $\mathscr{S}(\sigma \to \tau)$ is non-empty. We have to define $\phi(s) : \mathscr{S}(\sigma) \to \mathscr{S}(\tau)$, so consider $A \in \mathscr{S}(\sigma)$. If $A \notin Im(\phi)$ (the action of ϕ is already defined upon $\mathscr{T}(\sigma)$ and ϕ preserves types), then define $\phi(s)(A) = \xi(\tau) \in \mathscr{S}(\tau)$. Otherwise, there exists $t \in \mathscr{T}(\sigma)$ such that $\phi(t) = A$, and we define $\phi(s)(A) = \phi(s@t) \in \mathscr{S}(\tau)$. For this to be well-defined, we must show that $\phi(t) = \phi(u)$ implies $\phi(s@t) = \phi(s@u)$, for all $t, u \in \mathscr{S}(\sigma)$. Suppose $\phi(t) = \phi(u)$, so $\phi(t) \simeq_{\mathrm{CB}}^{\mathscr{S}} \phi(u)$, and, by the induction hypothesis at σ, $t \simeq_{\mathrm{CB}}^{\mathscr{T}} u$. By applicative compatibility of convex bisimilarity with respect to $\mathscr{T}$, $s@t \simeq_{\mathrm{CB}}^{\mathscr{T}} s@u$. Applying the induction hypothesis at τ gives $\phi(s@t) \simeq_{\mathrm{CB}}^{\mathscr{S}} \phi(s@u)$. However, $\mathscr{S}$ has an extensional convex bisimilarity, so $\phi(s@t) = \phi(s@u)$ as required. Therefore ϕ is well-defined at states of function type. By construction, ϕ preserves and reflects labelled transitions. We also need to show that, for all states $s, t \in \mathscr{T}(\sigma \to \tau)$, $s \simeq_{\mathrm{CB}}^{\mathscr{T}} t$ if and only if $\phi(s) \simeq_{\mathrm{CB}}^{\mathscr{S}} \phi(t)$. For the forward direction, consider $s, t \in \mathscr{T}(\sigma \to \tau)$ such that $s \simeq_{\mathrm{CB}}^{\mathscr{T}} t$, so $\xi(\sigma)$ is undefined or $\xi(\tau)$ is defined. We want to show that $\phi(s) \simeq_{\mathrm{CB}}^{\mathscr{S}} \phi(t)$. If $\xi(\sigma)$ is undefined, then $\mathscr{S}(\sigma)$ is empty, and so $\phi(s) = \phi(t)$ is the unique function from $\mathscr{S}(\sigma)$ to $\mathscr{S}(\tau)$. Otherwise, both $\xi(\sigma)$ and $\xi(\tau)$ are defined. Consider $A \in \mathscr{S}(\sigma)$. We have to show that $\phi(s)(A) \simeq_{\mathrm{CB}}^{\mathscr{S}} \phi(t)(A)$. If $A \notin Im(\phi)$, then $\phi(s)(A) = \xi(\tau) = \phi(t)(A)$. Otherwise, there exists a state $u \in \mathscr{T}(\sigma)$ such that $\phi(u) = A$, and so $\phi(s)(A) = \phi(s@u)$ and $\phi(t)(A) = \phi(t@u)$. Using $s \simeq_{\mathrm{CB}}^{\mathscr{T}} t$, we have $s@u \simeq_{\mathrm{CB}}^{\mathscr{T}} t@u$. Then, by the induction hypothesis at τ, $\phi(s@u) \simeq_{\mathrm{CB}}^{\mathscr{S}} \phi(t@u)$. Therefore $s \simeq_{\mathrm{CB}}^{\mathscr{T}} t$ implies $\phi(s) \simeq_{\mathrm{CB}}^{\mathscr{S}} \phi(t)$. The reverse direction follows by the same argument used for proposition 4.5.3.

Finally we claim that convex bisimilarity on $\mathscr{S} \upharpoonright Im(\phi)$ is extensional. It suffices to prove by induction on a type σ that, for all $s, t \in \mathscr{T}(\sigma)$, $\phi(s) \simeq_{\mathrm{CB}}^{\mathscr{S} \upharpoonright Im(\phi)} \phi(t)$ implies $\phi(s) = \phi(t)$. We consider the case for function types, the other cases are straightforward. Suppose that $s, t \in \mathscr{T}(\sigma \to \tau)$ and $\phi(s) \simeq_{\mathrm{CB}}^{\mathscr{S} \upharpoonright Im(\phi)} \phi(t)$. To prove $\phi(s) = \phi(t)$, we need to show that, for all $A \in \mathscr{S}(\sigma)$, $\phi(s)(A) = \phi(t)(A)$. Consider $A \in \mathscr{S}(\sigma)$. If $A \notin Im(\phi)$, then $\phi(s)(A) = \xi(\tau) = \phi(t)(A)$. Otherwise $A \in Im(\phi)$, and the definition of bisimilarity for $\mathscr{S} \restriction Im(\phi)$ implies that $\phi(s)(A) \simeq_{\mathrm{CB}}^{\mathscr{S} \restriction Im(\phi)} \phi(t)(A)$. Applying the induction hypothesis at τ gives $\phi(s)(A) = \phi(t)(A)$. Therefore $\phi(s) = \phi(t)$, and convex bisimilarity on $\mathscr{S} \restriction Im(\phi)$ is extensional. $\square$

The TTS $\mathscr{S}$ is a weakly terminal object but not a terminal object in the full subcategory of *PR* with TTSs satisfying the hypotheses of theorem 4.5.15. This is because there may be states other than the ones picked out by ξ that can be used as the image of non-definable elements.

Example 4.5.16 We define a TTS $\mathscr{T}$ with extensional convex bisimilarity that has more than one map to $\mathscr{S}$.

$$\mathscr{T}(\sigma) \stackrel{\text{def}}{=} \begin{cases} \{\star\} & \text{if } \sigma = \mathsf{unit} \\ \{\mathit{ff}\} & \text{if } \sigma = \mathsf{bool} \\ \{s\} & \text{if } \sigma = \mathsf{bool} \to \mathsf{bool} \\ \emptyset & \text{otherwise} \end{cases}$$

The transitions are determined $\mathit{ff} \xrightarrow{0} \star$ and $s \xrightarrow{@\mathit{ff}} \mathit{ff}$. Then $\star$ and ff have fixed interpretations in $\mathscr{S}$, but s can be mapped to either one of the functions $f : \mathscr{S}(\mathsf{bool}) \to \mathscr{S}(\mathsf{bool})$ such that $f(\mathit{ff}) = \mathit{ff}$.

In chapter 5 we show that many TTSs determined by fragments of $\mathscr{L}$ satisfy the hypotheses of theorem 4.5.15, proving that they are closely related to restrictions of $\mathscr{S}$.

Chapter 5

Programming Language TTSs

In this chapter we define a family of TTSs based upon the family of fragments of the programming language $\mathscr{L}$. The TTS structure provides definitions of the variants of similarity, mutual similarity, and bisimilarity upon the programs of each fragment, and they coincide with the usual definitions for non-deterministic λ-calculi. The majority of results from chapter 4 apply to TTSs derived from $\mathscr{L}$. Here we show that the lower, upper, and convex variants of similarity are compatible and satisfy the Scott induction principle for all fragments, and that the other relations are compatible on a restricted, but useful, collection of fragments. We investigate relative definability of different forms of erratic non-determinism with respect to convex bisimilarity, and show that the relative definability equivalence classes induce different convex bisimilarity relations, i.e., extending a fragment with a new form of erratic non-determinism can allow more terms from the original fragment to be distinguished.

Section 5.1 defines the family of TTSs based upon the programming language. Section 5.2 studies the similarity and bisimilarity operations on the family of TTSs and discusses how examples for the TTS $\mathscr{S}$ in sections 4.3 and 4.4 can be reconstructed in the programming language and its fragments. Section 5.3 considers operations upon relations on open terms that are used in section 5.4 to establish the compatibility results. The proofs are based upon Howe and Ong's techniques, and use Lassen's relational presentation to prove results that apply to collections of fragments. Section 5.5 identifies a sequence of relative definability equivalence classes of programs of type $\mathsf{P}_{\bot}(\mathsf{nat})$ and establishes a relationship with Turing degrees. Section 5.6 describes elementary properties of the variants of similarity, mutual similarity, and bisimilarity, and then shows that each member of the sequence of relative definability equivalence classes has a more discriminating convex bisimilarity than that of the preceding member. Section 5.7 proves the Scott induction principle for the variants of similarity.

5.1 $\mathscr{L}_0$ and $\mathscr{L}_0(\mathcal{E})$

The programming language $\mathscr{L}$ determines a TTS with programs as states (using the same type assignment system). We define the TTS based on $\mathscr{L}$ and consider restrictions to fragments of the programming language. Throughout this chapter, $\mathcal{E}$ ranges over sets of well-typed terms of the programming language $\mathscr{L}$.

The labelled transitions between programs capture aspects of the may convergence relation and decompositions of the canonical programs resulting from evaluation. They are based upon Gordon's [Gor94, Gor95a, Gor95b] labelled transition relations for functional languages, which in turn draw upon Abramsky's [Abr90] ATS for the lazy λ-calculus (see the discussion on page 87). The labelled transitions for programs of computation type correspond to Gordon's definitions for programs of *active* type. The distinction between active and *passive* types is not important for TTSs at value types because programs of value types always terminate. Note that the labelled transition relation for $\mathscr{L}_0$ does not incorporate silent τ-transitions for individual reduction steps, unlike the LTSs proposed for concurrent higher-order languages (see [FHJ95, Jef95]).

Definition 5.1.1 The states of the TTS $\mathscr{L}_0$ are the programs of the programming language $\mathscr{L}$ with the same type assignment system, i.e., $M \in \mathscr{L}_0(\sigma)$ if and only if $\vdash M : \sigma$. The may divergence predicate for the TTS $\mathscr{L}_0$ is the may divergence predicate defined in section 3.5 for the programming language. The labelled transitions of the TTS $\mathscr{L}_0$ are defined by:

1. $M \in \mathscr{L}_0(\mathsf{sum}\,\langle \sigma_n \mid n < \kappa \rangle) \Longrightarrow$
   $\forall m < \kappa. \forall N \in \mathscr{L}_0. M \xrightarrow{m} N \Longleftrightarrow M \Downarrow^{\mathrm{may}} \mathsf{inj}\, m\, \mathsf{of}\, N$
2. $M \in \mathscr{L}_0(\mathsf{prod}\,\langle \sigma_n \mid n < \kappa \rangle) \Longrightarrow$
   $\forall \langle N_n \in \mathscr{L}_0 \mid n < \kappa \rangle. (\forall m < \kappa. M \xrightarrow{m} N_m) \Longleftrightarrow M \Downarrow^{\mathrm{may}} \mathsf{tuple}\,\langle N_n \mid n < \kappa \rangle$
3. $M \in \mathscr{L}_0(\sigma \to \tau) \Longrightarrow$
   $\forall N \in \mathscr{L}. (\forall L \in \mathscr{L}_0(\sigma). M \xrightarrow{@L} N[L/x]) \Longleftrightarrow M \Downarrow^{\mathrm{may}} \lambda x. N$
4. $M \in \mathscr{L}_0(\mathsf{P}_\perp(\sigma)) \Longrightarrow$
   $\forall N \in \mathscr{L}_0. M \xrightarrow{\Downarrow} N \Longleftrightarrow M \Downarrow^{\mathrm{may}} [N]$

For each set of terms $\mathcal{E} \subseteq \mathscr{L}$, the TTS $\mathscr{L}_0(\mathcal{E})$ is defined to be the restriction $\mathscr{L}_0 \restriction \mathscr{L}(\mathcal{E})$ of the TTS $\mathscr{L}_0$ to the programs of the fragment $\mathscr{L}(\mathcal{E})$. We write $\mathscr{L}_0(M_1, \ldots, M_n)$ for $\mathscr{L}_0(\{M_1, \ldots, M_n\})$.

A summary of the notation may be useful at this point:

1. (a) $\mathscr{L}$ is the set of all well-typed terms.
   (b) $\mathscr{L}$ is also used to refer to the programming language defined in chapter 3, which includes the set of well-typed terms and the operational semantics.
2. (a) $\mathscr{L}_0$ is the subset of $\mathscr{L}$ consisting of all well-typed programs.
   (b) $\mathscr{L}_0$ is the largest TTS defined in definition 5.1.1. The states of $\mathscr{L}_0$ are well-typed programs, and as with other TTSs, $\mathscr{L}_0$ is used for the set of all states of that TTS. This usage coincides with 2(a).

3. (a) $\mathscr{L}_0(\mathcal{E})$ is the set of programs in the smallest fragment $\mathscr{L}(\mathcal{E})$ containing the set of well-typed terms $\mathcal{E}$.

   (b) $\mathscr{L}_0(\mathcal{E})$ is a TTS defined by restriction in definition 5.1.1. States are programs from the fragment $\mathscr{L}(\mathcal{E})$.

4. $\mathscr{L}_0(\sigma)$ and $\mathscr{L}_0(\mathcal{E})(\sigma)$ are the sets of states of type σ from the TTSs $\mathscr{L}_0$ and $\mathscr{L}_0(\mathcal{E})$ respectively.

Lemma 5.1.2 $\mathscr{L}_0$ is a TTS and, for all $\mathcal{E} \subseteq \mathscr{L}$, $\mathscr{L}_0(\mathcal{E})$ is a TTS.

Proof It is straightforward to prove that $\mathscr{L}_0$ is a TTS, because the may convergence relation is a function at value types by lemma 3.4.4, lemma 3.5.2, and proposition 3.6.3. For $\mathscr{L}_0(\mathcal{E})$ to be a TTS, we must establish the conditions of definition 4.5.8, i.e., if $M \in \mathscr{L}_0(\mathcal{E})$, $N \in \mathscr{L}_0$, and a is a label, such that $M \xrightarrow{a} N$, either $N \in \mathscr{L}_0(\mathcal{E})$ or there exists $L \in \mathscr{L}_0 \setminus \mathscr{L}_0(\mathcal{E})$ such that $a = @L$. This holds when M does not have function type because $\mathscr{L}_0(\mathcal{E})$ is closed under evaluation by lemma 3.7.2 and is closed under taking subterms by definition. When $M \in \mathscr{L}_0(\mathcal{E})(\sigma \to \tau)$ and $N \in \mathscr{L}_0(\mathcal{E})(\sigma)$, we also make use of the fact that $\mathscr{L}_0(\mathcal{E})$ is closed under substitution. Therefore $\mathscr{L}_0(\mathcal{E})$ is a TTS. $\square$

Defining the family of TTSs by restriction creates a rich collection of relationships between the TTSs, because every TTS $\mathscr{L}_0(\mathcal{E})$ is a restriction of $\mathscr{L}_0$. More generally, if $\mathscr{L}(\mathcal{E}_1) \subseteq \mathscr{L}(\mathcal{E}_2)$, then the TTS $\mathscr{L}_0(\mathcal{E}_1)$ is a restriction of the TTS $\mathscr{L}_0(\mathcal{E}_2)$:

$$\mathscr{L}_0(\mathcal{E}_1) = \mathscr{L}_0(\mathcal{E}_2) \restriction \mathscr{L}(\mathcal{E}_1)$$

With these relationships, we can deduce inclusions between the variants of convex bisimilarity for $\mathscr{L}_0(\mathcal{E}_1)$ and $\mathscr{L}_0(\mathcal{E}_2)$.

The generic projection and application operations that are available in all TTSs (see definition 4.1.1) do not coincide with syntactic projection and application because of the may convergence clauses in the definition of the transitions of $\mathscr{L}_0$. However, the resulting programs do have the same may convergence and may divergence behaviour.

Lemma 5.1.3

1. For a program $M \in \mathscr{L}_0(\mathcal{E})(\mathsf{prod}\,\langle \sigma_n \mid n < \kappa \rangle)$, $m < \kappa$, and a canonical program $K \in \mathscr{L}_0(\mathcal{E})(\sigma_m)$:

$$\begin{aligned} M@m \Downarrow^{\text{may}} K &\iff \mathsf{proj}\,m\,\mathsf{of}\,M \Downarrow^{\text{may}} K \\ M@m \Uparrow^{\text{may}} &\iff \mathsf{proj}\,m\,\mathsf{of}\,M \Uparrow^{\text{may}} \end{aligned}$$

2. For programs $M \in \mathscr{L}_0(\mathcal{E})(\sigma \to \tau)$ and $N \in \mathscr{L}_0(\mathcal{E})(\sigma)$, and a canonical program $K \in \mathscr{L}_0(\tau)$:

$$\begin{array}{rcl} M@N \Downarrow^{\mathrm{may}} K & \iff & M\,N \Downarrow^{\mathrm{may}} K \\ M@N \Uparrow^{\mathrm{may}} & \iff & M\,N \Uparrow^{\mathrm{may}} \end{array}$$

Proof Follows from the may convergence and may divergence rules for projection and function application. $\square$

It is useful to know when states are hereditarily deterministic or total because lemma 4.2.5 then gives identifications between variants of similarity, mutual similarity, and bisimilarity. Lemma 5.1.4 identifies conditions under which all states of a TTS derived from a language fragment will be hereditarily deterministic.

Lemma 5.1.4 If $\mathcal{E}$ contains no instances of erratic choice constructors, then every state of the TTS $\mathscr{L}_0(\mathcal{E})$ is hereditarily deterministic.

Proof Straightforward because the restrictions to $\mathscr{L}(\mathcal{E})$ of the reduction relation $\twoheadrightarrow$ and the deterministic reduction relation $\twoheadrightarrow_{\mathrm{det}}$ coincide. $\square$

However, a program may be hereditarily deterministic in one TTS but not in a larger TTS because of the behaviour of functions.

Example 5.1.5 The program $\vdash \lambda x.x : \mathsf{P}_\perp(\mathsf{bool}) \to \mathsf{P}_\perp(\mathsf{bool})$ is hereditarily deterministic as a state of $\mathscr{L}_0(\emptyset)$, but not as a state of $\mathscr{L}_0(?\langle\mathsf{false},\mathsf{true}\rangle)$ because:

$$\lambda x.x \xrightarrow{?\langle\mathsf{false},\mathsf{true}\rangle}_{\mathscr{L}_0(?\langle\mathsf{false},\mathsf{true}\rangle)} ?\langle\mathsf{false},\mathsf{true}\rangle$$

and $?\langle\mathsf{false},\mathsf{true}\rangle$ is not hereditarily deterministic.

For any TTS of the form $\mathscr{L}_0(\mathcal{E})$, every computation type $\mathsf{P}_\perp(\sigma)$ has a program Ω that is not hereditarily total. There may be types with P-order greater than one where all states of that type are hereditarily total, because fragments need not be closed under infinitary product constructors.

5.2 Similarity and Bisimilarity

The TTSs derived from the programming language $\mathscr{L}$ and its fragments inherit definitions of the variants of similarity, mutual similarity, and bisimilarity. In this section we give an elementary description of the variants of similarity and bisimilarity for language fragments, prove the existence of maps in the category *PR* from the TTSs for certain language fragments to $\mathscr{S}$, and consider how the examples from sections 4.3 and 4.4 can be reconstructed in TTSs derived from the language fragments. We start by defining the space of relations on programs.

$$\begin{array}{l}
\vdash M_1 \mathcal{R} N_1 : \mathsf{sum}\,\langle \sigma_n \mid n < \kappa \rangle \iff \\
\exists m < \kappa.\, \exists M_2, N_2. \\
\qquad (M_1 \Downarrow^{\mathrm{may}} \mathsf{inj}\, m\, \mathsf{of}\, M_2) \wedge (N_1 \Downarrow^{\mathrm{may}} \mathsf{inj}\, m\, \mathsf{of}\, N_2) \wedge \\
\qquad (\vdash M_2 \mathcal{R} N_2 : \sigma_m)
\end{array}$$

$$\begin{array}{l}
\vdash M \mathcal{R} N : \mathsf{prod}\,\langle \sigma_n \mid n < \kappa \rangle \iff \\
\exists \langle M'_n \mid n < \kappa \rangle, \langle N'_n \mid n < \kappa \rangle. \\
\qquad (M_1 \Downarrow^{\mathrm{may}} \mathsf{tuple}\,\langle M'_n \mid n < \kappa \rangle) \wedge (N_1 \Downarrow^{\mathrm{may}} \mathsf{tuple}\,\langle N'_n \mid n < \kappa \rangle) \wedge \\
\qquad (\forall m < \kappa.\, \vdash M'_m \mathcal{R} N'_m : \sigma_m)
\end{array}$$

$$\begin{array}{l}
\vdash M_1 \mathcal{R} N_1 : \sigma \to \tau \iff \\
\exists M_2, N_2. \\
\qquad (M_1 \Downarrow^{\mathrm{may}} \lambda x.\, M_2) \wedge (N_1 \Downarrow^{\mathrm{may}} \lambda x.\, N_2) \wedge \\
\qquad (\forall L \in \mathscr{L}_0(\mathcal{E})(\sigma).\, \vdash M_2[L/x] \mathcal{R} N_2[L/x] : \tau)
\end{array}$$

Figure 5.1: Unfoldings of similarity and bisimilarity for $\mathscr{L}_0(\mathcal{E})$ at value types

Definition 5.2.1 The set $Rel_0(\mathcal{E})$ consists of all sets of pairs $\mathcal{R} \subseteq \mathscr{L}_0(\mathcal{E}) \times \mathscr{L}_0(\mathcal{E})$ where $\langle M, N \rangle \in \mathcal{R}$ implies there exists a type σ such that $M, N \in \mathscr{L}_0(\mathcal{E})(\sigma)$. The complete lattice $\langle Rel_0(\mathcal{E}), \subseteq \rangle$ has meets defined as intersection for non-empty sets. The meet of the empty set is:

$$\{\langle M, N \rangle \mid \exists \sigma.\, M, N \in \mathscr{L}_0(\mathcal{E})(\sigma)\} \in Rel_0(\mathcal{E})$$

We write $Rel_0(M_1, \ldots, M_n)$ for $Rel_0(\{M_1, \ldots, M_n\})$. When $\mathcal{R} \in Rel_0(\mathcal{E})$, we write $\vdash M \mathcal{R} N : \sigma$ or $\vdash \langle M, N \rangle \in \mathcal{R} : \sigma$ for $M, N \in \mathscr{L}_0(\mathcal{E})(\sigma)$ and $\langle M, N \rangle \in \mathcal{R}$.

The variants of similarity and bisimilarity for TTSs derived from $\mathscr{L}$ and its fragments can be characterised directly in terms of the operational semantics. If $\mathcal{R} \in Rel_0(\mathcal{E})$ is a variant of similarity, mutual similarity, or bisimilarity, then the characterisations in figure 5.1 hold at value types for programs of $\mathscr{L}_0(\mathcal{E})$. The characterisations in figure 5.2 hold for the relations at computation types (cf. figure 4.1).

The TTS $\mathscr{L}_0$ derived from the programming language can be related to the TTS $\mathscr{S}$ described in sections 4.3 and 4.4 using theorem 4.5.15. To do this, we need to know that $\mathscr{L}_0(\sigma \to \tau)$ is empty whenever $\xi(\sigma \to \tau)$ is undefined (see lemma 4.5.14 for the definition of the function ξ). Lemma 5.2.2 establishes a more general result for open terms.

Lemma 5.2.2 Consider an environment $\Gamma = x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$. If $\xi(\sigma_1), \ldots, \xi(\sigma_n)$ are defined and there exists a term $\Gamma \vdash M : \sigma$, then $\xi(\sigma)$ is also defined.

Proof By induction on the derivation of $\Gamma \vdash M : \sigma$. The cases for terms of a computation type are trivial because every computation type is in the domain of ξ. We give three representative cases:

$$\vdash M_1 \lesssim^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{LS}} N_1 : \mathsf{P}_\perp(\sigma) \iff$$
$$\forall M_2 . M_1 \Downarrow^{\mathrm{may}} [M_2] \implies \exists N_2 . N_1 \Downarrow^{\mathrm{may}} [N_2] \wedge \vdash M_2 \lesssim^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{LS}} N_2 : \sigma$$

$$\vdash M_1 \lesssim^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{US}} N_1 : \mathsf{P}_\perp(\sigma) \iff$$
$$M_1 \Downarrow^{\mathrm{must}} \implies$$
$$(N_1 \Downarrow^{\mathrm{must}} \wedge \forall N_2 . N_1 \Downarrow^{\mathrm{may}} [N_2] \implies \exists M_2 . M_1 \Downarrow^{\mathrm{may}} [M_2] \wedge \vdash M_2 \lesssim^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{US}} N_2 : \sigma)$$

$$\vdash M_1 \lesssim^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{CS}} N_1 : \mathsf{P}_\perp(\sigma) \iff$$
$$(\forall M_2 . M_1 \Downarrow^{\mathrm{may}} [M_2] \implies \exists N_2 . N_1 \Downarrow^{\mathrm{may}} [N_2] \wedge \vdash M_2 \lesssim^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{CS}} N_2 : \sigma) \wedge$$
$$(M_1 \Downarrow^{\mathrm{must}} \implies$$
$$(N_1 \Downarrow^{\mathrm{must}} \wedge \forall N_2 . N_1 \Downarrow^{\mathrm{may}} [N_2] \implies \exists M_2 . M_1 \Downarrow^{\mathrm{may}} [M_2] \wedge \vdash M_2 \lesssim^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{CS}} N_2 : \sigma))$$

$$\vdash M_1 \lesssim^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{RS}} N_1 : \mathsf{P}_\perp(\sigma) \iff$$
$$(M_1 \Downarrow^{\mathrm{must}} \implies N_1 \Downarrow^{\mathrm{must}}) \wedge$$
$$(\forall N_2 . N_1 \Downarrow^{\mathrm{may}} [N_2] \implies \exists M_2 . M_1 \Downarrow^{\mathrm{may}} [M_2] \wedge \vdash M_2 \lesssim^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{RS}} N_2 : \sigma)$$

$$\vdash M_1 \simeq^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{LB}} N_1 : \mathsf{P}_\perp(\sigma) \iff$$
$$(\forall M_2 . M_1 \Downarrow^{\mathrm{may}} [M_2] \implies \exists N_2 . N_1 \Downarrow^{\mathrm{may}} [N_2] \wedge \vdash M_2 \simeq^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{LB}} N_2 : \sigma) \wedge$$
$$(\forall N_2 . N_1 \Downarrow^{\mathrm{may}} [N_2] \implies \exists M_2 . M_1 \Downarrow^{\mathrm{may}} [M_2] \wedge \vdash M_2 \simeq^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{LB}} N_2 : \sigma)$$

$$\vdash M_1 \simeq^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{UB}} N_1 : \mathsf{P}_\perp(\sigma) \iff$$
$$(M_1 \Uparrow^{\mathrm{may}} \wedge N_1 \Uparrow^{\mathrm{may}}) \vee$$
$$((M_1 \Downarrow^{\mathrm{must}} \wedge N_1 \Downarrow^{\mathrm{must}}) \wedge$$
$$(\forall M_2 . M_1 \Downarrow^{\mathrm{may}} [M_2] \implies \exists N_2 . N_1 \Downarrow^{\mathrm{may}} [N_2] \wedge \vdash M_2 \simeq^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{UB}} N_2 : \sigma) \wedge$$
$$(\forall N_2 . N_1 \Downarrow^{\mathrm{may}} [N_2] \implies \exists M_2 . M_1 \Downarrow^{\mathrm{may}} [M_2] \wedge \vdash M_2 \simeq^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{UB}} N_2 : \sigma))$$

$$\vdash M_1 \simeq^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{CB}} N_1 : \mathsf{P}_\perp(\sigma) \iff$$
$$(M_1 \Downarrow^{\mathrm{must}} \iff N_1 \Downarrow^{\mathrm{must}}) \wedge$$
$$(\forall M_2 . M_1 \Downarrow^{\mathrm{may}} [M_2] \implies \exists N_2 . N_1 \Downarrow^{\mathrm{may}} [N_2] \wedge \vdash M_2 \simeq^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{CB}} N_2 : \sigma) \wedge$$
$$(\forall N_2 . N_1 \Downarrow^{\mathrm{may}} [N_2] \implies \exists M_2 . M_1 \Downarrow^{\mathrm{may}} [M_2] \wedge \vdash M_2 \simeq^{\mathscr{L}_0(\mathcal{E})}_{\mathrm{CB}} N_2 : \sigma)$$

Figure 5.2: Unfoldings of similarity and bisimilarity for $\mathscr{L}_0(\mathcal{E})$ at $\mathsf{P}_\perp(\sigma)$

1. $\Gamma \vdash \mathsf{case}\, M\, \mathsf{of}\, \langle x_n.N_n \mid n < \kappa \rangle : \tau$
   By the induction hypothesis for $\Gamma \vdash M : \mathsf{sum}\, \langle \sigma_n \mid n < \kappa \rangle$, we have that $\xi(\mathsf{sum}\, \langle \sigma_n \mid n < \kappa \rangle)$ is defined, and so there exists $m < \kappa$ such that $\xi(\sigma_m)$ is defined. Then the induction hypothesis for $\Gamma, x_m : \sigma_m \vdash N_m : \tau$ implies that $\xi(\tau)$ is defined.

2. $\Gamma \vdash \lambda x{:}\sigma.M : \sigma \to \tau$
   If $\xi(\sigma)$ is undefined, then $\xi(\sigma \to \tau)$ is defined, and we are done. Otherwise $\xi(\sigma)$ is defined. The induction hypothesis for $\Gamma, x : \sigma \vdash M : \tau$ implies that $\xi(\tau)$ is also defined, and therefore $\xi(\sigma \to \tau)$ is defined.

3. $\Gamma \vdash M\,N : \tau$
   By the induction hypothesis, both $\xi(\sigma \to \tau)$ and $\xi(\sigma)$ are defined. Therefore $\xi(\tau)$ is also defined. □

If convex bisimilarity is applicatively compatible on a TTS $\mathscr{L}_0(\mathcal{E})$, then programs from $\mathscr{L}_0(\mathcal{E})$ can be related to states of the TTS $\mathscr{S}$. This (functional) relationship does not constitute a denotational semantics because its definition is not *compositional*, i.e., the image of a term M is not defined in terms of the images of the immediate subterms of M.

Corollary 5.2.3 If $\mathscr{L}(\mathcal{E})$ is a language fragment such that $\simeq_{\mathrm{CB}}^{\mathscr{L}_0(\mathcal{E})}$ is applicatively compatible, then there is a map $\phi \in PR(\mathscr{L}_0(\mathcal{E}), \mathscr{S})$.

Proof The hypotheses of theorem 4.5.15 hold by assumption and lemma 5.2.2. □

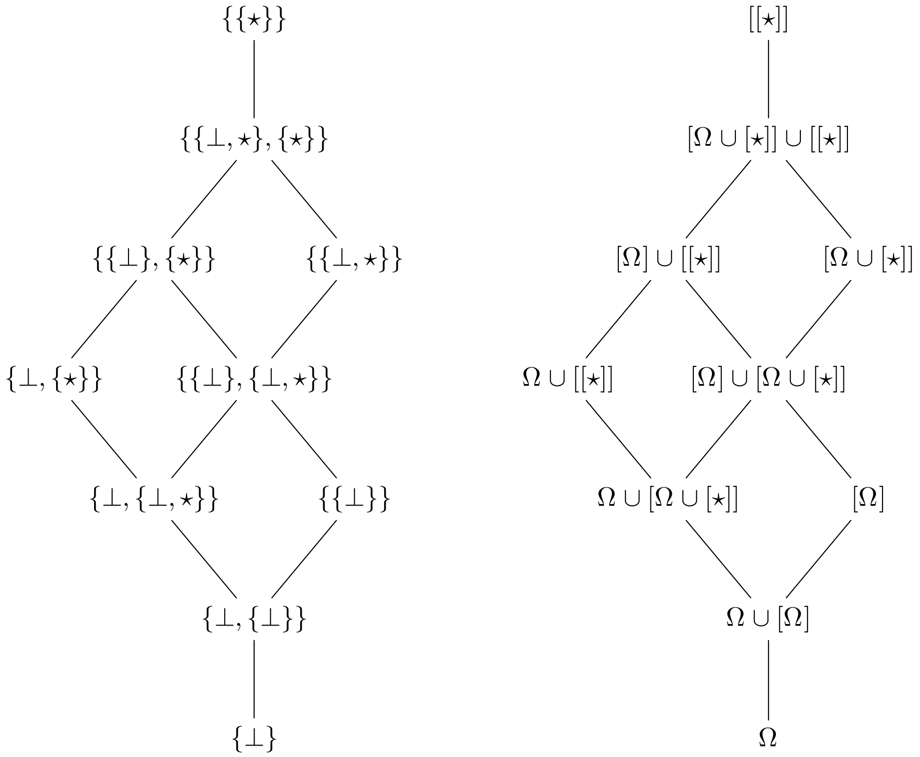
Convex bisimilarity is shown to be applicatively compatible for certain language fragments in section 5.4.

The majority of examples for the TTS $\mathscr{S}$ from sections 4.3 and 4.4 have analogues in sufficiently expressive language fragments. For types with no function type constructors, it is only necessary to show that states of $\mathscr{S}$ used in examples are in the image of $\phi \in PR(\mathscr{L}_0(\mathcal{E}), \mathscr{S})$. For example, figure 5.3 compares convex similarity upon programs with type $\mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{unit}))$ with convex similarity upon the states of $\mathscr{S}$ with type $\mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{unit}))$. A minor benefit of using Moggi's computational λ-calculus is that the example programs are simple because the unit term constructor can be used instead of λ-abstraction.

5.3 Relations on Open Terms

The definitions and results in this section concern relations on open terms, and are used in the proofs of compatibility (section 5.4) and fixed-point properties (section 5.7). Following Lassen [Las98b], a few key operators on relations (*closed restriction*, *open extension*, *relational substitution*, and *compatible refinement*) provide a layer of abstraction from terms, which makes it easier to construct proofs that apply to different programming languages obtained as fragments of $\mathscr{L}$.

The non-standard notation introduced below has been chosen to emphasise the parameter $\mathcal{E}$, because it plays an important role in the definition of the open extension and compatible refinement



(a) $\mathscr{S}(\mathsf{P}_\bot(\mathsf{P}_\bot(\mathsf{unit})))$ (b) $\mathscr{L}_0(\mathsf{P}_\bot(\mathsf{P}_\bot(\mathsf{unit})))$

Figure 5.3: Equivalence classes in $\mathscr{S}$ and $\mathscr{L}_0$ at $\mathsf{P}_\bot(\mathsf{P}_\bot(\mathsf{unit}))$ w.r.t. $\lesssim^{\mathscr{S}}_{\mathrm{CS}}$ and $\lesssim^{\mathscr{L}_0}_{\mathrm{CS}}$

operators by restricting universal quantifications to $\mathscr{L}(\mathcal{E})$. Most of the proofs in this section are brief or omitted altogether because they are straightforward or appear in the literature. However, care is required because some usual properties fail for certain choices of $\mathcal{E}$ (see example 5.3.7).

To accommodate the definition of open extension in the presence of the empty type $\mathsf{sum}\,\langle\rangle$, the space of relations on open terms must include information about the environments in which terms are related and must not enforce strengthening (see exercise 3.6.5 of [Cro93]). In addition, definition 5.3.1 forces relations on open terms to be closed under weakening and renaming of variables in the environment.

Definition 5.3.1 The set $Rel(\mathcal{E})$ consists of all sets of triples:

$$\mathcal{R} \subseteq \{\Gamma \mid \Gamma \text{ an environment}\} \times \mathscr{L}(\mathcal{E}) \times \mathscr{L}(\mathcal{E})$$

such that:

1. $\langle \Gamma, M, N\rangle \in \mathcal{R}$ implies there exists a (necessarily unique) type σ such that $\Gamma \vdash M : \sigma$ and $\Gamma \vdash N : \sigma$.
2. $\langle \Gamma, M, N\rangle \in \mathcal{R}$ and $\Gamma \subseteq \Delta$ implies $\langle \Delta, M, N\rangle \in \mathcal{R}$.
3. $\langle (\Gamma, x:\sigma), M, N\rangle \in \mathcal{R}$ and $y \notin Dom(\Gamma)$ implies $\langle (\Gamma, y:\sigma), M[y/x], N[y/x]\rangle \in \mathcal{R}$.

The complete lattice $\langle Rel(\mathcal{E}), \subseteq\rangle$ has meets defined as intersection for non-empty sets. The meet of the empty set is:

$$\{\langle \Gamma, M, N\rangle \mid \exists \sigma.\, \Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \sigma \wedge \Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \sigma\} \in Rel(\mathcal{E})$$

We write $Rel(M_1, \ldots, M_n)$ for $Rel(\{M_1, \ldots, M_n\})$. When $\mathcal{R} \in Rel(\mathcal{E})$, we write $\Gamma \vdash M\,\mathcal{R}\,N : \sigma$ or $\Gamma \vdash \langle M, N\rangle \in \mathcal{R} : \sigma$ for $\Gamma \vdash M : \sigma$, $\Gamma \vdash N : \sigma$, and $\langle \Gamma, M, N\rangle \in \mathcal{R}$.

The dual $\mathcal{R}^{\mathrm{op}} \in Rel(\mathcal{E})$ of a relation $\mathcal{R} \in Rel(\mathcal{E})$ is defined by:

$$\langle \Gamma, M, N\rangle \in \mathcal{R}^{\mathrm{op}} \iff \langle \Gamma, N, M\rangle \in \mathcal{R}$$

This induces a monotone function with respect to inclusion on $Rel(\mathcal{E})$.

Relations on open terms $\mathcal{R}, \mathcal{S} \in Rel(\mathcal{E})$ can be composed as follows. For terms L, N and an environment Γ:

$$\langle \Gamma, L, N\rangle \in \mathcal{R};\mathcal{S} \iff \exists M.\, \langle \Gamma, L, M\rangle \in \mathcal{R} \wedge \langle \Gamma, M, N\rangle \in \mathcal{S}$$

The identity relation $Id(\mathscr{L}(\mathcal{E})) \in Rel(\mathcal{E})$ is an identity for composition.

$$Id(\mathscr{L}(\mathcal{E})) \stackrel{\text{def}}{=} \{\langle \Gamma, M, M\rangle \mid \exists \sigma.\, \Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \sigma\}$$

In addition, composition is associative, and monotone with respect to (point wise) inclusion. The properties of reflexivity, symmetry, and transitivity are defined upon elements of $Rel(\mathcal{E})$ in the usual way in terms of inclusion, the identity, and the dual and composition operators.

The closed restriction and open extension operators are functions between the relations on closed terms $Rel_0(\mathcal{E})$ and the relations on open terms $Rel(\mathcal{E})$. Relational substitution is useful for describing properties of Howe's *congruence candidate* (see section 5.4).

Definition 5.3.2

1. For $\mathcal{R} \in Rel(\mathcal{E})$, the ***closed restriction*** of $\mathcal{R}$, $Cls(\mathcal{R}) \in Rel_0(\mathcal{E})$, is defined by:

$$Cls(\mathcal{R}) \stackrel{\text{def}}{=} \{\langle M,N\rangle \mid \langle \emptyset, M,N\rangle \in \mathcal{R}\}$$

2. For $\mathcal{R} \in Rel_0(\mathcal{E})$, the ***open extension*** of $\mathcal{R}$ with respect to $\mathcal{E}$, $Opn(\mathcal{E},\mathcal{R}) \in Rel(\mathcal{E})$, is defined, for terms M, N and an environment $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$, by:

$$\begin{aligned}
&\Gamma \vdash \langle M,N\rangle \in Opn(\mathcal{E},\mathcal{R}) : \sigma \Longleftrightarrow \\
&\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \sigma \wedge \\
&\Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \sigma \wedge \\
&\forall L_1 \in \mathscr{L}_0(\mathcal{E})(\sigma_1), \ldots, L_n \in \mathscr{L}_0(\mathcal{E})(\sigma_n).\ \vdash \langle M[\vec{L}/\vec{x}], N[\vec{L}/\vec{x}]\rangle \in \mathcal{R} : \sigma
\end{aligned}$$

where $\vec{L} = L_1, \ldots, L_n$.

3. For $\mathcal{R},\mathcal{S} \in Rel(\mathcal{E})$, the ***relational substitution*** of $\mathcal{S}$ into $\mathcal{R}$, written $\mathcal{R}[\mathcal{S}] \in Rel(\mathcal{E})$, is defined, for terms L_1, L_2 and an environment Γ, by:

$$\begin{aligned}
&\Gamma \vdash \langle L_1, L_2\rangle \in \mathcal{R}[\mathcal{S}] : \tau \Longleftrightarrow \\
&\exists M, N, \vec{M}, \vec{N}. \\
&\qquad L_1 = M[\vec{M}/\vec{x}] \wedge L_2 = N[\vec{N}/\vec{x}] \wedge \\
&\qquad \Gamma, \Delta \vdash \langle M,N\rangle \in \mathcal{R} : \tau \wedge \\
&\qquad \forall 1 \le i \le n. \Gamma \vdash \langle M_i, N_i\rangle \in \mathcal{S} : \sigma_i
\end{aligned}$$

where $\Delta = x_1 : \sigma_1, \ldots, x_n : \sigma_n$, $\vec{M} = M_1, \ldots, M_n$, and $\vec{N} = N_1, \ldots, N_n$.

It can be verified that the open extension and relational substitution are always elements of $Rel(\mathcal{E})$. In addition, all three operators are monotone with respect to inclusion.

Lemma 5.3.3

1. If $\mathcal{R},\mathcal{S} \in Rel(\mathcal{E})$ and $\mathcal{R} \subseteq \mathcal{S}$ then $Cls(\mathcal{R}) \subseteq Cls(\mathcal{S})$.

2. If $\mathcal{R},\mathcal{S} \in Rel_0(\mathcal{E})$ and $\mathcal{R} \subseteq \mathcal{S}$ then $Opn(\mathcal{E},\mathcal{R}) \subseteq Opn(\mathcal{E},\mathcal{S})$.

3. If $\mathcal{R}_1,\mathcal{R}_2,\mathcal{S}_1,\mathcal{S}_2 \in Rel(\mathcal{E})$, $\mathcal{R}_1 \subseteq \mathcal{S}_1$, and $\mathcal{R}_2 \subseteq \mathcal{S}_2$, then $\mathcal{R}_1[\mathcal{R}_2] \subseteq \mathcal{S}_1[\mathcal{S}_2]$.

Proof Straightforward. □

The operators also satisfy the following basic properties that are analogous to results in [Las98b].

Lemma 5.3.4

1. If $\mathcal{R} \in Rel_0(\mathcal{E})$ and $Id(\mathscr{L}_0(\mathcal{E})) \subseteq \mathcal{R}$, then $Id(\mathscr{L}(\mathcal{E})) \subseteq Opn(\mathcal{E},\mathcal{R})$.
2. If $\mathcal{R} \in Rel_0(\mathcal{E})$ and $Id(\mathscr{L}_0(\mathcal{E})) \subseteq \mathcal{R}$, then $Opn(\mathcal{E},\mathcal{R})[Id(\mathscr{L}(\mathcal{E}))] = Opn(\mathcal{E},\mathcal{R})$.
3. If $\mathcal{R},\mathcal{S} \in Rel_0(\mathcal{E})$, then $Opn(\mathcal{E},\mathcal{R});Opn(\mathcal{E},\mathcal{S}) \subseteq Opn(\mathcal{E},\mathcal{R};\mathcal{S})$.
4. If $\mathcal{R}_1,\mathcal{R}_2,\mathcal{S}_1,\mathcal{S}_2 \in Rel(\mathcal{E})$, then $(\mathcal{R}_1;\mathcal{R}_2)[\mathcal{S}_1;\mathcal{S}_2] \subseteq \mathcal{R}_1[\mathcal{S}_1];\mathcal{R}_2[\mathcal{S}_2]$.
5. If $\mathcal{R} \in Rel(\mathcal{E})$ and $\mathcal{S} \in Rel_0(\mathcal{E})$, then $\mathcal{R} \subseteq Opn(\mathcal{E},\mathcal{S})$ if and only if $Cls(\mathcal{R}[Id(\mathscr{L}(\mathcal{E}))]) \subseteq \mathcal{S}$.

Proof Straightforward. (2) makes use of lemma 3.3.4. □

Howe [How89] introduces a now standard *compatible refinement* operator on relations between terms in order to prove compatibility of applicative similarity (see section 5.4). Terms are related by the compatible refinement of $\mathcal{R} \in Rel(\mathcal{E})$ if they have the same outermost constructor and the immediate subterms are pointwise related by $\mathcal{R}$.

Definition 5.3.5 The ***compatible refinement*** $Cmp(\mathcal{E},\mathcal{R}) \in Rel(\mathcal{E})$ of a relation $\mathcal{R} \in Rel(\mathcal{E})$ with respect to $\mathcal{E}$ is the least set closed under the rules of figure 5.4.

Note that the notation used here for the compatible refinement $Cmp(\mathcal{E},\mathcal{R})$ differs from the standard notation $\widehat{\mathcal{R}}$.

Lemma 5.3.6 For $\mathcal{R},\mathcal{S} \in Rel(\mathcal{E})$:

1. If $\mathcal{R} \subseteq \mathcal{S}$, then $Cmp(\mathcal{E},\mathcal{R}) \subseteq Cmp(\mathcal{E},\mathcal{S})$.
2. If $Id(\mathscr{L}(\mathcal{E})) \subseteq \mathcal{R}$, then $Id(\mathscr{L}(\mathcal{E})) \subseteq Cmp(\mathcal{E},\mathcal{R})$.
3. If $\mathcal{R} \subseteq Id(\mathscr{L}(\mathcal{E}))$, then $Cmp(\mathcal{E},\mathcal{R}) \subseteq Id(\mathscr{L}(\mathcal{E}))$.
4. $Cmp(\mathcal{E},\mathcal{R});Cmp(\mathcal{E},\mathcal{S}) \subseteq Cmp(\mathcal{E},\mathcal{R};\mathcal{S})$
5. $Cmp(\mathcal{E},\mathcal{R})[\mathcal{S}] \subseteq \mathcal{S} \cup Cmp(\mathcal{E},\mathcal{R}[\mathcal{S}])$
6. $Cmp(\mathcal{E},\mathcal{R}^{\mathrm{op}}) = (Cmp(\mathcal{E},\mathcal{R}))^{\mathrm{op}}$

$$\Gamma \vdash \langle x, x \rangle \in \mathit{Cmp}(\mathcal{E}, \mathcal{R}) : \sigma \quad (\Gamma(x) = \sigma)$$

$$\frac{\Gamma \vdash \langle M, N \rangle \in \mathcal{R} : \sigma_m}{\Gamma \vdash \langle \mathsf{inj}\, m\, \mathsf{of}\, M, \mathsf{inj}\, m\, \mathsf{of}\, N \rangle \in \mathit{Cmp}(\mathcal{E}, \mathcal{R}) : \mathsf{sum}\, \langle \sigma_n \mid n < \kappa \rangle}$$

$$\frac{\Gamma \vdash \langle L_1, L_2 \rangle \in \mathcal{R} : \mathsf{sum}\, \langle \sigma_n \mid n < \kappa \rangle \qquad \{\Gamma, x_n : \sigma_n \vdash \langle M_n, N_n \rangle \in \mathcal{R} : \tau \mid n < \kappa\}}{\Gamma \vdash \langle \mathsf{case}\, L_1\, \mathsf{of}\, \langle x_n . M_n \mid n < \kappa \rangle, \mathsf{case}\, L_2\, \mathsf{of}\, \langle x_n . N_n \mid n < \kappa \rangle \rangle \in \mathit{Cmp}(\mathcal{E}, \mathcal{R}) : \tau}$$

$$\frac{\{\Gamma \vdash \langle M_n, N_n \rangle \in \mathcal{R} : \sigma_n \mid n < \kappa\}}{\Gamma \vdash \langle \mathsf{tuple}\, \langle M_n \mid n < \kappa \rangle, \mathsf{tuple}\, \langle N_n \mid n < \kappa \rangle \rangle \in \mathit{Cmp}(\mathcal{E}, \mathcal{R}) : \mathsf{prod}\, \langle \sigma_n \mid n < \kappa \rangle}$$

$$\frac{\Gamma \vdash \langle M, N \rangle \in \mathcal{R} : \mathsf{prod}\, \langle \sigma_n \mid n < \kappa \rangle}{\Gamma \vdash \langle \mathsf{proj}\, m\, \mathsf{of}\, M, \mathsf{proj}\, m\, \mathsf{of}\, N \rangle \in \mathit{Cmp}(\mathcal{E}, \mathcal{R}) : \sigma_m}$$

$$\frac{\Gamma, x : \sigma \vdash \langle M, N \rangle \in \mathcal{R} : \tau}{\Gamma \vdash \langle \lambda x{:}\sigma. M, \lambda x{:}\sigma. N \rangle \in \mathit{Cmp}(\mathcal{E}, \mathcal{R}) : \sigma \to \tau}$$

$$\frac{\Gamma \vdash \langle M_1, N_1 \rangle \in \mathcal{R} : \sigma \to \tau \qquad \Gamma \vdash \langle M_2, N_2 \rangle \in \mathcal{R} : \sigma}{\Gamma \vdash \langle M_1 M_2, N_1 N_2 \rangle \in \mathit{Cmp}(\mathcal{E}, \mathcal{R}) : \tau}$$

$$\frac{\Gamma \vdash \langle M, N \rangle \in \mathcal{R} : \sigma}{\Gamma \vdash \langle [M], [N] \rangle \in \mathit{Cmp}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash \langle M_1, N_1 \rangle \in \mathcal{R} : \mathsf{P}_\perp(\sigma) \qquad \Gamma, x : \sigma \vdash \langle M_2, N_2 \rangle \in \mathcal{R} : \mathsf{P}_\perp(\tau)}{\Gamma \vdash \langle \mathsf{let}\, x{:}\sigma \Leftarrow M_1\, \mathsf{in}\, M_2, \mathsf{let}\, x{:}\sigma \Leftarrow N_1\, \mathsf{in}\, N_2 \rangle \in \mathit{Cmp}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\tau)}$$

$$\frac{\Gamma, x : \mathsf{P}_\perp(\sigma) \vdash \langle M, N \rangle \in \mathcal{R} : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \langle \mathsf{fix}\, x{:}\mathsf{P}_\perp(\sigma). M, \mathsf{fix}\, x{:}\mathsf{P}_\perp(\sigma). N \rangle \in \mathit{Cmp}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\{\Gamma \vdash \langle M_n, N_n \rangle \in \mathcal{R} : \sigma \mid n < \kappa\}}{\Gamma \vdash \langle ?\langle M_n \mid n < \kappa \rangle, ?\langle N_n \mid n < \kappa \rangle \rangle \in \mathit{Cmp}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)}$$

In addition, each rule schema has implicit side conditions $\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \sigma$ and $\Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \sigma$ whenever the conclusion is $\Gamma \vdash \langle M, N \rangle \in \mathit{Cmp}(\mathcal{E}, \mathcal{R}) : \sigma$.

Figure 5.4: Compatible refinement

Proof (1)-(3) are straightforward case analyses. For (4), the conclusions of the compatible refinement schema do not overlap, so if $\langle L,M\rangle \in Cmp(\mathcal{E},\mathcal{R})$ and $\langle M,N\rangle \in Cmp(\mathcal{E},\mathcal{R})$ then both are instances of just one rule schema, and L, M, N have the same outermost constructor. (5) is a case analysis on terms M, N related by $Cmp(\mathcal{E},\mathcal{R})$. If M and N are the same variable, then the results of substituting $\mathcal{S}$-related terms into M and N are related by S. Otherwise the results are related by $Cmp(\mathcal{E},\mathcal{R}[\mathcal{S}])$. (6) follows easily from the definition of compatible refinement. □

Example 5.3.7 The inclusion that is the opposite of lemma 5.3.6(4):

$$Cmp(\mathcal{E},\mathcal{R};\mathcal{S}) \subseteq Cmp(\mathcal{E},\mathcal{R});Cmp(\mathcal{E},\mathcal{S})$$

does not hold for arbitrary fragments, including some of the fragments that we are most interested in. For example, consider $\mathcal{E} = \{?\langle \underline{0},\underline{1}\rangle\}$ and $\mathcal{R} = Opn(\mathcal{E},\{\langle \underline{0},\underline{2}\rangle,\langle \underline{1},\underline{3}\rangle\}) \in Rel(\mathcal{E})$. Then, because $\mathcal{R};\mathcal{R}^{\mathrm{op}} = Opn(\mathcal{E},\{\langle \underline{0},\underline{0}\rangle,\langle \underline{1},\underline{1}\rangle\})$, we have:

$$\vdash \langle ?\langle \underline{0},\underline{1}\rangle, ?\langle \underline{0},\underline{1}\rangle\rangle \in Cmp(\mathcal{E},\mathcal{R};\mathcal{R}^{\mathrm{op}}) : \mathsf{P}_{\perp}(\mathsf{nat})$$

However, because $?\langle \underline{2},\underline{3}\rangle \notin \mathscr{L}(\mathcal{E})$:

$$\langle ?\langle \underline{0},\underline{1}\rangle, ?\langle \underline{0},\underline{1}\rangle\rangle \notin Cmp(\mathcal{E},\mathcal{R});Cmp(\mathcal{E},\mathcal{R}^{\mathrm{op}})$$

The inclusion is desirable because it greatly simplifies the proof of compatibility for variants of bisimilarity. An alternative is proposed in section 5.4.

Analogous inclusions do hold in other settings when there are no inconsistencies concerning term formation as there are in proper fragments of $\mathscr{L}$ such as $\mathscr{L}(?\langle \underline{0},\underline{1}\rangle)$. One way to resolve this problem is to impose stronger closure conditions upon fragments. For example, the smallest fragment $\mathscr{L}(?\langle \underline{0},\underline{1}\rangle)$ containing $?\langle \underline{0},\underline{1}\rangle$ can be required to contain every term of the form $?\langle M,N\rangle$ when $\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{nat}$ and $\Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \mathsf{nat}$. However, it is difficult to extend this approach in a way that permits fragments containing $?\underline{\omega}$ but not $?\underline{A}$, where $A \subseteq_{\mathrm{ne}} \omega$ is a non-recursively enumerable set.

Alternatively, the term $?\langle \underline{0},\underline{1}\rangle$ can be treated as a distinct term constructor of arity 0, and compatible refinement can be extended with a new rule schema (with no premises):

$$\Gamma \vdash \langle ?\langle \underline{0},\underline{1}\rangle, ?\langle \underline{0},\underline{1}\rangle\rangle \in Cmp(\mathcal{E},\mathcal{R}) : \mathsf{P}_{\perp}(\mathsf{nat})$$

Unfortunately, this approach becomes complex when a fragment does contain every term of the form $?\langle M,N\rangle$, where $\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{nat}$ and $\Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \mathsf{nat}$, because then the original rule schema for compatible refinement should be used.

Howe's compatibility proof relies on a relation, the *congruence candidate*, that is the least fixed-point of a function defined in terms of the compatible refinement operator. The form of the function ensures that the least fixed-point is the unique fixed-point, and hence it is also the greatest fixed-point. This fact is recorded in the following lemma, along with an easy corollary stating that the identity relation is the unique fixed-point of compatible refinement. In section 5.4 it is used to rephrase Howe's argument as a proof by coinduction that avoids explicit mention of terms.

Lemma 5.3.8

1. For a relation $\mathcal{R} \in Rel(\mathcal{E})$:

$$\mu\mathcal{S}.\,Cmp(\mathcal{E},\mathcal{S});\mathcal{R} = \nu\mathcal{S}.\,Cmp(\mathcal{E},\mathcal{S});\mathcal{R}$$

2. The identity relation on the fragment generated by $\mathcal{E}$ is both a least and a greatest fixed-point, i.e.:

$$Id(\mathscr{L}(\mathcal{E})) = \mu\mathcal{S}.\,Cmp(\mathcal{E},\mathcal{S}) = \nu\mathcal{S}.\,Cmp(\mathcal{E},\mathcal{S})$$

Proof

1. Define a monotone function $F : Rel(\mathcal{E}) \to Rel(\mathcal{E})$ by $F(S) \stackrel{\text{def}}{=} Cmp(\mathcal{E},\mathcal{S});\mathcal{R}$. Using lemma 2.3.11, it suffices to exhibit a well-founded relation $< \subseteq (\nu S.F(S)) \times (\nu S.F(S))$ such that, for all $\langle\Gamma,M_1,N_1\rangle \in \nu S.F(S)$, there exists $T \subseteq \nu S.F(S)$ such that $\langle\Gamma,M_1,N_1\rangle \in F(T)$ and $T < \langle\Gamma,M_1,N_1\rangle$, i.e., for all $\langle\Delta,M_2,N_2\rangle \in T$, $\langle\Delta,M_2,N_2\rangle < \langle\Gamma,M_1,N_1\rangle$. Define the well-founded relation for all triples by $\langle\Delta,M_2,N_2\rangle < \langle\Gamma,M_1,N_1\rangle$ if and only if M_2 is a proper subterm of M_1. For $\langle\Gamma,M_1,N_1\rangle \in \nu S.F(S)$, set:

$$T \stackrel{\text{def}}{=} \{\langle\Delta,M_2,N_2\rangle \mid \langle\Delta,M_2,N_2\rangle < \langle\Gamma,M_1,N_1\rangle\}$$

   We want to show $\langle\Gamma,M_1,N_1\rangle \in F(T)$. We have that:

$$\langle\Gamma,M_1,N_1\rangle \in \nu S.F(S) = F(\nu S.F(S)) = Cmp(\mathcal{E},\nu S.F(S));\mathcal{R}$$

   Thus there exists L_1 such that $\langle\Gamma,M_1,L_1\rangle \in Cmp(\mathcal{E},\nu S.F(S))$ and $\langle\Gamma,L_1,N_1\rangle \in \mathcal{R}$. However, $\langle\Gamma,M_1,L_1\rangle \in Cmp(\mathcal{E},\mathcal{T})$, because the terms in the premises of every rule schema for compatible refinement are always proper subterms of the terms in the conclusion. Therefore $\langle\Gamma,M_1,N_1\rangle \in F(\mathcal{T})$. By lemma 2.3.11, we conclude that the least and greatest fixed points of F coincide.

2. By lemma 5.3.6(2,3), $Id(\mathscr{L}(\mathcal{E}))$ is a fixed-point of $Cmp(\mathcal{E},\cdot)$. The result follows from (1) by taking $\mathcal{R} = Id(\mathscr{L}(\mathcal{E}))$.

□

Lemma 5.3.9 expresses the behaviour of $\langle\mathcal{R}\rangle_{\text{LS}}^{\mathscr{L}_0(\mathcal{E})}$ and $\langle\mathcal{R}\rangle_{\text{US}}^{\mathscr{L}_0(\mathcal{E})}$ in terms of compatible refinement, open extension, and may and must convergence. The compatible refinement of the open extension of $\mathcal{R}$ replaces the type specific definitions found in figures 5.1 and 5.2.

Lemma 5.3.9 For a relation $\mathcal{R} \in Rel_0(\mathcal{E})$ and programs $M,N \in \mathscr{L}_0(\mathcal{E})(\sigma)$:

1. $\vdash \langle M,N\rangle \in \langle\mathcal{R}\rangle_{\text{LS}}^{\mathscr{L}_0(\mathcal{E})} : \sigma \Longleftrightarrow$
   $\forall K_1.\, M \Downarrow^{\text{may}} K_1 \Longrightarrow \exists K_2.\, N \Downarrow^{\text{may}} K_2 \wedge \vdash \langle K_1,K_2\rangle \in Cmp(\mathcal{E},Opn(\mathcal{E},\mathcal{R})) : \sigma$

2. $\vdash \langle M,N\rangle \in \langle \mathcal{R}\rangle^{\mathscr{L}_0(\mathcal{E})}_{\text{US}} : \sigma \Longleftrightarrow$
   $M \Downarrow^{\text{must}} \Longrightarrow$
   $(N \Downarrow^{\text{must}} \wedge$
   $\forall K_2. N \Downarrow^{\text{may}} K_2 \Longrightarrow \exists K_1. M \Downarrow^{\text{may}} K_1 \wedge \vdash \langle K_1,K_2\rangle \in \mathit{Cmp}(\mathcal{E},\mathit{Opn}(\mathcal{E},\mathcal{R})) : \sigma)$

Proof Both (1) and (2) are proven by case analysis of the type σ. The cases for value types are the same for (1) and (2) because:

$$\vdash \langle M,N\rangle \in \langle \mathcal{R}\rangle^{\mathscr{L}_0(\mathcal{E})}_{\text{LS}} : \sigma \Longleftrightarrow \vdash \langle M,N\rangle \in \langle \mathcal{R}\rangle^{\mathscr{L}_0(\mathcal{E})}_{\text{US}} : \sigma$$

The case for computation types is straightforward. We illustrate with the forward direction of the case for function types. Suppose that $\vdash \langle M,N\rangle \in \langle \mathcal{R}\rangle^{\mathscr{L}_0(\mathcal{E})}_{\text{LS}} : \sigma \to \tau$ and there is a term M_1 such that $M \Downarrow^{\text{may}} \lambda x.M_1$. The term N has a value type and so always converges to a unique term $\lambda x.N_1$, for some N_1. To prove $\vdash \langle \lambda x.M_1, \lambda x.N_1\rangle \in \mathit{Cmp}(\mathcal{E},\mathit{Opn}(\mathcal{E},\mathcal{R})) : \sigma \to \tau$ it suffices to show $x:\sigma \vdash \langle M_1,N_1\rangle \in \mathit{Opn}(\mathcal{E},\mathcal{R}) : \tau$, i.e., for all $L \in \mathscr{L}_0(\mathcal{E})(\sigma)$, $\vdash \langle M_1[L/x],N_1[L/x]\rangle \in \mathcal{R} : \tau$. However, $M \Downarrow^{\text{may}} \lambda x.M_1$ implies $M \xrightarrow{@L} M_1[L/x]$. From $\langle M,N\rangle \in \langle \mathcal{R}\rangle^{\mathscr{L}_0(\mathcal{E})}_{\text{LS}}$, we have that $N \xrightarrow{@L} N@L$ and $\langle M_1[L/x],N@L\rangle \in \mathcal{R}$, but $N@L = N_1[L/x]$ because $N \Downarrow^{\text{may}} \lambda x.N_1$. Therefore $\vdash \langle M_1[L/x],N_1[L/x]\rangle \in \mathcal{R} : \tau$, as required. □

5.4 Compatibility

This section establishes compatibility of the open extensions of the variants of similarity, mutual similarity, and bisimilarity. Lower, upper, and convex similarity are dealt with first. Next, a new idea is used for lower, upper, and convex similarity bisimilarity. Finally, we introduce a second novel technique for refinement similarity.

We start by defining what it means for a relation on open terms to be compatible.

Definition 5.4.1 A relation on open terms $\mathcal{R} \in \mathit{Rel}(\mathcal{E})$ is ***compatible*** with respect to $\mathcal{E}$ if $\mathit{Cmp}(\mathcal{E},\mathcal{R}) \subseteq \mathcal{R}$.

It is easier to prove that terms are related by an equivalence relation (often capturing a notion of behavioural equality) if equivalent terms can be used interchangeably as subterms of a larger term. Compatibility ensures that such compositional reasoning is permissible. From a different perspective, compatibility can be used to show that term constructors induce well-defined functions on equivalence classes of terms, which is useful, if not essential, for constructing denotational models. Applicative compatibility serves a similar role for TTSs (recall that a relation must be applicative compatible for the extensional collapse of definition 4.5.5 to exist). Lemma 5.4.2 shows that compatibility implies applicative compatibility.

Lemma 5.4.2 Let $\mathcal{R} \in \mathit{Rel}_0(\mathcal{E})$ be a variant of similarity, mutual similarity, or bisimilarity. If $\mathit{Opn}(\mathcal{E},\mathcal{R})$ is compatible with respect to $\mathcal{E}$, then $\mathcal{R}$ is applicatively compatible upon the TTS $\mathscr{L}_0(\mathcal{E})$.

Proof Consider types σ, τ and programs $L \in \mathscr{L}_0(\mathcal{E})(\sigma \to \tau)$ and $M,N \in \mathscr{L}_0(\mathcal{E})(\sigma)$ such that $\langle M,N\rangle \in \mathcal{R}$. We want to show $\langle L@M, L@N\rangle \in \mathcal{R}$. With lemma 5.1.3 it can be shown that $L@M \simeq_{\mathrm{CB}}^{\mathscr{L}_0(\mathcal{E})} LM$ and $LN \simeq_{\mathrm{CB}}^{\mathscr{L}_0(\mathcal{E})} L@N$. We have $\vdash \langle L,L\rangle \in Opn(\mathcal{E},\mathcal{R}) : \sigma \to \tau$, because $Id(\mathscr{L}(\mathcal{E})) \subseteq Opn(\mathcal{E},\mathcal{R})$, and $\vdash \langle M,N\rangle \in Opn(\mathcal{E},\mathcal{R}) : \sigma$. Compatibility of $Opn(\mathcal{E},\mathcal{R})$ implies that $\vdash \langle LM, LN\rangle \in Opn(\mathcal{E},\mathcal{R}) : \sigma$, and so $\langle LM, LN\rangle \in \mathcal{R}$. Convex bisimilarity is the finest of the variants of similarity, mutual similarity, and bisimilarity, and each one is transitive, so $\simeq_{\mathrm{CB}}^{\mathscr{L}_0(\mathcal{E})};\mathcal{R};\simeq_{\mathrm{CB}}^{\mathscr{L}_0(\mathcal{E})} \subseteq \mathcal{R}$. Therefore $\langle L@M, L@N\rangle \in \mathcal{R}$. $\square$

It is non-trivial to prove that the open extensions of the variants of similarity, mutual similarity, and bisimilarity are compatible, because substitution is used in the operational semantics yet substituting related terms into related terms may produce unrelated terms. In other words, if $\mathcal{R} \in Rel_0(\mathcal{E})$ is the variant of similarity, mutual similarity, and bisimilarity then we do not know that:

$$Opn(\mathcal{E},\mathcal{R})[Opn(\mathcal{E},\mathcal{R})] \subseteq Opn(\mathcal{E},\mathcal{R})$$

There are two well known methods for establishing compatibility. Abramsky [Abr90] proves that applicative similarity (lower similarity in the terminology used here) for the lazy λ-calculus is compatible using domain-theoretic methods, and offers the following challenge:

> Our proof will make essential use of domain logic, despite the fact that the *statement* of the result does not mention domains at all. The reader who may be sceptical of our approach is invited to attempt a direct proof.

Howe [How89] responds with a syntactic proof involving inductions on terms and operational semantics derivations. The compatibility result applies to lower similarity upon a family of languages including the lazy λ-calculus and non-deterministic λ-calculi.

Howe's method makes use of a compatible relation, called the *congruence candidate*, that does satisfy the relational substitution closure property above.

Definition 5.4.3 (Howe) The ***congruence candidate*** $Cand(\mathcal{E},\mathcal{R}) \in Rel(\mathcal{E})$ of a relation on programs $\mathcal{R} \in Rel_0(\mathcal{E})$ with respect to $\mathcal{E}$ is defined by:

$$Cand(\mathcal{E},\mathcal{R}) \stackrel{\mathrm{def}}{=} \mu\mathcal{S}.Cmp(\mathcal{E},\mathcal{S});Opn(\mathcal{E},\mathcal{R})$$

The congruence candidate contains the open extension of the original variant of similarity, mutual similarity, or bisimilarity by construction. Coinduction can then be used to show that the open extension contains the congruence candidate and so the open extension is also compatible.

Ong [Ong92a] extends Howe's method to convex similarity for a non-deterministic λ-calculus, and the same technique would apply to upper similarity. Howe [How96] independently uses an extension that is similar to Ong's. In addition, Howe introduces another idea to show that variants of bisimilarity are compatible, and then proves that lower similarity, lower bisimilarity, and convex bisimilarity are compatible, but the techniques also apply to the lower, upper, and convex variants of similarity, mutual similarity, and bisimilarity. However, both Ong and Howe's arguments for upper and convex variants require that the programming language exhibits only finite non-determinism, because the must convergence rank of a program is assumed to be finite.

Lassen and the author [Las97, Las98b, LP98] independently observe that Ong and Howe's arguments can be modified for infinite erratic non-determinism by rephrasing the definition of must convergence and applying well-founded induction.

The techniques discussed above can be applied to the lower, upper, and convex variants of similarity and mutual similarity upon the language fragments, but not to the variants of bisimilarity. We prove compatibility of the variants of bisimilarity for a restricted collection of language fragments by modifying the standard arguments. In addition, a new technique is introduced to prove the compatibility of refinement similarity and mutual refinement similarity, again for a restricted collection of language fragments, in response to a question posed by Lassen.

Lower, Upper, and Convex Similarity

We first examine properties of the congruence candidate. By an earlier result, the congruence candidate is a unique fixed-point because of the placement of the compatible refinement operator, and the fact that terms in the premises of each compatible refinement rule are strictly smaller than terms in the conclusion. This means that coinduction can be used to reason about the congruence candidate, serving the same role as an induction on the left-hand term of a pair of terms related by the congruence candidate.

Lemma 5.4.4 The congruence candidate is also the greatest fixed-point:

$$Cand(\mathcal{E},\mathcal{R}) = \nu\mathcal{S}.Cmp(\mathcal{E},\mathcal{S});Opn(\mathcal{E},\mathcal{R})$$

Proof Apply lemma 5.3.8(1). □

Lemma 5.4.5 proves analogues of results in Howe's original paper [How89].

Lemma 5.4.5 For a preorder $\mathcal{R} \in Rel_0(\mathcal{E})$:

1. $Id(\mathscr{L}(\mathcal{E})) \subseteq Cand(\mathcal{E},\mathcal{R})$
2. $Opn(\mathcal{E},\mathcal{R}) \subseteq Cand(\mathcal{E},\mathcal{R})$
3. $Cand(\mathcal{E},\mathcal{R});Opn(\mathcal{E},\mathcal{R}) = Cand(\mathcal{E},\mathcal{R})$
4. $Cmp(\mathcal{E},Cand(\mathcal{E},\mathcal{R})) \subseteq Cand(\mathcal{E},\mathcal{R})$
5. $Cand(\mathcal{E},\mathcal{R})[Cand(\mathcal{E},\mathcal{R})] \subseteq Cand(\mathcal{E},\mathcal{R})$

Proof

1. By coinduction it suffices to show $Id(\mathscr{L}(\mathcal{E})) \subseteq Cmp(\mathcal{E}, Id(\mathscr{L}(\mathcal{E}))); Opn(\mathcal{E}, \mathcal{R})$. This follows from lemmas 5.3.6(2) and 5.3.4(1).

2. By coinduction it suffices to show $Opn(\mathcal{E}, \mathcal{R}) \subseteq Cmp(\mathcal{E}, Opn(\mathcal{E}, \mathcal{R})); Opn(\mathcal{E}, \mathcal{R})$. This follows from lemmas 5.3.6(2), 5.3.4(1), and 5.3.6(1):

$$\begin{array}{cl} & Opn(\mathcal{E}, \mathcal{R}) \\ = & Id(\mathscr{L}(\mathcal{E})); Opn(\mathcal{E}, \mathcal{R}) \\ = & Cmp(\mathcal{E}, Id(\mathscr{L}(\mathcal{E}))); Opn(\mathcal{E}, \mathcal{R}) \\ \subseteq & Cmp(\mathcal{E}, Opn(\mathcal{E}, \mathcal{R})); Opn(\mathcal{E}, \mathcal{R}) \end{array}$$

3. Use the transitivity of $Opn(\mathcal{E}, \mathcal{R})$ and the fact that $Cand(\mathcal{E}, \mathcal{R})$ is a fixed-point:

$$\begin{array}{cl} & Cand(\mathcal{E}, \mathcal{R}); Opn(\mathcal{E}, \mathcal{R}) \\ = & (Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})); Opn(\mathcal{E}, \mathcal{R})); Opn(\mathcal{E}, \mathcal{R}) \\ = & Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})); Opn(\mathcal{E}, \mathcal{R}) \\ = & Cand(\mathcal{E}, \mathcal{R}) \end{array}$$

4. Use the reflexivity of $Opn(\mathcal{E}, \mathcal{R})$ and the fact that $Cand(\mathcal{E}, \mathcal{R})$ is a fixed-point:

$$\begin{array}{cl} & Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})) \\ = & Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})); Id(\mathscr{L}(\mathcal{E})) \\ \subseteq & Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})); Opn(\mathcal{E}, \mathcal{R}) \\ = & Cand(\mathcal{E}, \mathcal{R}) \end{array}$$

5. By strong coinduction (see lemma 2.3.4) it suffices to show:

$$\begin{array}{cl} & Cand(\mathcal{E}, \mathcal{R})[Cand(\mathcal{E}, \mathcal{R})] \\ \subseteq & Cand(\mathcal{E}, \mathcal{R}) \cup Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})[Cand(\mathcal{E}, \mathcal{R})]); Opn(\mathcal{E}, \mathcal{R}) \end{array}$$

   This is proven by:

$$\begin{array}{cll} & Cand(\mathcal{E}, \mathcal{R})[Cand(\mathcal{E}, \mathcal{R})] & \\ = & (Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})); Opn(\mathcal{E}, \mathcal{R}))[(Cand(\mathcal{E}, \mathcal{R}); Id(\mathscr{L}(\mathcal{E})))] & \\ \subseteq & (Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R}))[Cand(\mathcal{E}, \mathcal{R})]); (Opn(\mathcal{E}, \mathcal{R})[Id(\mathscr{L}(\mathcal{E}))]) & (5.3.4(4)) \\ \subseteq & (Cand(\mathcal{E}, \mathcal{R}) \cup Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})[Cand(\mathcal{E}, \mathcal{R})])); Opn(\mathcal{E}, \mathcal{R}) & (5.3.6(5), 5.3.4(2)) \\ = & (Cand(\mathcal{E}, \mathcal{R}); Opn(\mathcal{E}, \mathcal{R})) \cup (Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})[Cand(\mathcal{E}, \mathcal{R})]); Opn(\mathcal{E}, \mathcal{R})) & \\ = & Cand(\mathcal{E}, \mathcal{R}) \cup Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})[Cand(\mathcal{E}, \mathcal{R})]); Opn(\mathcal{E}, \mathcal{R}) & (5.4.5(3)) \end{array}$$

$\square$

Proposition 5.4.6 is an amalgamation of the results due to Howe and Ong, and the extension to infinite erratic non-determinism. It allows pre-fixed-point properties of the closed restriction of the congruence candidate to be inferred from pre-fixed-point properties of the underlying relation. Fortunately, only two cases are necessary because all of the variants of similarity and bisimilarity are defined in terms of the simulation functions $\langle\cdot\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$ and $\langle\cdot\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$. Note that the induction in the second part of the proof makes use of the fact that the must convergence rules for programs of value type are not axioms.

Proposition 5.4.6 For a preorder $\mathcal{R} \in Rel_0(\mathcal{E})$:

1. If $\mathcal{R} \subseteq \langle\mathcal{R}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$ then $Cls(Cand(\mathcal{E},\mathcal{R})) \subseteq \langle Cls(Cand(\mathcal{E},\mathcal{R}))\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$.
2. If $\mathcal{R} \subseteq \langle\mathcal{R}\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$ then $Cls(Cand(\mathcal{E},\mathcal{R})) \subseteq \langle Cls(Cand(\mathcal{E},\mathcal{R}))\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$.

Proof

1. Assume $\mathcal{R} \subseteq \langle\mathcal{R}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$. We show that for all types σ and programs $M, M' \in \mathscr{L}_0(\mathcal{E})(\sigma)$ such that $\langle M, M'\rangle \in Cls(Cand(\mathcal{E},\mathcal{R}))$, we have:

   $$\langle M, M'\rangle \in \langle Cls(Cand(\mathcal{E},\mathcal{R}))\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$$

   By lemma 5.3.9(1), $\langle M, M'\rangle \in \langle Cls(Cand(\mathcal{E},\mathcal{R}))\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$ if and only if, for all K such that $M \Downarrow^{\mathrm{may}} K$, there exists K' such that $M' \Downarrow^{\mathrm{may}} K'$ and:

   $$\vdash \langle K, K'\rangle \in Cmp(\mathcal{E}, Opn(\mathcal{E}, Cls(Cand(\mathcal{E},\mathcal{R})))) : \sigma$$

   The latter condition can be simplified. By lemmas 5.3.3(3) and 5.4.5(1,5):

   $$Cand(\mathcal{E},\mathcal{R})[Id(\mathscr{L}(\mathcal{E}))] \subseteq Cand(\mathcal{E},\mathcal{R})[Cand(\mathcal{E},\mathcal{R})] \subseteq Cand(\mathcal{E},\mathcal{R})$$

   By forming the closed restrictions of both sides (a monotone operation) and applying lemma 5.3.4(5):

   $$Cand(\mathcal{E},\mathcal{R}) \subseteq Opn(\mathcal{E}, Cls(Cand(\mathcal{E},\mathcal{R})))$$

   Therefore, by monotonicity of compatible refinement, it suffices to show:

   $$\vdash \langle K, K'\rangle \in Cmp(\mathcal{E}, Cand(\mathcal{E},\mathcal{R})) : \sigma$$

   We prove this by induction on the derivation of $M \Downarrow^{\mathrm{may}} K$. In each case, $\langle M, M'\rangle \in Cls(Cand(\mathcal{E},\mathcal{R}))$, so there exists $M'' \in \mathscr{L}_0(\mathcal{E})(\sigma)$ such that:

   $$\vdash \langle M, M''\rangle \in Cmp(\mathcal{E}, Cand(\mathcal{E},\mathcal{R})) : \sigma$$

   and:

   $$\vdash \langle M'', M'\rangle \in Opn(\mathcal{E},\mathcal{R}) : \sigma$$

If we can find K'' such that $M'' \Downarrow^{\mathrm{may}} K''$ and:

$$\vdash \langle K, K'' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \sigma$$

then $\langle M'', M' \rangle \in \mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$, and so, by lemma 5.3.9(1), there exists K' such that $M' \Downarrow^{\mathrm{may}} K'$ and:

$$\vdash \langle K'', K' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{R})) : \sigma$$

But, by lemmas 5.3.6(4) and 5.4.5(3):

$$\begin{array}{ll} & \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})); \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{R})) \\ \subseteq & \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R}); \mathit{Opn}(\mathcal{E}, \mathcal{R})) \\ = & \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) \end{array}$$

so $\vdash \langle K, K' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \sigma$, as required. We illustrate finding K'' with the cases for sequential composition and fixed-point programs. Suppose $L \Downarrow^{\mathrm{may}} [M]$ and $N[M/x] \Downarrow^{\mathrm{may}} K$, so $\mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, N \Downarrow^{\mathrm{may}} K$, and that:

$$\vdash \langle \mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, N, \mathsf{let}\, x \Leftarrow L''\, \mathsf{in}\, N'' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \mathsf{P}_\perp(\tau)$$

Hence $\vdash \langle L, L'' \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)$ and $x : \sigma \vdash \langle N, N'' \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\tau)$. By applying the induction hypothesis to $L \Downarrow^{\mathrm{may}} [M]$, we deduce there exists M'' such that $L'' \Downarrow^{\mathrm{may}} [M'']$ and $\vdash \langle [M], [M''] \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \mathsf{P}_\perp(\sigma)$. Therefore $\vdash \langle M, M'' \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R}) : \sigma$. By lemma 5.4.5(5):

$$\vdash \langle N[M/x], N''[M''/x] \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\tau)$$

Applying the induction hypothesis to $N[M/x] \Downarrow^{\mathrm{may}} K$ yields K'' such that $N''[M''/x] \Downarrow^{\mathrm{may}} K''$ and $\vdash \langle K, K'' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \mathsf{P}_\perp(\tau)$. Therefore $\mathsf{let}\, x \Leftarrow L''\, \mathsf{in}\, N'' \Downarrow^{\mathrm{may}} K''$, and this completes the case for sequential composition. For fixed-point programs, suppose $\mathsf{fix}\, x. M \Downarrow^{\mathrm{may}} K$ and:

$$\vdash \langle \mathsf{fix}\, x. M, \mathsf{fix}\, x. M'' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \mathsf{P}_\perp(\sigma)$$

Hence $x : \mathsf{P}_\perp(\sigma) \vdash \langle M, M'' \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)$. In addition, the congruence candidate is compatible by lemma 5.4.5(4), so:

$$\vdash \langle \mathsf{fix}\, x. M, \mathsf{fix}\, x. M'' \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)$$

By lemma 5.4.5(5):

$$\vdash \langle M[\mathsf{fix}\, x. M/x], M''[\mathsf{fix}\, x. M''/x] \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)$$

By applying the induction hypothesis to $M[\mathsf{fix}\, x. M/x] \Downarrow^{\mathrm{may}} K$, we obtain K'' such that $M''[\mathsf{fix}\, x. M''/x] \Downarrow^{\mathrm{may}} K''$ and $\vdash \langle K, K'' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \mathsf{P}_\perp(\sigma)$. Thus $\mathsf{fix}\, x. M'' \Downarrow^{\mathrm{may}} K''$, and this completes the case for fixed-point programs.

2. Assume $\mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$. We show that for all types σ and programs $M, M' \in \mathscr{L}_0(\mathcal{E})(\sigma)$ such that $\langle M, M' \rangle \in \mathit{Cls}(\mathit{Cand}(\mathcal{E}, \mathcal{R}))$, we have:

$$\langle M, M' \rangle \in \langle \mathit{Cls}(\mathit{Cand}(\mathcal{E}, \mathcal{R})) \rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$$

By lemma 5.3.9(2), $\langle M, M' \rangle \in \langle \mathit{Cls}(\mathit{Cand}(\mathcal{E}, \mathcal{R})) \rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$ if and only if $M \Downarrow^{\mathrm{must}}$ implies $M' \Downarrow^{\mathrm{must}}$ and, for all K' such that $M' \Downarrow^{\mathrm{may}} K'$, there exists K such that $M \Downarrow^{\mathrm{may}} K$, and:

$$\vdash \langle K, K' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathit{Cls}(\mathit{Cand}(\mathcal{E}, \mathcal{R})))) : \sigma$$

By the same argument as in (1), it suffices to show:

$$\vdash \langle K, K' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \sigma$$

We prove this by well-founded induction on the derivation of $M \Downarrow^{\mathrm{must}}$. In each case, $\langle M, M' \rangle \in \mathit{Cls}(\mathit{Cand}(\mathcal{E}, \mathcal{R}))$, so there exists $M'' \in \mathscr{L}_0(\mathcal{E})(\sigma)$ such that:

$$\vdash \langle M, M'' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \sigma$$

and:

$$\vdash \langle M'', M' \rangle \in \mathit{Opn}(\mathcal{E}, \mathcal{R}) : \sigma$$

Suppose $M \Downarrow^{\mathrm{must}}$. If we can show that $M'' \Downarrow^{\mathrm{must}}$, then, by lemma 5.3.9(2) and $\langle M'', M' \rangle \in \mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$, $M' \Downarrow^{\mathrm{must}}$ and, for all K' such that $M' \Downarrow^{\mathrm{may}} K'$, there exists K'' such that $M'' \Downarrow^{\mathrm{may}} K''$ and:

$$\vdash \langle K'', K' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{R})) : \sigma$$

If we can also find a program K such that $M \Downarrow^{\mathrm{may}} K$ and:

$$\vdash \langle K, K'' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \sigma$$

then as in (1), $\vdash \langle K, K' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \sigma$, as required. Again, we give the cases for sequential composition and fixed-point programs. Suppose $\mathsf{let}\, x \Leftarrow L \,\mathsf{in}\, N \Downarrow^{\mathrm{must}}$ and:

$$\vdash \langle \mathsf{let}\, x \Leftarrow L \,\mathsf{in}\, N, \mathsf{let}\, x \Leftarrow L'' \,\mathsf{in}\, N'' \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \mathsf{P}_\perp(\tau)$$

Hence $\vdash \langle L, L'' \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)$ and $x : \sigma \vdash \langle N, N'' \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\tau)$. By applying the induction hypothesis to $L \Downarrow^{\mathrm{must}}$, we have that $L'' \Downarrow^{\mathrm{must}}$ and, for all M'' such that $L'' \Downarrow^{\mathrm{may}} [M'']$, there exists M such that $L \Downarrow^{\mathrm{may}} [M]$ and:

$$\vdash \langle [M], [M''] \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) : \mathsf{P}_\perp(\sigma)$$

Consider M'' such that $L'' \Downarrow^{\text{may}} [M'']$. We now want to show that $N''[M''/x] \Downarrow^{\text{must}}$. By the induction hypothesis as above, there exists M such that $L \Downarrow^{\text{may}} [M]$ and $\vdash \langle M, M'' \rangle \in Cand(\mathcal{E}, \mathcal{R}) : \sigma$. By lemma 5.4.5(5):

$$\vdash \langle N[M/x], N''[M''/x] \rangle \in Cand(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\tau)$$

By applying the induction hypothesis to $N[M/x] \Downarrow^{\text{must}}$ we have that $N''[M''/x] \Downarrow^{\text{must}}$ and, for all K'' such that $N''[M''/x] \Downarrow^{\text{may}} K''$, there exists K such that $N[M/x] \Downarrow^{\text{may}} K$ and $\vdash \langle K, K'' \rangle \in Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})) : \mathsf{P}_\perp(\tau)$. In conjunction with the first use of the induction hypothesis, we deduce that, for all K'' such that $\mathsf{let}\, x \Leftarrow L'' \,\mathsf{in}\, N'' \Downarrow^{\text{may}} K''$, there exists K such that $\mathsf{let}\, x \Leftarrow L \,\mathsf{in}\, N \Downarrow^{\text{may}} K$ and $\vdash \langle K, K'' \rangle \in Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})) : \mathsf{P}_\perp(\tau)$, as required. This completes the case for sequential composition. For fixed-point programs, suppose $\mathsf{fix}\, x. M \Downarrow^{\text{must}}$ and:

$$\vdash \langle \mathsf{fix}\, x. M, \mathsf{fix}\, x. M'' \rangle \in Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})) : \mathsf{P}_\perp(\sigma)$$

Then as in (1):

$$\vdash \langle M[\mathsf{fix}\, x. M/x], M''[\mathsf{fix}\, x. M''/x] \rangle \in Cand(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)$$

Applying the induction hypothesis to $M[\mathsf{fix}\, x. M/x] \Downarrow^{\text{must}}$ we find that $M''[\mathsf{fix}\, x. M''/x] \Downarrow^{\text{must}}$ and, for all K'' such that $M''[\mathsf{fix}\, x. M''/x] \Downarrow^{\text{may}} K''$, there exists a program K such that $M[\mathsf{fix}\, x. M/x] \Downarrow^{\text{may}} K$ and $\vdash \langle K, K'' \rangle \in Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})) : \mathsf{P}_\perp(\sigma)$. Therefore, for all K'' such that $\mathsf{fix}\, x. M'' \Downarrow^{\text{may}} K''$, there exists K such that $\mathsf{fix}\, x. M \Downarrow^{\text{may}} K$ and $\vdash \langle K, K'' \rangle \in Cmp(\mathcal{E}, Cand(\mathcal{E}, \mathcal{R})) : \mathsf{P}_\perp(\sigma)$. This completes the case for fixed-point programs. □

The following lemma is used in the three compatibility theorems to deduce that the open extension $\mathcal{R}$ of a variant of similarity or bisimilarity coincides with a relation $\mathcal{S}$ that is either the congruence candidate or its transitive closure. It follows that the open extension of $\mathcal{R}$ is compatible because the congruence candidate is.

Lemma 5.4.7 Consider relations $\mathcal{R} \in Rel_0(\mathcal{E})$ and $\mathcal{S} \in Rel(\mathcal{E})$ such that:

1. $Opn(\mathcal{E}, \mathcal{R}) \subseteq \mathcal{S}$
2. $Cls(\mathcal{S}) \subseteq \mathcal{R}$
3. $\mathcal{S}[Id(\mathscr{L}(\mathcal{E}))] \subseteq S$

Then $Opn(\mathcal{E}, \mathcal{R}) = S$.

Proof By (1), it suffices to show $S \subseteq Opn(\mathcal{E}, \mathcal{R})$. By monotonicity of closed restriction and (2), (3) we have $Cls(\mathcal{S}[Id(\mathscr{L}(\mathcal{E}))]) \subseteq Cls(\mathcal{S}) \subseteq R$. Then $S \subseteq Opn(\mathcal{E}, \mathcal{R})$ by lemma 5.3.4(5). □

With the results thus far we can prove compatibility of the open extensions of the lower, upper, and convex variants of similarity.

Theorem 5.4.8

1. The open extensions $Opn(\mathcal{E}, \lesssim_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})})$, $Opn(\mathcal{E}, \lesssim_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})})$, $Opn(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})$ of the lower, upper, and convex variants of similarity are compatible.
2. The open extensions $Opn(\mathcal{E}, \simeq_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})})$, $Opn(\mathcal{E}, \simeq_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})})$, $Opn(\mathcal{E}, \simeq_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})$ of the lower, upper, and convex variants of mutual similarity are compatible.

Proof

1. If the hypotheses of lemma 5.4.7 can be shown to hold when $\mathcal{R}$ is one of the variants of similarity $\lesssim_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$, $\lesssim_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$, $\lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}$ and $S = Cand(\mathcal{E}, \mathcal{R})$, then $Opn(\mathcal{E}, \mathcal{R}) = Cand(\mathcal{E}, \mathcal{R})$. Consequently, $Opn(\mathcal{E}, \mathcal{R})$ is compatible by lemma 5.4.5(4). We establish the hypotheses of lemma 5.4.7 for convex similarity, i.e., $\mathcal{R} = \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}$. Hypothesis (1), $Opn(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}) \subseteq Cand(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})$, is an instance of lemma 5.4.5(2). Hypothesis (3), $Cand(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})[Id(\mathscr{L}(\mathcal{E}))] \subseteq Cand(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})$, follows from lemmas 5.3.3(3) and 5.4.5(1,5). For hypothesis (2), $Cls(Cand(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})) \subseteq \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}$, recall that:

   $$\lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})} \stackrel{\mathrm{def}}{=} \nu\mathcal{T}\,.\,\langle\mathcal{T}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})} \cap \langle\mathcal{T}\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$$

   By coinduction, hypothesis (2) holds if:

   $$\begin{array}{ll} & Cls(Cand(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})) \\ \subseteq & \langle Cls(Cand(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}))\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})} \cap \langle Cls(Cand(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}))\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})} \end{array}$$

   But this follows from proposition 5.4.6 because:

   $$\lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})} = \langle\lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})} \cap \langle\lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$$

   Therefore, lemma 5.4.7 applies and $Opn(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}) = Cand(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})$, so the open extension of convex similarity is compatible. The arguments for the lower and upper variants of similarity are similar.
2. We prove compatibility of mutual convex similarity. The other cases are similar. First note that, for all $R, S \in Rel_0(\mathcal{E})$, $Opn(\mathcal{E}, \mathcal{R} \cap \mathcal{S}) = Opn(\mathcal{E}, \mathcal{R}) \cap Opn(\mathcal{E}, \mathcal{S})$. Using (1) and the monotonicity of open extension and compatible refinement, we deduce:

   $$\begin{array}{ll} & Cmp(\mathcal{E}, Opn(\mathcal{E}, \simeq_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})) \\ \subseteq & Cmp(\mathcal{E}, Opn(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})) \cap Cmp(\mathcal{E}, Opn(\mathcal{E}, (\lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}})) \\ = & Cmp(\mathcal{E}, Opn(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})) \cap (Cmp(\mathcal{E}, Opn(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})))^{\mathrm{op}} \\ \subseteq & Opn(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}) \cap (Opn(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}))^{\mathrm{op}} \\ = & Opn(\mathcal{E}, \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}) \cap Opn(\mathcal{E}, (\lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}}) \\ = & Opn(\mathcal{E}, \simeq_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}) \end{array}$$

Therefore the open extension $Opn(\mathcal{E}, \simeq_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})})$ of convex similarity is compatible.

$\square$

Lower, Upper, and Convex Bisimilarity

The congruence candidate cannot be used directly to establish that refinement similarity and the variants of bisimilarity are compatible. The problem lies in the (essential) asymmetry of the definition of the congruence candidate. The placement of compatible refinement on the LHS of the sequential composition permits arguments by induction on the size of the left-hand term (rephrased here as a coinduction). However, for refinement similarity and the variants of bisimilarity, the argument has to proceed from LHS to RHS and vice-versa. The latter requires crossing the open extension of the refinement similarity or variant of bisimilarity. There is no guarantee that this produces a strictly smaller term, so an inductive argument fails.

Howe [How96] resolved this problem for bisimilarity by using the transitive closure of the congruence candidate instead of the congruence candidate itself. Pre-fixed-point properties of the congruence candidate can be lifted to the transitive closure with lemma 5.4.9.

Lemma 5.4.9 Consider a TTS $\mathscr{T}$ and a relation $\mathcal{R} \in \mathscr{T} \times \mathscr{T}$:

1. If $\mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\mathrm{LS}}^{\mathscr{T}}$ then $\mathcal{R}^+ \subseteq \langle \mathcal{R}^+ \rangle_{\mathrm{LS}}^{\mathscr{T}}$.
2. If $\mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\mathrm{US}}^{\mathscr{T}}$ then $\mathcal{R}^+ \subseteq \langle \mathcal{R}^+ \rangle_{\mathrm{US}}^{\mathscr{T}}$.

Proof We prove (2), the argument for (1) is similar. First note that the relation $\langle \mathcal{R}^+ \rangle_{\mathrm{US}}^{\mathscr{T}}$ is transitive because, by lemma 4.2.2 and the monotonicity of $\langle \cdot \rangle_{\mathrm{US}}^{\mathscr{T}}$:

$$\langle \mathcal{R}^+ \rangle_{\mathrm{US}}^{\mathscr{T}} ; \langle \mathcal{R}^+ \rangle_{\mathrm{US}}^{\mathscr{T}} \subseteq \langle \mathcal{R}^+ ; \mathcal{R}^+ \rangle_{\mathrm{US}}^{\mathscr{T}} \subseteq \langle \mathcal{R}^+ \rangle_{\mathrm{US}}^{\mathscr{T}}$$

Hence $\mathcal{R}^+ \subseteq \langle \mathcal{R}^+ \rangle_{\mathrm{US}}^{\mathscr{T}}$ holds if $\mathcal{R} \subseteq \langle \mathcal{R}^+ \rangle_{\mathrm{US}}^{\mathscr{T}}$. The latter follows from the hypothesis $\mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\mathrm{US}}^{\mathscr{T}}$ and the monotonicity of $\langle \cdot \rangle_{\mathrm{US}}^{\mathscr{T}}$, because $\mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\mathrm{US}}^{\mathscr{T}} \subseteq \langle \mathcal{R}^+ \rangle_{\mathrm{US}}^{\mathscr{T}}$. $\square$

The transitive closure of the congruence candidate is compatible when term constructors are finitary (but see the discussion below). With compatibility, the transitive closure of the congruence candidate of a variant of bisimilarity is symmetric, and so we can deduce the RHS to LHS property from the LHS to RHS property, which is in turn established by an inductive argument similar to that used for the variants of similarity.

Lemma 5.4.10 Consider an equivalence relation $\mathcal{R} \in Rel_0(\mathcal{E})$. If $Cand(\mathcal{E}, \mathcal{R})^+$ is compatible, then it is symmetric and hence an equivalence relation.

Proof We want to show $Cand(\mathcal{E},\mathcal{R})^+ \subseteq (Cand(\mathcal{E},\mathcal{R})^+)^{\mathrm{op}}$. The relation on the right-hand side is transitive, so it suffices to show that $Cand(\mathcal{E},\mathcal{R}) \subseteq (Cand(\mathcal{E},\mathcal{R})^+)^{\mathrm{op}}$. This follows by induction from:

$$Cmp(\mathcal{E},(Cand(\mathcal{E},\mathcal{R})^+)^{\mathrm{op}}); Opn(\mathcal{E},\mathcal{R}) \subseteq (Cand(\mathcal{E},\mathcal{R})^+)^{\mathrm{op}}$$

By assumption, $Cand(\mathcal{E},\mathcal{R})^+$ is compatible, so:

$$\begin{array}{rl} & Cmp(\mathcal{E},(Cand(\mathcal{E},\mathcal{R})^+)^{\mathrm{op}}) \\ = & Cmp(\mathcal{E},Cand(\mathcal{E},\mathcal{R})^+)^{\mathrm{op}} \\ \subseteq & (Cand(\mathcal{E},\mathcal{R})^+)^{\mathrm{op}} \end{array}$$

In addition, because $\mathcal{R}$ is an equivalence relation:

$$Opn(\mathcal{E},\mathcal{R}) = Opn(\mathcal{E},\mathcal{R})^{\mathrm{op}} \subseteq Cand(\mathcal{E},\mathcal{R})^{\mathrm{op}}$$

Hence:

$$\begin{array}{rl} & Cmp(\mathcal{E},(Cand(\mathcal{E},\mathcal{R})^+)^{\mathrm{op}}); Opn(\mathcal{E},\mathcal{R}) \\ \subseteq & (Cand(\mathcal{E},\mathcal{R})^+)^{\mathrm{op}}; Cand(\mathcal{E},\mathcal{R})^{\mathrm{op}} \\ = & (Cand(\mathcal{E},\mathcal{R}); Cand(\mathcal{E},\mathcal{R})^+)^{\mathrm{op}} \\ \subseteq & (Cand(\mathcal{E},\mathcal{R})^+)^{\mathrm{op}} \end{array}$$

Therefore $Cand(\mathcal{E},\mathcal{R})^+$ is symmetric. □

Unfortunately, it is non-trivial to show that the transitive closures of the congruence candidates of variants of bisimilarity are compatible in fragments of $\mathscr{L}$ for two reasons. The first is a consequence of example 5.3.7. For example, although terms may be related by:

$$Cmp(\mathcal{E}, Cand(\mathcal{E},\mathcal{R}); Cand(\mathcal{E},\mathcal{R}))$$

it does not follow that they are related by:

$$Cmp(\mathcal{E}, Cand(\mathcal{E},\mathcal{R})); Cmp(\mathcal{E}, Cand(\mathcal{E},\mathcal{R}))$$

This is because the intermediate term in the latter relation may not be in the language fragment $\mathscr{L}(\mathcal{E})$. Of course, the programming language $\mathscr{L}$ does not suffer from this problem because it is closed under every term constructor.

The second reason is that the usual proof of compatibility relies upon the following property of the compatible refinement of a reflexive relation $S \in Rel(\mathcal{E})$ (note that the left-hand term is equal to $Cmp(\mathcal{E},\mathcal{S}^+)$):

$$Cmp(\mathcal{E}, \bigcup\{\mathcal{S}^m \mid m \geq 1\}) = \bigcup\{Cmp(\mathcal{E},\mathcal{S}^m) \mid m \geq 1\}$$

The equality holds when all terms have a finite collection of immediate subterms, but may fail for term constructors with infinitely many immediate subterms. For example, define $\mathcal{S} \in Rel(\mathcal{E})$ by (although $\mathcal{S}$ is not reflexive, it does illustrate the problem):

$$\mathcal{S} = \{\langle \Gamma, \underline{n}, \underline{n+1} \rangle \mid \Gamma \text{ an environment} \wedge n \in \omega\}$$

Then, for all $n \in \omega$, $y_n : \mathsf{unit} \vdash \langle \underline{0}, \underline{n+1} \rangle \in \mathcal{S}^{n+1} : \mathsf{nat}$. Hence:

$$x : \mathsf{nat} \vdash \langle \mathsf{case}\, x\, \mathsf{of}\, \langle y_n.\underline{0} \mid n < \kappa \rangle, \mathsf{case}\, x\, \mathsf{of}\, \langle y_n.\underline{n+1} \mid n < \kappa \rangle \rangle \in Cmp(\mathcal{E}, \mathcal{S}^{+}) : \mathsf{nat}$$

But there is no natural number $m \in \omega$ such that:

$$x : \mathsf{nat} \vdash \langle \mathsf{case}\, x\, \mathsf{of}\, \langle y_n.\underline{0} \mid n < \kappa \rangle, \mathsf{case}\, x\, \mathsf{of}\, \langle y_n.\underline{n+1} \mid n < \kappa \rangle \rangle \in Cmp(\mathcal{E}, \mathcal{S}^{m}) : \mathsf{nat}$$

Therefore the terms are not related by $\bigcup\{Cmp(\mathcal{E}, \mathcal{S}^m) \mid m \geq 1\}$. The programming language $\mathscr{L}$ and all of the fragments suffer from this problem.

The partial solution proposed here is to use the existing compatibility results for variants of mutual similarity to force through the cases for problematic term constructors (erratic choice and infinitary coproducts and products). If the types of the immediate subterms of problematic term constructors have P-orders less than or equal to 1, then they are related by a variant of bisimilarity if and only if they are related by the corresponding variant of mutual similarity, and so the existing compatibility result can be applied. Thus the argument places restrictions upon the types of immediate subterms of problematic term constructors in a language fragment.

Definition 5.4.11 A fragment $\mathscr{L}(\mathcal{E})$ is said to be bounded by an ordinal A if the following hold:

1. If $\Gamma \vdash \mathsf{inj}\, m\, \mathsf{of}\, M \in \mathscr{L}(\mathcal{E}) : \mathsf{sum}\, \langle \sigma_n \mid n < \omega \rangle$, then $POrd(\sigma_m) \leq A$.
2. If $\Gamma \vdash \mathsf{case}\, M\, \mathsf{of}\, \langle x_n.N_n \mid n < \omega \rangle \in \mathscr{L}(\mathcal{E}) : \tau$ and $\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{sum}\, \langle \sigma_n \mid n < \omega \rangle$, then $POrd(\mathsf{sum}\, \langle \sigma_n \mid n < \omega \rangle) \leq A$ and $POrd(\tau) \leq A$.
3. If $\Gamma \vdash \mathsf{tuple}\, \langle M_n \mid n < \omega \rangle \in \mathscr{L}(\mathcal{E}) : \mathsf{prod}\, \langle \sigma_n \mid n < \omega \rangle$, then, for all $n \in \omega$, $POrd(\sigma_n) \leq A$.
4. If $\Gamma \vdash \mathsf{proj}\, m\, \mathsf{of}\, M \in \mathscr{L}(\mathcal{E}) : \sigma_m$ and $\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{prod}\, \langle \sigma_n \mid n < \omega \rangle$, then $POrd(\mathsf{prod}\, \langle \sigma_n \mid n < \omega \rangle) \leq A$.
5. If $\Gamma \vdash ?\langle M_n \mid n < \kappa \rangle \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_{\perp}(\sigma)$, then $POrd(\sigma) \leq A$.

Under these conditions and with $A = 1$, it can be shown that the transitive closures of the congruence candidates of the variants of bisimilarity are compatible, and hence symmetric by lemma 5.4.10.

Proposition 5.4.12 Let $\mathcal{R} \in Rel_0(\mathcal{E})$ be one of the variants of bisimilarity $\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})}$, $\simeq_{\mathrm{UB}}^{\mathscr{L}_0(\mathcal{E})}$, $\simeq_{\mathrm{CB}}^{\mathscr{L}_0(\mathcal{E})}$. If the language fragment $\mathscr{L}(\mathcal{E})$ is bounded by 1, then $Cand(\mathcal{E}, \mathcal{R})^{+}$ is compatible.

Proof By case analysis of the compatible refinement rule schema. We start with the rule schema for finite term constructors, with the exception of erratic choice. These cases do not depend upon the bound for the language fragment, because they have a finite collection of terms in the premises and every fragment is closed under these term constructors. We illustrate with the case for the sequential composition constructor. Suppose:

$$\Gamma \vdash \langle \mathsf{let}\, x \Leftarrow M_1 \,\mathsf{in}\, M_2, \mathsf{let}\, x \Leftarrow N_1 \,\mathsf{in}\, N_2 \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})^+) : \mathsf{P}_\perp(\tau)$$

So $\Gamma \vdash \langle M_1, N_1 \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R})^+ : \mathsf{P}_\perp(\sigma)$ and $\Gamma, x : \sigma \vdash \langle M_2, N_2 \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R})^+ : \mathsf{P}_\perp(\tau)$. There must exist natural numbers $i, j \geq 1$ as well as terms $M_1^0, M_1^1, \ldots, M_1^i$ and $M_2^0, M_2^1, \ldots, M_2^j$ such that $M_1 = M_1^0$, $M_1^i = N_1$, $M_2 = M_2^0$, $M_2^j = N_2$, and:

1. For all $0 \leq k \leq i$, $\Gamma \vdash M_1^k \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\sigma)$.

2. For all $0 \leq k \leq j$, $\Gamma, x : \sigma \vdash M_2^k \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\tau)$.

3. For all $0 \leq k < i$, $\Gamma \vdash \langle M_1^k, M_1^{k+1} \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)$.

4. For all $0 \leq k < j$, $\Gamma, x : \sigma \vdash \langle M_2^k, M_2^{k+1} \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\tau)$.

The closure conditions upon fragments ensure that, for all k_1, k_2 such that $0 \leq k_1 \leq i$ and $0 \leq k_2 \leq j$, we have $\Gamma \vdash \mathsf{let}\, x \Leftarrow M_1^{k_1} \,\mathsf{in}\, M_2^{k_2} \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\tau)$. Now consider the list of terms:

$$\begin{array}{l}
\mathsf{let}\, x \Leftarrow M_1^1 \,\mathsf{in}\, M_2^1 \\
\mathsf{let}\, x \Leftarrow M_1^2 \,\mathsf{in}\, M_2^1 \\
\qquad \vdots \\
\mathsf{let}\, x \Leftarrow M_1^i \,\mathsf{in}\, M_2^1 \\
\mathsf{let}\, x \Leftarrow M_1^i \,\mathsf{in}\, M_2^2 \\
\qquad \vdots \\
\mathsf{let}\, x \Leftarrow M_1^i \,\mathsf{in}\, M_2^j
\end{array}$$

Each consecutive pair of terms is related by $\mathit{Cmp}(\mathcal{E}, \mathit{Cand}(\mathcal{E}, \mathcal{R})) \subseteq \mathit{Cand}(\mathcal{E}, \mathcal{R})$ because one pair of immediate subterms are related by $\mathit{Cand}(\mathcal{E}, \mathcal{R})$ and the other pair by $\mathit{Id}(\mathscr{L}(\mathcal{E})) \subseteq \mathit{Cand}(\mathcal{E}, \mathcal{R})$. Therefore:

$$\Gamma \vdash \langle \mathsf{let}\, x \Leftarrow M_1^1 \,\mathsf{in}\, M_2^1, \mathsf{let}\, x \Leftarrow M_1^i \,\mathsf{in}\, M_2^j \rangle \in \mathit{Cand}(\mathcal{E}, \mathcal{R})^+ : \mathsf{P}_\perp(\tau)$$

This completes the case for sequential composition.

We now consider the remaining coproduct and product cases (when $\kappa = \omega$) and the erratic choice case. Let $\mathcal{S} \in \mathit{Rel}_0(\mathcal{E})$ be the variant of mutual similarity corresponding to the variant of bisimilarity $\mathcal{R}$, e.g., if $\mathcal{R} = \simeq_{\mathrm{CB}}^{\mathscr{L}_0(\mathcal{E})}$ then $\mathcal{S} = \simeq_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}$. By theorem 5.4.8(2), the open extension $\mathit{Opn}(\mathcal{E}, \mathcal{S})$ of $\mathcal{S}$ is compatible. Using the compatibility and transitivity of $\mathit{Opn}(\mathcal{E}, \mathcal{S})$, it can be shown by induction that $Cand(\mathcal{E},\mathcal{S}) = Opn(\mathcal{E},\mathcal{S})$ and so $Cand(\mathcal{E},\mathcal{S})^{+} = Opn(\mathcal{E},\mathcal{S})$. In addition, a simple induction shows that $Cand(\mathcal{E},\mathcal{R}) \subseteq Cand(\mathcal{E},\mathcal{S})$ because $Cand(\mathcal{E},\mathcal{S})$ is compatible and $Opn(\mathcal{E},\mathcal{R}) \subseteq Opn(\mathcal{E},\mathcal{S})$. Now consider a type σ such that $POrd(\sigma) \leq 1$, and programs M, N such that $\Gamma \vdash M \in \mathscr{L}(\mathcal{E}) : \sigma$ and $\Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \sigma$. Using lemma 4.2.5(1,2) it can be shown that $\Gamma \vdash \langle M,N\rangle \in Opn(\mathcal{E},\mathcal{S}) : \sigma$ if and only if $\Gamma \vdash \langle M,N\rangle \in Opn(\mathcal{E},\mathcal{R}) : \sigma$. Therefore:

$$
\begin{array}{rl}
 & \Gamma \vdash \langle M,N\rangle \in Cand(\mathcal{E},\mathcal{R})^{+} : \sigma \\
\Longrightarrow & \Gamma \vdash \langle M,N\rangle \in Cand(\mathcal{E},\mathcal{S})^{+} : \sigma \\
\Longleftrightarrow & \Gamma \vdash \langle M,N\rangle \in Opn(\mathcal{E},\mathcal{S}) : \sigma \\
\Longleftrightarrow & \Gamma \vdash \langle M,N\rangle \in Opn(\mathcal{E},\mathcal{R}) : \sigma \\
\Longrightarrow & \Gamma \vdash \langle M,N\rangle \in Cand(\mathcal{E},\mathcal{R}) : \sigma
\end{array}
$$

Now when terms are related by $Cmp(\mathcal{E},Cand(\mathcal{E},\mathcal{R})^{+})$, their immediate subterms are related by $Cand(\mathcal{E},\mathcal{R})^{+}$. If the types of the immediate subterms have P-orders less than or equal to 1, then the argument above holds, and so the immediate subterms are related by $Cand(\mathcal{E},\mathcal{R})$. This implies that the original terms are related by:

$$Cmp(\mathcal{E},Cand(\mathcal{E},\mathcal{R})) \subseteq Cand(\mathcal{E},\mathcal{R}) \subseteq Cand(\mathcal{E},\mathcal{R})^{+}$$

By assumption, the language fragment $\mathscr{L}(\mathcal{E})$ is bounded by 1, and this ensures that the immediate subterms of the remaining cases always have types with P-order less than or equal to 1. For example, consider:

$$\Gamma \vdash \langle ?\langle M_n \mid n < \kappa\rangle, ?\langle N_n \mid n < \kappa\rangle\rangle \in Cmp(\mathcal{E},Cand(\mathcal{E},\mathcal{R})^{+}) : \mathsf{P}_{\perp}(\sigma)$$

We know that $POrd(\sigma) \leq 1$ and, for all $n < \kappa$, $\Gamma \vdash \langle M_n,N_n\rangle \in Cand(\mathcal{E},\mathcal{R})^{+} : \sigma$. This implies that, for all $n < \kappa$, $\Gamma \vdash \langle M_n,N_n\rangle \in Cand(\mathcal{E},\mathcal{R}) : \sigma$. Hence:

$$\Gamma \vdash \langle ?\langle M_n \mid n < \kappa\rangle, ?\langle N_n \mid n < \kappa\rangle\rangle \in Cmp(\mathcal{E},Cand(\mathcal{E},\mathcal{R})) : \mathsf{P}_{\perp}(\sigma)$$

Finally we deduce:

$$\Gamma \vdash \langle ?\langle M_n \mid n < \kappa\rangle, ?\langle N_n \mid n < \kappa\rangle\rangle \in Cand(\mathcal{E},\mathcal{R})^{+} : \mathsf{P}_{\perp}(\sigma)$$

The arguments for coproduct and product cases when $\kappa = \omega$ are similar. $\square$

It can be verified that $\mathscr{L}(\emptyset)$ is bounded by 0, because there are no erratic choice terms and the arithmetic operators (and their subterms) decompose and produce terms of type nat only. Similarly, language fragments such as $\mathscr{L}(?\langle \mathsf{false},\mathsf{true}\rangle)$ and $\mathscr{L}(?\underline{\omega})$ are also bounded by 0. In addition, the language fragment $\mathscr{L}(?\langle\Omega,[\mathsf{false}],[\mathsf{true}]\rangle)$ is bounded by 1 (this fragment is used in section 5.6). More generally, we can define a substantial language fragment $\mathscr{M}$ that is bounded by 1, includes the most common forms of erratic non-determinism, includes analogues of the examples given in sections 4.3 and 4.4, and suffices for the results in sections 5.5 and 5.6. However, there are no ordinals strictly less than ω_1 that bound the programming language $\mathscr{L}$.

Definition 5.4.13 Define the language fragment $\mathscr{M}$ to be the smallest language fragment that is closed under the following rule schema when the premises have types with P-orders less than or equal to 1:

$$\frac{\Gamma \vdash M \in \mathscr{M} : \sigma_m}{\Gamma \vdash \mathsf{inj}\, m\, \mathsf{of}\, M \in \mathscr{M} : \mathsf{sum}\, \langle \sigma_n \mid n < \omega \rangle}$$

$$\frac{\Gamma \vdash M \in \mathscr{M} : \mathsf{sum}\, \langle \sigma_n \mid n < \omega \rangle \qquad \{\Gamma, x_n : \sigma_n \vdash N_n \in \mathscr{M} : \tau \mid n < \omega\}}{\Gamma \vdash \mathsf{case}\, M\, \mathsf{of}\, \langle x_n.N_n \mid n < \omega \rangle \in \mathscr{M} : \tau}$$

$$\frac{\{\Gamma \vdash M_n \in \mathscr{M} : \sigma_n \mid n < \omega\}}{\Gamma \vdash \mathsf{tuple}\, \langle M_n \mid n < \omega \rangle \in \mathscr{M} : \mathsf{prod}\, \langle \sigma_n \mid n < \omega \rangle}$$

$$\frac{\Gamma \vdash M \in \mathscr{M} : \mathsf{prod}\, \langle \sigma_n \mid n < \omega \rangle}{\Gamma \vdash \mathsf{proj}\, m\, \mathsf{of}\, M \in \mathscr{M} : \sigma_m}$$

$$\frac{\{\Gamma \vdash M_n \in \mathscr{M} : \sigma \mid n < \kappa\}}{\Gamma \vdash ?\langle M_n \mid n < \kappa \rangle \in \mathscr{M} : \mathsf{P}_\perp(\sigma)}$$

The set of programs in $\mathscr{M}$ is denoted $\mathscr{M}_0$.

Now that compatibility and symmetry of the transitive closure of the congruence candidate have been established for some language fragments, we are in a position to use Howe's technique to prove compatibility of the variants of bisimilarity.

Note that if we know that the transitive closure of the congruence candidate is compatible, then we have that it is symmetric by lemma 5.4.10. We use the weaker condition in the statement of the result.

Theorem 5.4.14 Let $\mathcal{R} \in Rel_0(\mathcal{E})$ be the lower $\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})}$, upper $\simeq_{\text{UB}}^{\mathscr{L}_0(\mathcal{E})}$, or convex $\simeq_{\text{CB}}^{\mathscr{L}_0(\mathcal{E})}$ variant of bisimilarity. If $Cand(\mathcal{E},\mathcal{R})^+$ is symmetric, then $Opn(\mathcal{E},\mathcal{R})$ is compatible.

Proof We prove the case for lower bisimilarity. Assume that $Cand(\mathcal{E},\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})})^+$ is symmetric. As with theorem 5.4.8, we want to apply lemma 5.4.7, but in this case $\mathcal{R} = \simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})}$ and $\mathcal{S} = Cand(\mathcal{E},\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})})^+$. Hypothesis (1) is $Opn(\mathcal{E},\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})}) \subseteq Cand(\mathcal{E},\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})})^+$ which follows from lemma 5.4.5(2). Hypothesis (2) is $Cls(Cand(\mathcal{E},\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})})^+) \subseteq \simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})}$. Note that $Cls(Cand(\mathcal{E},\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})})^+) = Cls(Cand(\mathcal{E},\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})}))^+$ and both are symmetric. By coinduction, it suffices to show that:

$$\begin{aligned} Cls(Cand(\mathcal{E},\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})}))^+ \subseteq{} & \langle Cls(Cand(\mathcal{E},\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})}))^+\rangle_{\text{LS}}^{\mathscr{L}_0(\mathcal{E})} \cap \\ & (\langle (Cls(Cand(\mathcal{E},\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})}))^+)^{\text{op}}\rangle_{\text{LS}}^{\mathscr{L}_0(\mathcal{E})})^{\text{op}} \end{aligned}$$

Consider the first part of the inclusion:

$$Cls(Cand(\mathcal{E},\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})}))^+ \subseteq \langle Cls(Cand(\mathcal{E},\simeq_{\text{LB}}^{\mathscr{L}_0(\mathcal{E})}))^+\rangle_{\text{LS}}^{\mathscr{L}_0(\mathcal{E})}$$

By lemma 5.4.9(1), this holds if:

$$Cls(Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})) \subseteq \langle Cls(Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})}))\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$$

And, returning to proposition 5.4.6(1), this holds if:

$$\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})} \subseteq \langle \simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$$

This follows immediately from the definition of lower bisimilarity. For the second part of the inclusion, we want to show that:

$$Cls(Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})}))^+ \subseteq (\langle (Cls(Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})}))^+)^{\mathrm{op}}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}}$$

Or equivalently:

$$(Cls(Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})}))^+)^{\mathrm{op}} \subseteq \langle (Cls(Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})}))^+)^{\mathrm{op}}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$$

But $Cls(Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})}))^+$ is symmetric, so this is the same as the first inclusion. Hence, hypothesis (2) of lemma 5.4.7 holds. Hypothesis (3) is:

$$Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})^+[Id(\mathscr{L}(\mathcal{E}))] \subseteq Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})^+$$

Without loss of generality, consider environments $\Gamma,\Delta = x_1 : \sigma_1,\ldots,x_n : \sigma_n$ such that:

$$\Gamma,\Delta \vdash \langle M,N\rangle \in Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})^+ : \tau$$

and, for all i such that $1 \le i \le n$:

$$\Gamma \vdash L_i \in \mathscr{L}(\mathcal{E}) : \sigma_i$$

So, with $\vec{L} = L_1,\ldots,L_n$:

$$\Gamma \vdash \langle M[\vec{L}/\vec{x}],N[\vec{L}/\vec{x}]\rangle \in Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})^+[Id(\mathscr{L}(\mathcal{E}))] : \tau$$

There exists $k \ge 1$ such that:

$$\Gamma,\Delta \vdash \langle M,N\rangle \in Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})^k : \tau$$

and, for all i such that $1 \le i \le n$:

$$\Gamma \vdash \langle L_i,L_i\rangle \in Id(\mathscr{L}(\mathcal{E}))^k : \sigma_i$$

By induction and lemma 5.3.4(4) we have:

$$Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})^k[Id(\mathscr{L}(\mathcal{E}))^k] \subseteq (Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})[Id(\mathscr{L}(\mathcal{E}))])^k$$

Another induction and lemmas 5.3.3(3) and 5.4.5(1,5) shows that:

$$(Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})[Id(\mathscr{L}(\mathcal{E}))])^k \subseteq Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})^k \subseteq Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})^+$$

Therefore:

$$\Gamma \vdash \langle M[\vec{L}/\vec{x}],N[\vec{L}/\vec{x}]\rangle \in Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})^+ : \tau$$

and hypothesis (3) of lemma 5.4.7 holds.

By applying lemma 5.4.7 we find that $Opn(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})}) = Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})^+$. But then:

$$Opn(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})}) \subseteq Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})}) \subseteq Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})^+ = Opn(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})$$

Therefore $Opn(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})}) = Cand(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})$, and so $Opn(\mathcal{E},\simeq_{\mathrm{LB}}^{\mathscr{L}_0(\mathcal{E})})$ is compatible. $\square$

Refinement Similarity

Refinement similarity requires a different technique because the argument cannot be completed by symmetry as it was for the variants of bisimilarity (refinement similarity is not symmetric). Again, we seek a relation $\mathcal{S} \in Rel(\mathcal{E})$ for which we can prove $Cls(\mathcal{S}) \subseteq \lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}$ by coinduction:

$$Cls(\mathcal{S}) \subseteq (\langle Cls(\mathcal{S})^{\mathrm{op}}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}} \cap \langle Cls(\mathcal{S})\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$$

The first part of the inclusion is problematic when $Cls(\mathcal{S})$ is not symmetric. The solution used in theorem 5.4.18 is to set $\mathcal{S}$ to be the transitive closure of the congruence candidate of refinement similarity and then move the dual operation inside the congruence candidate, which allows the argument to proceed as for bisimilarity. However, attempting this move reverses the positions of the open extension and compatible refinement operators inside the least fixed-point as shown in lemma 5.4.15.

Lemma 5.4.15 If $\mathcal{R} \in Rel_0(\mathcal{E})$ then:

$$Cand(\mathcal{E}, \mathcal{R}^{\mathrm{op}}) = (\mu\mathcal{T} \,.\, Opn(\mathcal{E}, \mathcal{R}); Cmp(\mathcal{E}, \mathcal{T}))^{\mathrm{op}}$$

Proof By definition:

$$Cand(\mathcal{E}, \mathcal{R}^{\mathrm{op}}) = \mu\mathcal{T} \,.\, Cmp(\mathcal{E}, \mathcal{T}); Opn(\mathcal{E}, \mathcal{R}^{\mathrm{op}})$$

Taking the dual commutes with open extension and compatible refinement, so by lemma 2.3.7:

$$\begin{aligned}
& \mu\mathcal{T} \,.\, Cmp(\mathcal{E}, \mathcal{T}); Opn(\mathcal{E}, \mathcal{R}^{\mathrm{op}}) \\
= \ & \mu\mathcal{T} \,.\, (Cmp(\mathcal{E}, \mathcal{T})^{\mathrm{op}})^{\mathrm{op}}; Opn(\mathcal{E}, \mathcal{R})^{\mathrm{op}} \\
= \ & \mu\mathcal{T} \,.\, (Opn(\mathcal{E}, \mathcal{R}); Cmp(\mathcal{E}, \mathcal{T}^{\mathrm{op}}))^{\mathrm{op}} \\
= \ & (\mu\mathcal{T} \,.\, Opn(\mathcal{E}, \mathcal{R}); Cmp(\mathcal{E}, \mathcal{T}))^{\mathrm{op}}
\end{aligned}$$

□

The transitive closure of the reversed least fixed-point can be related to the transitive closure of the congruence candidate, but only when both are known to be compatible.

Lemma 5.4.16 Let $\mathcal{R} \in Rel_0(\mathcal{E})$ be a preorder such that both of the following relations are compatible:

$$\begin{aligned}
\mathcal{S}_1 &\stackrel{\mathrm{def}}{=} Cand(\mathcal{E}, \mathcal{R})^+ = (\mu\mathcal{T} \,.\, Cmp(\mathcal{E}, \mathcal{T}); Opn(\mathcal{E}, \mathcal{R}))^+ \in Rel(\mathcal{E}) \\
\mathcal{S}_2 &\stackrel{\mathrm{def}}{=} (\mu\mathcal{T} \,.\, Opn(\mathcal{E}, \mathcal{R}); Cmp(\mathcal{E}, \mathcal{T}))^+ \in Rel(\mathcal{E})
\end{aligned}$$

Then:

1. $\mathcal{S}_1 = \mathcal{S}_2$

2. $Cand(\mathcal{E},\mathcal{R}^{\mathrm{op}})^{+} = (Cand(\mathcal{E},\mathcal{R})^{+})^{\mathrm{op}}$

Proof

1. We show that $\mathcal{S}_1 \subseteq \mathcal{S}_2$. The other direction is similar. By transitivity of $\mathcal{S}_2$ it suffices to show $Cand(\mathcal{E},\mathcal{R}) \subseteq \mathcal{S}_2$. This follows by induction from:

   $$Cmp(\mathcal{E},\mathcal{S}_2);Opn(\mathcal{E},\mathcal{R}) \subseteq \mathcal{S}_2$$

   But $\mathcal{S}_2$ is compatible, $Opn(\mathcal{E},\mathcal{R}) \subseteq \mathcal{S}_2$, and $\mathcal{S}_2$ is transitive, so:

   $$Cmp(\mathcal{E},\mathcal{S}_2);Opn(\mathcal{E},\mathcal{R}) \subseteq \mathcal{S}_2;\mathcal{S}_2 \subseteq \mathcal{S}_2$$

   Therefore $\mathcal{S}_1 \subseteq \mathcal{S}_2$.

2. By lemma 5.4.15 and (1):

   $$\begin{array}{rl} & Cand(\mathcal{E},\mathcal{R}^{\mathrm{op}})^{+} \\ = & ((\mu\mathcal{T}\,.\,Opn(\mathcal{E},\mathcal{R});Cmp(\mathcal{E},\mathcal{T}))^{\mathrm{op}})^{+} \\ = & \mathcal{S}_2{}^{\mathrm{op}} \\ = & \mathcal{S}_1{}^{\mathrm{op}} \\ = & (Cand(\mathcal{E},\mathcal{R})^{+})^{\mathrm{op}} \end{array}$$

   Therefore $Cand(\mathcal{E},\mathcal{R}^{\mathrm{op}})^{+} = (Cand(\mathcal{E},\mathcal{R})^{+})^{\mathrm{op}}$.

□

To establish the compatibility hypothesis of lemma 5.4.16 when $\mathcal{R}$ is refinement similarity we face the same problem as for the variants of bisimilarity. In addition, there are no compatible relations that coincide with refinement similarity when states have a type with P-order 1, although any of the other variants of similarity will suffice when the P-order is 0. Therefore the sufficient condition for compatibility that we prove in lemma 5.4.17 is that the language fragment is bounded by 0.

Proposition 5.4.17 If the language fragment $\mathscr{L}(\mathcal{E})$ is bounded by 0, then the hypothesis of lemma 5.4.16 is satisfied when $\mathcal{R}$ is refinement similarity $\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}$.

Proof The argument in proposition 5.4.12 can be modified to show that both of the relations $Cand(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{+}$ and $Cand(\mathcal{E},(\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}})^{+}$ are compatible. The only change is to use the coincidence (see lemma 4.2.5(1)) between, for example, lower similarity and refinement similarity for terms with a type of P-order of 0. If S_1 and S_2 are defined as in lemma 5.4.16, then this shows that S_1 is compatible. Using lemma 5.4.15 we can deduce that S_2 is compatible:

$$\begin{array}{ll} & \mathit{Cmp}(\mathcal{E},\mathcal{S}_2) \\ = & \mathit{Cmp}(\mathcal{E},(\mathit{Cand}(\mathcal{E},(\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}})^{\mathrm{op}})^{+}) \\ = & \mathit{Cmp}(\mathcal{E},\mathit{Cand}(\mathcal{E},(\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}})^{+})^{\mathrm{op}} \\ \subseteq & (\mathit{Cand}(\mathcal{E},(\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}})^{+})^{\mathrm{op}} \\ = & (((\mu\mathcal{T}\,.\,\mathit{Opn}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})});\mathit{Cmp}(\mathcal{E},\mathcal{T}))^{\mathrm{op}})^{+})^{\mathrm{op}} \\ = & (\mu\mathcal{T}\,.\,\mathit{Opn}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})});\mathit{Cmp}(\mathcal{E},\mathcal{T}))^{+} \\ = & \mathcal{S}_2 \end{array}$$

Therefore $\mathcal{S}_1$ and $\mathcal{S}_2$ are compatible. □

Finally, we prove compatibility of the open extensions of refinement similarity and mutual refinement similarity for a collection of language fragments, including those bounded by 0.

Theorem 5.4.18 If the hypothesis of lemma 5.4.16 is satisfied with $\mathcal{R} = \lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}$, then the open extensions $\mathit{Opn}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})$ and $\mathit{Opn}(\mathcal{E},\simeq_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})$ of refinement similarity and mutual refinement similarity are compatible.

Proof We consider refinement similarity first. Again, we want to apply lemma 5.4.7 with $\mathcal{R} = \lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}$ and $\mathcal{S} = \mathit{Cand}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{+}$. Hypothesis (1) is:

$$\mathit{Opn}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}) \subseteq \mathit{Cand}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{+}$$

and follows by lemma 5.4.5(2). Hypothesis (3) is:

$$\mathit{Cand}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{+}[\mathit{Id}(\mathscr{L}(\mathcal{E}))] \subseteq \mathit{Cand}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{+}$$

and can be proven with the argument used in theorem 5.4.14. Hypothesis (2) is:

$$\mathit{Cls}(\mathit{Cand}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{+}) \subseteq \lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}$$

Note that $\mathit{Cls}(\mathit{Cand}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{+}) = \mathit{Cls}(\mathit{Cand}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}))^{+}$. By coinduction, it suffices to show that:

$$\begin{array}{l} \mathit{Cls}(\mathit{Cand}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}))^{+} \subseteq (\langle(\mathit{Cls}(\mathit{Cand}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}))^{+})^{\mathrm{op}}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}} \cap \\ \qquad\qquad\qquad\qquad\qquad \langle \mathit{Cls}(\mathit{Cand}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}))^{+}\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})} \end{array}$$

Both inclusions use an argument similar to that for bisimilarity. In addition, the first inclusion uses lemma 5.4.16(2) to shuffle the dual operation into the congruence candidate. For the first part of the inclusion, we want to show that:

$$\mathit{Cls}(\mathit{Cand}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}))^{+} \subseteq (\langle(\mathit{Cls}(\mathit{Cand}(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}))^{+})^{\mathrm{op}}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}}$$

Or equivalently:

$$(Cls(Cand(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}))^{+})^{\mathrm{op}} \subseteq \langle (Cls(Cand(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}))^{+})^{\mathrm{op}}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$$

And, by lemma 5.4.16(2), this is equivalent to:

$$Cls(Cand(\mathcal{E},(\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}}))^{+} \subseteq \langle Cls(Cand(\mathcal{E},(\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}}))^{+}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$$

As before, lemma 5.4.9(1) can be used to remove the transitive closure operation, so the inclusion above holds if:

$$Cls(Cand(\mathcal{E},(\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}})) \subseteq \langle Cls(Cand(\mathcal{E},(\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}}))\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$$

By proposition 5.4.6(1), this holds if:

$$(\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}} \subseteq \langle (\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{\mathrm{op}}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$$

This follows from the definition of refinement similarity. For the second inclusion, we want to show:

$$Cls(Cand(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}))^{+} \subseteq \langle Cls(Cand(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}))^{+}\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$$

By lemma 5.4.9(2), this holds if:

$$Cls(Cand(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})) \subseteq \langle Cls(Cand(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}))\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$$

And, by proposition 5.4.6(2), this holds if:

$$\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})} \subseteq \langle \lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$$

Which follows immediately from the definition of refinement similarity. Thus hypothesis (2) of lemma 5.4.7 holds.

By applying lemma 5.4.7 we find that $Opn(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})}) = Cand(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})^{+}$. The latter is compatible by assumption, and therefore $Opn(\mathcal{E},\lesssim_{\mathrm{RS}}^{\mathscr{L}_0(\mathcal{E})})$ is compatible. Compatibility of mutual refinement similarity can be deduced from the compatibility of refinement similarity by the argument used in theorem 5.4.8. □

To summarise, we have shown that the open extensions of the lower, upper, and convex variants of similarity and mutual similarity are compatible. If a language fragment is bounded by 1, the open extensions of the lower, upper, and convex variants of bisimilarity are compatible. If a language fragment is bounded by 0, the open extensions of refinement similarity and mutual similarity are compatible.

5.5 Relative Definability

By definition, the language fragments consist of overlapping but different sets of terms. It is useful to know when a term, or one with equivalent behaviour, is a member of a particular fragment, because it may affect the variants of similarity, mutual similarity, and bisimilarity for that fragment. For example, in section 5.6 it is shown that adding countable non-determinism to a finite non-deterministic fragment allows more terms to be discriminated by the upper and convex variants of similarity, mutual similarity, and bisimilarity. In this section, we examine the more general notion of *relative definability* between terms with respect to convex bisimilarity.

Definition 5.5.1 Consider types σ and τ, and programs $M \in \mathscr{M}_0(\tau)$ and $N \in \mathscr{M}_0(\sigma)$. The program M is ***relatively definable*** in terms of N with respect to convex bisimilarity on $\mathscr{M}_0$, denoted $M \leq_{\mathrm{CB}}^{\mathscr{M}_0} N$, if there exists a program $L \in \mathscr{L}_0(\emptyset)(\sigma \to \tau)$ such that $M \simeq_{\mathrm{CB}}^{\mathscr{M}_0} LN$. The notation $M =_{\mathrm{CB}}^{\mathscr{M}_0} N$ means $M \leq_{\mathrm{CB}}^{\mathscr{M}_0} N$ and $N \leq_{\mathrm{CB}}^{\mathscr{M}_0} M$.

Only deterministic, recursive programs can be used to define one program in terms of another because L ranges over $\mathscr{L}_0(\emptyset)$. This definition of relative definability can be generalised by replacing the TTS $\mathscr{M}_0$ (see definition 5.4.13) with others such as $\mathscr{L}_0$, and replacing $\mathscr{L}_0(\emptyset)$ with a distinguished set of states of function type from the new TTS. In addition, other equivalence relations could be used instead of convex bisimilarity. However, the extra generality (and notational complexity) is not required here, because the TTS $\mathscr{M}_0$ contains all of the non-deterministic and non-recursive elements that we wish to compare, yet convex bisimilarity is still compatible on $\mathscr{M}_0$ by theorem 5.4.14.

Note that the program N is passed by name and so may be evaluated more than once. This is important when N is non-deterministic.

As with other forms of relative definability (see [Rog67]), the relation $\leq_{\mathrm{CB}}^{\mathscr{M}_0}$ is a preorder and the corresponding partial order forms an upper semilattice. In addition, $\leq_{\mathrm{CB}}^{\mathscr{M}_0}$ contains convex bisimilarity, and programs from $\mathscr{L}_0(\emptyset)$ are minimal elements.

Lemma 5.5.2

1. The relative definability relation $\leq_{\mathrm{CB}}^{\mathscr{M}_0}$ is a preorder on $\mathscr{M}_0$.
2. For all $M, N \in \mathscr{M}_0$, the program $\mathsf{tuple}\langle M,N\rangle$ is a join of M and N with respect to the preorder $\leq_{\mathrm{CB}}^{\mathscr{M}_0}$, i.e., $M \leq_{\mathrm{CB}}^{\mathscr{M}_0} \mathsf{tuple}\langle M,N\rangle$, $N \leq_{\mathrm{CB}}^{\mathscr{M}_0} \mathsf{tuple}\langle M,N\rangle$, and, for all $L \in \mathscr{M}_0$, $M \leq_{\mathrm{CB}}^{\mathscr{M}_0} L$ and $N \leq_{\mathrm{CB}}^{\mathscr{M}_0} L$ implies $\mathsf{tuple}\langle M,N\rangle \leq_{\mathrm{CB}}^{\mathscr{M}_0} L$.
3. For all $M, N \in \mathscr{M}_0$, $M \simeq_{\mathrm{CB}}^{\mathscr{M}_0} N$ implies $M \leq_{\mathrm{CB}}^{\mathscr{M}_0} N$.
4. For all $M \in \mathscr{L}_0(\emptyset)$ and $N \in \mathscr{M}_0$, $M \leq_{\mathrm{CB}}^{\mathscr{M}_0} N$.

Proof

1. If $M \in \mathscr{M}_0(\sigma)$, then $\lambda x.x \in \mathscr{L}_0(\emptyset)(\sigma \to \sigma)$ and $M \simeq_{\mathrm{CB}}^{\mathscr{M}_0} (\lambda x.x)M$. Therefore $M \leq_{\mathrm{CB}}^{\mathscr{M}_0} M$ and $\leq_{\mathrm{CB}}^{\mathscr{M}_0}$ is reflexive. Now suppose $M_1 \in \mathscr{M}_0(\sigma_1)$, $M_2 \in \mathscr{M}_0(\sigma_2)$, and $M_3 \in \mathscr{M}_0(\sigma_3)$ satisfy $M_1 \leq_{\mathrm{CB}}^{\mathscr{M}_0} M_2$ and $M_2 \leq_{\mathrm{CB}}^{\mathscr{M}_0} M_3$. Then there exist $L_1 \in \mathscr{L}_0(\emptyset)(\sigma_2 \to \sigma_1)$ and $L_2 \in$

$\mathscr{L}_0(\emptyset)(\sigma_3 \to \sigma_2)$ such that $M_1 \simeq^{\mathscr{M}_0}_{\text{CB}} L_1\, M_2$ and $M_2 \simeq^{\mathscr{M}_0}_{\text{CB}} L_2\, M_3$. By compatibility of convex bisimilarity, $L_1\, M_2 \simeq^{\mathscr{M}_0}_{\text{CB}} L_1\,(L_2\, M_3)$, so:

$$M_1 \simeq^{\mathscr{M}_0}_{\text{CB}} (\lambda x.\, L_1\,(L_2\, x))\, M_3$$

We have $\lambda x.\, L_1\,(L_2\, x) \in \mathscr{L}_0(\emptyset)(\sigma_3 \to \sigma_1)$. Therefore $M_1 \leq^{\mathscr{M}_0}_{\text{CB}} M_3$, and $\leq^{\mathscr{M}_0}_{\text{CB}}$ is transitive.

2. Consider $M \in \mathscr{M}_0(\sigma_1)$ and $N \in \mathscr{M}_0(\sigma_2)$. Then we have $M \leq^{\mathscr{M}_0}_{\text{CB}} \mathsf{tuple}\,\langle M, N\rangle$, because $\lambda x.\, \mathsf{proj}\,0\,\mathsf{of}\,x \in \mathscr{L}_0(\emptyset)(\sigma_1 \times \sigma_2)$ and $M \simeq^{\mathscr{M}_0}_{\text{CB}} (\lambda x.\, \mathsf{proj}\,0\,\mathsf{of}\,x)\,(\mathsf{tuple}\,\langle M, N\rangle)$. Similarly, we have $N \leq^{\mathscr{M}_0}_{\text{CB}} \mathsf{tuple}\,\langle M, N\rangle$. Now suppose that $L \in \mathscr{M}_0(\tau)$ is such that $M \leq^{\mathscr{M}_0}_{\text{CB}} L$ and $N \leq^{\mathscr{M}_0}_{\text{CB}} L$. There must exist $L_1 \in \mathscr{L}_0(\emptyset)(\tau \to \sigma_1)$ and $L_2 \in \mathscr{L}_0(\emptyset)(\tau \to \sigma_2)$ such that $M \simeq^{\mathscr{M}_0}_{\text{CB}} L_1\, L$ and $N \simeq^{\mathscr{M}_0}_{\text{CB}} L_2\, L$. By compatibility of convex bisimilarity, $\mathsf{tuple}\,\langle M, N\rangle \simeq^{\mathscr{M}_0}_{\text{CB}} \mathsf{tuple}\,\langle L_1\, L, L_2\, L\rangle$, and so:

$$\mathsf{tuple}\,\langle M, N\rangle \simeq^{\mathscr{M}_0}_{\text{CB}} (\lambda x.\, \mathsf{tuple}\,\langle L_1\, x, L_2\, x\rangle)\, L$$

We have $\lambda x.\, \mathsf{tuple}\,\langle L_1\, x, L_2\, x\rangle \in \mathscr{L}_0(\emptyset)(\tau \to \sigma_1 \times \sigma_2)$. Therefore $\mathsf{tuple}\,\langle M, N\rangle \leq^{\mathscr{M}_0}_{\text{CB}} L$.

3. Consider $M, N \in \mathscr{M}_0(\sigma)$ such that $M \simeq^{\mathscr{M}_0}_{\text{CB}} N$. Then $M \leq^{\mathscr{M}_0}_{\text{CB}} N$, because of the identity function $\lambda x.\, x \in \mathscr{L}_0(\emptyset)(\sigma \to \sigma)$ and $M \simeq^{\mathscr{M}_0}_{\text{CB}} (\lambda x.\, x)\, N$.

4. Consider $M \in \mathscr{L}_0(\emptyset)(\sigma)$ and $N \in \mathscr{M}_0(\tau)$. Then $M \leq^{\mathscr{M}_0}_{\text{CB}} N$, because of the constant function $\lambda x.\, M \in \mathscr{L}_0(\emptyset)(\tau \to \sigma)$ and $M \simeq^{\mathscr{M}_0}_{\text{CB}} (\lambda x.\, M)\, N$.

□

We examine relative definability upon the programs of $\mathscr{M}_0$ with type $\mathsf{P}_\perp(\mathsf{nat})$. The convex bisimilarity equivalence classes of such programs are in bijection with the non-empty subsets of $\omega_\perp$. With the previous lemma, every program of type $\mathsf{P}_\perp(\mathsf{nat})$ is in the same relative definability equivalence class as a program of a certain form.

Lemma 5.5.3 If $M \in \mathscr{M}_0(\mathsf{P}_\perp(\mathsf{nat}))$, then there exists $\kappa > 0$ and a strictly increasing sequence of natural numbers $\langle a_n \mid n < \kappa\rangle$ such that $M =^{\mathscr{M}_0}_{\text{CB}} \Omega \cup ?\langle \underline{a_n} \mid n < \kappa\rangle$ or $M =^{\mathscr{M}_0}_{\text{CB}} ?\langle \underline{a_n} \mid n < \kappa\rangle$.

Proof By lemma 5.5.2(3), it suffices to prove the statement for convex bisimilarity in place of $=^{\mathscr{M}_0}_{\text{CB}}$. However, we first have to exclude the case when M has no convergent behaviour, i.e., $M \simeq^{\mathscr{M}_0}_{\text{CB}} \Omega$. In this case, we can use $\Omega =^{\mathscr{M}_0}_{\text{CB}} ?\langle \underline{0}\rangle$, which follows from lemma 5.5.2(4). Now suppose that M has at least one convergent behaviour. For all canonical programs K such that $M \Downarrow^{\text{may}} K$, it can be shown that there exists $m \in \omega$ such that $K \simeq^{\mathscr{M}_0}_{\text{CB}} [\underline{m}]$. Let $?\langle a_n \mid n < \kappa\rangle$ be the strictly increasing sequence of all such natural numbers. It follows that $M \simeq^{\mathscr{M}_0}_{\text{CB}} \Omega \cup ?\langle \underline{a_n} \mid n < \kappa\rangle$, when $M \Uparrow^{\text{may}}$, and otherwise $M \simeq^{\mathscr{M}_0}_{\text{CB}} ?\langle \underline{a_n} \mid n < \kappa\rangle$. □

There are identifications between the programs in lemma 5.5.3. Lemma 5.5.4 shows that there are only four equivalence classes when κ is finite.

Lemma 5.5.4 For $m,n \geq 2$, consider strictly increasing sequences of natural numbers $a_1,\dots,a_m$ and $b_1,\dots,b_n$. Then we have the following equivalences:

1. $?\langle\underline{a_1}\rangle =^{\mathscr{M}_0}_{\mathrm{CB}} ?\langle\underline{b_1}\rangle$
2. $\Omega\cup?\langle\underline{a_1}\rangle =^{\mathscr{M}_0}_{\mathrm{CB}} \Omega\cup?\langle\underline{b_1}\rangle$
3. $\Omega\cup?\langle\underline{a_1},\dots,\underline{a_m}\rangle =^{\mathscr{M}_0}_{\mathrm{CB}} \Omega\cup?\langle\underline{b_1},\dots,\underline{b_n}\rangle$
4. $?\langle\underline{a_1},\dots,\underline{a_m}\rangle =^{\mathscr{M}_0}_{\mathrm{CB}} ?\langle\underline{b_1},\dots,\underline{b_n}\rangle$

And the following strict inequivalences:

5. $?\langle\underline{a_1}\rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega\cup?\langle\underline{b_1}\rangle$
6. $\Omega\cup?\langle\underline{a_1}\rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega\cup?\langle\underline{b_1},\dots,\underline{b_n}\rangle$
7. $\Omega\cup?\langle\underline{a_1},\dots,\underline{a_m}\rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\langle\underline{b_1},\dots,\underline{b_n}\rangle$

Proof

1. Use lemma 5.5.2(3,4) and $?\langle\underline{a_1}\rangle \simeq^{\mathscr{M}_0}_{\mathrm{CB}} [\underline{a_1}] =^{\mathscr{M}_0}_{\mathrm{CB}} [\underline{b_1}] \simeq^{\mathscr{M}_0}_{\mathrm{CB}} ?\langle\underline{b_1}\rangle$.
2. The inequivalence $\Omega\cup?\langle\underline{a_1}\rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega\cup?\langle\underline{b_1}\rangle$ follows from:

   $$\Omega\cup?\langle\underline{a_1}\rangle \simeq^{\mathscr{M}_0}_{\mathrm{CB}} (\lambda x.\,\mathsf{let}\,y \Leftarrow x\,\mathsf{in}\,[\underline{a_1}])\,(\Omega\cup?\langle\underline{b_1}\rangle)$$

   The other direction is similar.
3. We show $\Omega\cup?\langle\underline{a_1},\dots,\underline{a_m}\rangle =^{\mathscr{M}_0}_{\mathrm{CB}} \Omega\cup?\langle\underline{0},\underline{1}\rangle$, and the result follows by transitivity. For the inequivalence $\Omega\cup?\langle\underline{0},\underline{1}\rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega\cup?\langle\underline{a_1},\dots,\underline{a_m}\rangle$, use the function:

   $$\lambda x.\,\mathsf{let}\,y \Leftarrow x\,\mathsf{in}\,\mathsf{let}\,z \Leftarrow x\,\mathsf{in}\,\mathsf{if}\,\mathsf{lt}\,(y,z)\,\mathsf{then}\,[\underline{0}]\,\mathsf{else}\,[\underline{1}]$$

   For $\Omega\cup?\langle\underline{a_1},\dots,\underline{a_m}\rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega\cup?\langle\underline{0},\underline{1}\rangle$, use:

   $$\begin{array}{l} \lambda x.\,\mathsf{let}\,y_1 \Leftarrow x\,\mathsf{in}\,\dots\mathsf{let}\,y_{m-1} \Leftarrow x\,\mathsf{in} \\ \qquad \mathsf{if}\,\mathsf{eq}\,(y_1,\underline{0})\,\mathsf{then}\,[\underline{a_1}]\,\mathsf{else} \\ \qquad \mathsf{if}\,\mathsf{eq}\,(y_2,\underline{0})\,\mathsf{then}\,[\underline{a_2}]\,\mathsf{else} \\ \qquad\quad \vdots \\ \qquad \mathsf{if}\,\mathsf{eq}\,(y_{m-1},\underline{0})\,\mathsf{then}\,[\underline{a_{m-1}}]\,\mathsf{else} \\ \qquad [\underline{a_{\mathrm{m}}}] \end{array}$$

4. Show $?\langle \underline{a_1}, \ldots, \underline{a_m} \rangle =^{\mathscr{M}_0}_{\mathrm{CB}} ?\langle \underline{0}, \underline{1} \rangle$ using the functions from (3).

5. Use lemma 5.5.2(3,4) and $?\langle \underline{a_1} \rangle \simeq^{\mathscr{M}_0}_{\mathrm{CB}} [\underline{a_1}]$. Now, if $M \in \mathscr{L}_0(\emptyset)(\mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{nat}))$ then $M\,(?\langle \underline{a_1} \rangle) \simeq^{\mathscr{M}_0}_{\mathrm{CB}} M\,[\underline{a_1}]$ either diverges or converges, but not both, so:

$$\Omega \cup ?\langle \underline{b_1} \rangle \not\simeq^{\mathscr{M}_0}_{\mathrm{CB}} M\,(?\langle \underline{a_1} \rangle)$$

Therefore the inequivalence is strict.

6. Use the function from (2) for the inequivalence. To see that the inequivalence is strict, first note that $\Omega \cup [\underline{0}] =^{\mathscr{M}_0}_{\mathrm{CB}} \Omega \cup ?\langle \underline{a_1} \rangle$. Consider the set of programs:

$$X \stackrel{\mathrm{def}}{=} \{M[\Omega \cup [\underline{0}]/x] \mid x : \mathsf{P}_\perp(\mathsf{nat}) \vdash M \in \mathscr{L}(\emptyset) : \mathsf{P}_\perp(\mathsf{nat})\}$$

We claim that every program in X converges to at most one canonical program. Consider $M[\Omega \cup [\underline{0}]/x] \in X$, where $M \in \mathscr{L}(\emptyset)$. By lemmas 3.4.6 and 3.4.10(2), M is canonical, makes a reduction independently of the substitutions, or is blocked on x. If M is canonical we are done. If $M \twoheadrightarrow N$, then $M \twoheadrightarrow_{\mathrm{det}} N$ because $M \in \mathscr{L}(\emptyset)$. By lemma 3.4.10(1), $M[\Omega \cup [\underline{0}]/x] \twoheadrightarrow_{\mathrm{det}} N[\Omega \cup [\underline{0}]/x]$. Otherwise $M \not\downarrow x$, and by lemma 3.4.11(2):

$$M[\Omega \cup [\underline{0}]/x] = M[x \mapsto \Omega \cup [\underline{0}]][\Omega \cup [\underline{0}]/x]$$

By lemmas 3.4.11(3) and 3.4.10(1), $M[x \mapsto \Omega \cup [\underline{0}]][\Omega \cup [\underline{0}]/x]$ may reduce in several steps to either $M[x \mapsto \Omega][\Omega \cup [\underline{0}]/x]$ or $M[x \mapsto [\underline{0}]][\Omega \cup [\underline{0}]/x]$, and every reduction sequence passes through one of these terms. The former program $M[x \mapsto \Omega][\Omega \cup [\underline{0}]/x] \simeq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega$ always diverges. For the latter, $M[x \mapsto [\underline{0}]][\Omega \cup [\underline{0}]/x] \in X$ because $M[x \mapsto [\underline{0}]] \in \mathscr{L}(\emptyset)$. Therefore every program in X has at most one convergent reduction sequence, although it may have one or more divergent reduction sequences.

7. For the inequivalence, by (4), it suffices to show $\Omega \cup ?\langle \underline{a_1}, \ldots, \underline{a_m} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\langle \underline{a_1}, \ldots, \underline{a_m} \rangle$ using the function:

$$\lambda x.\, \mathsf{let}\, y \Leftarrow x \,\mathsf{in}\, \mathsf{let}\, z \Leftarrow x \,\mathsf{in}\, \mathsf{if}\, \mathsf{lt}\,(y,z)\, \mathsf{then}\, \Omega\, \mathsf{else}\, [y]$$

For strictness, use the technique from (6) to show that every term from $\mathscr{L}(\emptyset)$ with a closing substitution of $\Omega \cup ?\langle \underline{a_1}, \ldots, \underline{a_m} \rangle$ for a variable x either has exactly one reduction to another such term, or has at least one reduction to a program that always diverges. Therefore $?\langle \underline{a_1}, \ldots, \underline{a_m} \rangle$ cannot be defined.

□

Therefore we can choose representatives from each equivalence class to obtain the following chain:

$$?\langle \underline{0} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega \cup ?\langle \underline{0} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega \cup ?\langle \underline{0}, \underline{1} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\langle \underline{0}, \underline{1} \rangle$$

For the infinite case, it is natural to start with $\Omega \cup ?\underline{\omega}$ and $?\underline{\omega}$. The former program lies in the same relative definability equivalence class as $\Omega \cup ?\langle \underline{0}, \underline{1} \rangle$, and so is strictly less expressive than $?\langle \underline{0}, \underline{1} \rangle$. On the other hand, $?\underline{\omega}$ is strictly more expressive than $?\langle \underline{0}, \underline{1} \rangle$, and so extends the above chain.

Lemma 5.5.5

1. $\Omega \cup ?\langle \underline{0}, \underline{1} \rangle =^{\mathscr{M}_0}_{\mathrm{CB}} \Omega \cup ?\underline{\omega}$

2. $?\langle \underline{0}, \underline{1} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{\omega}$ and this inequivalence is strict.

Proof For $\Omega \cup ?\langle \underline{0}, \underline{1} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega \cup ?\underline{\omega}$ and $?\langle \underline{0}, \underline{1} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{\omega}$ use the function:

$$\lambda x.\, \mathsf{let}\, y \Leftarrow x\, \mathsf{in}\, \mathsf{let}\, z \Leftarrow x\, \mathsf{in}\, \mathsf{if}\, \mathsf{lt}\,(y,z)\, \mathsf{then}\, [\underline{0}]\, \mathsf{else}\, [\underline{1}]$$

1. For $\Omega \cup ?\underline{\omega} \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega \cup ?\langle \underline{0}, \underline{1} \rangle$, use the function:

$$\lambda w.\, \mathsf{fix} x.\, \mathsf{let}\, y \Leftarrow w\, \mathsf{in}\, \mathsf{if}\, \mathsf{eq}\,(y,\underline{0})\, \mathsf{then}\, [\underline{0}]\, \mathsf{else}\, \mathsf{let}\, z \Leftarrow x\, \mathsf{in}\, [\mathsf{plus}\,(z,\underline{1})]$$

2. Strictness follows from lemma 3.4.8(2).

□

Now we can show that $\Omega \cup ?\underline{\omega}$ and $?\underline{\omega}$ are the least expressive of the infinitely non-deterministic terms of type $\mathsf{P}_\perp(\mathsf{nat})$. In particular, we have that the equivalence class of $?\underline{\omega}$ is the unique successor of the equivalence class of $?\langle \underline{0}, \underline{1} \rangle$ with respect to relative definability. The equivalence classes of $\Omega \cup ?\underline{\omega}$ and $?\underline{\omega}$ consist of all recursively enumerable sets of natural numbers with and without divergence respectively, following theorem 5.3 of [AP86].

Lemma 5.5.6 Suppose that $A \subseteq_{\mathrm{ne}} \omega$ is an infinite set. Then:

1. $\Omega \cup ?\underline{\omega} \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega \cup ?\underline{A}$

2. $?\underline{\omega} \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{A}$

In addition, the opposite inequivalences hold if and only if A is recursively enumerable:

3. $\Omega \cup ?\underline{A} \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega \cup ?\underline{\omega}$

4. $?\underline{A} \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{\omega}$

Proof We use the same function for (1) and (2). First define a term:

$$w : \mathsf{P}_\perp(\mathsf{nat}) \vdash \mathsf{choose} \in \mathscr{L}(\emptyset) : \mathsf{P}_\perp(\mathsf{nat})$$

by:

$$\begin{aligned}\mathsf{choose} \stackrel{\mathrm{def}}{=} {} & \mathsf{ffix}\, f{:}\mathsf{nat} \to \mathsf{P}_\perp(\mathsf{nat}).\\ & \quad \lambda x{:}\mathsf{nat}.\\ & \quad\quad \mathsf{if}\, \mathsf{eq}\,(x,\underline{0})\, \mathsf{then}\, [\underline{0}]\, \mathsf{else}\\ & \quad\quad\quad \mathsf{let}\, y \Leftarrow w\, \mathsf{in}\, \mathsf{let}\, z \Leftarrow w\, \mathsf{in}\, \mathsf{if}\, \mathsf{lt}\,(y,z)\, \mathsf{then}\, f\,(\mathsf{minus}\,(x,\underline{1}))\, \mathsf{else}\, [x]\end{aligned}$$

Suppose that M is a non-deterministic program of type $\mathsf{P}_\perp(\mathtt{nat})$ that can converge to at least two programs representing different natural numbers. Then, for any natural number n, the program $\mathsf{choose}[M/w]\,\underline{n}$ converges to canonical programs that, up to convex bisimilarity, are of the form $[\underline{m}]$, where $0 \leq m \leq n$. If M cannot diverge, then $\mathsf{choose}[M/w]\,\underline{n}$ cannot diverge. Then the function $\lambda w.\,\mathsf{let}\,x \Leftarrow w\,\mathsf{in}\,\mathsf{choose}\,x$ can be used to define $\Omega\cup ?\underline{\omega}$ and $?\underline{\omega}$ in terms of $\Omega\cup ?\underline{A}$ and $?\underline{A}$ respectively, because A is infinite and hence unbounded.

We consider (3) and (4) together. Suppose that A is recursively enumerable. We assume the existence of a program $M \in \mathscr{L}_0(\emptyset)(\mathtt{nat} \to \mathsf{P}_\perp(\mathtt{nat}))$ such that, for all $m \in \omega$, $M\,\underline{m} \Downarrow^{\text{must}}$ and:

$$A = \{n \in \omega \mid \exists m \in \omega.\, \exists N.\, M\,\underline{m} \Downarrow^{\text{may}} [N] \wedge N \simeq^{\mathscr{M}_0}_{\text{CB}} \underline{n}\}$$

That is, A is the image of a deterministic and total function M. The function $\lambda x.\,\mathsf{let}\,y \Leftarrow x\,\mathsf{in}\,M\,y$ can be used to define $\Omega\cup ?\underline{A}$ and $?\underline{A}$ in terms of $\Omega\cup ?\underline{\omega}$ and $?\underline{\omega}$ respectively. For the other direction, suppose that $\Omega\cup ?\underline{A} \leq^{\mathscr{M}_0}_{\text{CB}} \Omega\cup ?\underline{\omega}$, so there exists a program $L \in \mathscr{L}_0(\emptyset)(\mathsf{P}_\perp(\mathtt{nat}) \to \mathsf{P}_\perp(\mathtt{nat}))$ such that $\Omega\cup ?\underline{A} \simeq^{\mathscr{M}_0}_{\text{CB}} L\,(\Omega\cup ?\underline{\omega})$. We can encode terms of $\mathscr{L}(\Omega\cup ?\underline{\omega})$ as natural numbers so that it is decidable whether an encoded reduction is valid or not. Therefore we can recursively enumerate the natural numbers that $L\,(\Omega\cup ?\underline{\omega})$ produces (converges to the lift of a convex bisimilar program), and A is recursively enumerable. The same argument applies to $?\underline{A} \leq^{\mathscr{M}_0}_{\text{CB}} ?\underline{\omega}$. □

This leaves the non-recursively enumerable sets which are only known to be strictly more expressive than $\Omega\cup ?\underline{\omega}$ or $?\underline{\omega}$, depending on whether or not divergence is possible. Note that any program of the form $?\underline{A}$, where $A \subseteq_{\text{ne}} \omega$, can be defined in terms of $?\underline{\omega}$ and an infinite case statement (not necessarily recursive).

It is appealing to relate $\leq^{\mathscr{M}_0}_{\text{CB}}$ on programs of type $\mathsf{P}_\perp(\mathtt{nat})$ with reducibility concepts found in the literature (see [Rog67, Odi89]). For example, for sets $A, B \subseteq \omega$, A is *Turing reducible* to B, written $A \leq_{\text{T}} B$, if the characteristic function of A is definable in terms of the characteristic function of B.

Programs of type $\mathsf{P}_\perp(\mathtt{nat})$ can be used as "unhelpful" characteristic functions that may give a correct positive answer or no information at all. For example, for a non-empty set $A \subseteq_{\text{ne}} \omega$, define the program $M \in \mathscr{M}_0(\mathtt{nat} \to \mathsf{P}_\perp(\mathtt{bool}))$ by:

$$M \stackrel{\text{def}}{=} \lambda x.\,\mathsf{let}\,y \Leftarrow ?\underline{A}\,\mathsf{in}\,[\mathsf{eq}\,(x,y)]$$

When M is applied, it chooses a number from the set A and compares it to its argument. If we define, for all $n \in \omega$, a program N_n by:

$$N_n \stackrel{\text{def}}{=} \begin{cases} ?\langle \mathsf{false} \rangle & \text{if } n \notin A \\ ?\langle \mathsf{false}, \mathsf{true} \rangle & \text{if } n \in A \end{cases}$$

then:

$$M \simeq^{\mathscr{M}_0}_{\text{CB}} \lambda x.\,\mathsf{case}\,x\,\mathsf{of}\,\langle y_n.N_n \mid n < \omega \rangle$$

This property is insufficient to relate $\leq^{\mathscr{M}_0}_{\text{CB}}$ on $\mathscr{M}_0(\mathsf{P}_\perp(\mathtt{nat}))$ with Turing reducibility because it is not possible to distinguish a negative answer from an answer with no information. However, if we consider the join of $?\underline{A}$ and $?\underline{(\omega\setminus A)}$, then we may receive positive and negative answers, and so obtain a a relationship with Turing reducibility. We first show that finite joins of non-divergent elements exist in $\mathscr{M}_0(\mathsf{P}_\perp(\mathsf{nat}))$.

Lemma 5.5.7 If $A, B \subseteq_{\mathrm{ne}} \omega$, then there exists $C \subseteq_{\mathrm{ne}} \omega$ such that $\mathsf{tuple}\,\langle ?\underline{A}, ?\underline{B}\rangle =^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{C}$.

Proof Without loss of generality, assume that A and B are infinite. Define C by:

$$C \stackrel{\mathrm{def}}{=} \{2m \mid m \in A\} \cup \{2m+1 \mid m \in B\}$$

We assume the existence of a program $\mathsf{split} \in \mathscr{L}_0(\emptyset)(\mathsf{nat} \to \mathsf{P}_\perp(\mathsf{nat}+\mathsf{nat}))$ such that, for all $m \in \omega$, $\mathsf{split}\,\underline{2m} \simeq^{\mathscr{M}_0}_{\mathrm{CB}} [\mathsf{inj}\,0\,\mathsf{of}\,\underline{m}]$ and $\mathsf{split}\,\underline{2m+1} \simeq^{\mathscr{M}_0}_{\mathrm{CB}} [\mathsf{inj}\,1\,\mathsf{of}\,\underline{m}]$. For $?\underline{C} \leq^{\mathscr{M}_0}_{\mathrm{CB}} \mathsf{tuple}\,\langle ?\underline{A}, ?\underline{B}\rangle$, use the function:

$$\begin{array}{l}\lambda p{:}\mathsf{P}_\perp(\mathsf{nat}) \times \mathsf{P}_\perp(\mathsf{nat}).\\ \quad \mathsf{let}\,x \Leftarrow \mathsf{proj}\,0\,\mathsf{of}\,p\,\mathsf{in}\,\mathsf{let}\,y \Leftarrow \mathsf{proj}\,0\,\mathsf{of}\,p\,\mathsf{in}\\ \quad\quad \mathsf{if}\,\mathsf{lt}\,(x,y)\,\mathsf{then}\,[\mathsf{mult}\,(\underline{2},x)]\,\mathsf{else}\,\mathsf{let}\,z \Leftarrow \mathsf{proj}\,1\,\mathsf{of}\,p\,\mathsf{in}\,[\mathsf{plus}\,(\mathsf{mult}\,(\underline{2},z),\underline{1})]\end{array}$$

In the other direction, fix $m \in A$ and $n \in B$. Then, for $\mathsf{tuple}\,\langle ?\underline{A}, ?\underline{B}\rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{C}$, use the function:

$$\begin{array}{l}\lambda x{:}\mathsf{P}_\perp(\mathsf{nat}).\\ \quad \mathsf{tuple}\,\langle \mathsf{let}\,y \Leftarrow x\,\mathsf{in}\,\mathsf{case}\,\mathsf{split}\,y\,\mathsf{of}\,\langle z_0.[z_0], z_1.[\underline{m}]\rangle,\\ \quad\quad\quad \mathsf{let}\,y \Leftarrow x\,\mathsf{in}\,\mathsf{case}\,\mathsf{split}\,y\,\mathsf{of}\,\langle z_0.[\underline{n}], z_1.[z_1]\rangle\rangle\end{array}$$

Each component of the pair chooses a number from $?\underline{C}$ and uses split to find out whether it is in A or B. If the component requires a member A but received a member of B, or vice-versa, then it returns the appropriate fixed number. □

Now, if A is Turing reducible to B and B is infinite, $?\underline{A}$ can be defined in terms of $?\underline{B}$ and $?\underline{(\omega\setminus B)}$.

Proposition 5.5.8 Consider $A, B \subsetneq_{\mathrm{ne}} \omega$ such that $A \leq_{\mathrm{T}} B$ and B is infinite. If $?\underline{C}$ is a join of $?\underline{A}$ and $?\underline{(\omega\setminus A)}$, and $?\underline{D}$ is a join of $?\underline{B}$ and $?\underline{(\omega\setminus B)}$, then $?\underline{C} \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{D}$.

Proof We claim that there exists a term M such that $M[?\underline{D}/w]$ tests whether $n \in A$ for some natural number n. There are three possible outcomes, "no", "yes", and "maybe", which we represent using the type $\mathsf{unit}+\mathsf{unit}+\mathsf{unit}$. We require:

$$w : \mathsf{P}_\perp(\mathsf{nat}) \vdash M \in \mathscr{L}(\emptyset) : \mathsf{nat} \to \mathsf{P}_\perp(\mathsf{unit}+\mathsf{unit}+\mathsf{unit})$$

The term M is obtained by modifying the algorithm that defines the characteristic function of A in terms of the characteristic function of B. Wherever the characteristic function of B is invoked in the original algorithm, we should use $?\underline{D}$ to get $?\underline{B}$ and $?\underline{(\omega\setminus B)}$, and then test membership in those sets as described on page 158. If either test produces a positive result, then the algorithm can continue, otherwise the algorithm immediately returns "maybe". Thus the behaviour of M is described by, for all $n \in \omega$:

$$M[?\underline{D}/w]\,\underline{n} \simeq^{\mathscr{M}_0}_{\mathrm{CB}} \begin{cases} ?\langle \mathsf{inj}\,0\,\mathsf{of}\,\star, \mathsf{inj}\,2\,\mathsf{of}\,\star\rangle & \text{if } n \notin A\\ ?\langle \mathsf{inj}\,1\,\mathsf{of}\,\star, \mathsf{inj}\,2\,\mathsf{of}\,\star\rangle & \text{if } n \in A\end{cases}$$

By fixing $m \in A$, we can now define $?\underline{A}$ by choosing a natural number n and using M to test whether $n \in A$. If $n \in A$, then we return n. If $n \notin A$, or we receive a "maybe" result, then we return m. By lemma 5.5.6, we know that $?\underline{\omega}$ can be defined in terms of $?\underline{D}$ because B is infinite and $?\underline{B} \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{D}$, so we may use the following function to show that $?\underline{A} \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{D}$:

$$\lambda w.\,\mathsf{let}\,x \Leftarrow ?\underline{\omega}\,\mathsf{in}\,\mathsf{let}\,y \Leftarrow M\,x\,\mathsf{in}\,\mathsf{casesplit}\,y\,\mathsf{of}\,\langle z_0.[\underline{m}], z_1.[x], z_2.[\underline{m}]\rangle$$

Similarly, by choosing $m \in \omega \setminus A$, we can show that $?\underline{(\omega \setminus A)} \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{D}$ using:

$$\lambda w.\,\mathsf{let}\,x \Leftarrow ?\underline{\omega}\,\mathsf{in}\,\mathsf{let}\,y \Leftarrow M\,x\,\mathsf{in}\,\mathsf{casesplit}\,y\,\mathsf{of}\,\langle z_0.[x], z_1.[\underline{m}], z_2.[\underline{m}]\rangle$$

Therefore $?\underline{C} \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{D}$. □

5.6 Theory of the Language

There are variants of similarity, mutual similarity, and bisimilarity for each language fragment and it is useful to understand the inclusions and differences between these relations. For example, we might ask whether the restriction of $\simeq^{\mathscr{L}_0(?\underline{\omega})}_{\mathrm{CB}}$ to $\mathscr{L}_0(\emptyset)$ coincides with the restriction of $\simeq^{\mathscr{L}_0(?\langle\underline{0},\underline{1}\rangle)}_{\mathrm{CB}}$ to $\mathscr{L}_0(\emptyset)$, i.e., are there deterministic programs that can be distinguished using countably non-deterministic programs, such as $?\underline{\omega}$, but not by finitely non-deterministic programs, such as $?\langle\underline{0},\underline{1}\rangle$. In this section, we show that such programs do exist. Consequently, there can be no single model that is sound and complete for convex bisimilarity on both $\mathscr{L}_0(?\langle\underline{0},\underline{1}\rangle)$ and $\mathscr{L}_0(?\underline{\omega})$.

We begin by considering inclusions between the relations. Lemma 4.2.4 describes the inclusions between the variants of similarity, mutual similarity, and bisimilarity on a fixed language fragment, and these inclusions trivially extend to the open extensions for that fragment. Lemma 5.6.1 shows that the smaller of two comparable fragments has coarser variants of the open extensions of similarity, mutual similarity, and bisimilarity than the larger fragment. The case for convex bisimilarity could also be inferred from propositions 4.5.3 and 4.5.9.

Lemma 5.6.1 Consider sets of well-typed terms $\mathcal{E}_1$ and $\mathcal{E}_2$ such that $\mathscr{L}_0(\mathcal{E}_1) \subseteq \mathscr{L}_0(\mathcal{E}_2)$. Suppose that $\mathcal{R}_1 \in Rel_0(\mathcal{E}_1)$ and $\mathcal{R}_2 \in Rel_0(\mathcal{E}_2)$ are the same variant of similarity, mutual similarity, or bisimilarity on $\mathscr{L}_0(\mathcal{E}_1)$ and $\mathscr{L}_0(\mathcal{E}_2)$ respectively, e.g., $\mathcal{R}_1 = \simeq^{\mathscr{L}_0(\mathcal{E}_1)}_{\mathrm{CB}}$ and $\mathcal{R}_2 = \simeq^{\mathscr{L}_0(\mathcal{E}_2)}_{\mathrm{CB}}$. If $M,N \in \mathscr{L}(\mathcal{E}_1)$ and $\Gamma \vdash \langle M,N\rangle \in Opn(\mathcal{E}_2,\mathcal{R}_2) : \sigma$, then $\Gamma \vdash \langle M,N\rangle \in Opn(\mathcal{E}_1,\mathcal{R}_1) : \sigma$.

Proof If $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$, consider programs $L_i \in \mathscr{L}_0(\mathcal{E}_1)(\sigma_i)$, for $1 \leq i \leq n$. We know $\langle M[\vec{L}/x], N[\vec{L}/x]\rangle \in \mathcal{R}_2$ and want to show $\langle M[\vec{L}/x], N[\vec{L}/x]\rangle \in \mathcal{R}_1$, where $\vec{L} = L_1,\ldots,L_n$. This can be established using a simple coinduction (or induction because the type system is well-founded). The inclusion $\mathscr{L}_0(\mathcal{E}_1) \subseteq \mathscr{L}_0(\mathcal{E}_2)$ is used to show that $\mathcal{R}_2$ is finer than $\mathcal{R}_1$ on programs of function type because there are more programs to use as tests. □

Therefore $Opn(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}})$ is the finest relation amongst the variants of similarity, mutual similarity, and bisimilarity on the language fragments, and so the majority of rules in lemma 5.6.2 can be used to deduce properties of the other relations. However, as discussed in section 5.4, the open extension of convex bisimilarity upon $\mathscr{L}_0$ is not known to be compatible. Note also that results for refinement similarity can be used to deduce properties of lower similarity and upper similarity because:

$$Opn(\mathscr{L}, \lesssim^{\mathscr{L}_0}_{\mathrm{RS}}) \subseteq Opn(\mathscr{L}, \lesssim^{\mathscr{L}_0}_{\mathrm{LS}})^{\mathrm{op}} \cap Opn(\mathscr{L}, \lesssim^{\mathscr{L}_0}_{\mathrm{US}})$$

Lemma 5.6.2

1. Consider $M \in \mathscr{L}_0(\mathsf{P}_\perp(\sigma))$ and suppose that $\langle N_n \mid n < \kappa \rangle$ is such that, for all K, $M \Downarrow^{\mathrm{may}} K$ if and only if there exists $n < \kappa$ such that $K = [N_n]$. If $M \Uparrow^{\mathrm{may}}$, then $M \simeq^{\mathscr{L}_0}_{\mathrm{CB}} \Omega \cup ?\langle N_n \mid n < \kappa \rangle$. Otherwise, $M \Downarrow^{\mathrm{must}}$ and $M \simeq^{\mathscr{L}_0}_{\mathrm{CB}} ?\langle N_n \mid n < \kappa \rangle$.
2. The rule schema illustrated in figures 5.5 and 5.6 are valid, where term variables range over $\mathscr{L}$. Side conditions are presented as premises for brevity.

Proof A straightforward analysis of the may convergence and may divergence properties of both sides after performing a closing substitution. □

Restrictions of convex bisimilarity for $\mathscr{L}(\mathcal{E})$ to $\mathscr{L}_0(\emptyset)$ may differ according to the relative definability equivalence classes of the programs in $\mathscr{L}_0(\mathcal{E})$. We now show that there are strict inclusions corresponding to each equivalence class in the chain:

$$?\langle \underline{0} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega \cup ?\langle \underline{0} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega \cup ?\langle \underline{0}, \underline{1} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\langle \underline{0}, \underline{1} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{\omega}$$

In fact, convex bisimilar programs are used instead of $\Omega \cup ?\langle \underline{0} \rangle$ and $\Omega \cup ?\langle \underline{0}, \underline{1} \rangle$ to sidestep the closure conditions on fragments that would force $?\langle \underline{0}, \underline{1} \rangle$ into the corresponding fragments, so the chain becomes:

$$?\langle \underline{0} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \mathsf{let}\, x \Leftarrow ?\langle \Omega, [\underline{0}] \rangle \,\mathsf{in}\, x \leq^{\mathscr{M}_0}_{\mathrm{CB}} \mathsf{let}\, x \Leftarrow ?\langle \Omega, [\underline{0}], [\underline{1}] \rangle \,\mathsf{in}\, x \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\langle \underline{0}, \underline{1} \rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{\omega}$$

Lemma 5.6.3 is used in proposition 5.6.4 to show that the second and third programs cannot be used to distinguish other programs.

Lemma 5.6.3 Consider a term M such that:

$$x : \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat})) \vdash M \in \mathscr{L}(\emptyset) : \mathsf{P}_\perp(\mathsf{nat})$$

Then:

1. If $M[?\langle \Omega, [\underline{0}] \rangle / x]$ reduces to any canonical program, then there exists $n \in \omega$ such that, for all N, $M[?\langle \Omega, [\underline{0}] \rangle / x] \Downarrow^{\mathrm{may}} [N]$ implies $N \simeq^{\mathscr{L}_0}_{\mathrm{CB}} \underline{n}$.

$$\frac{\Gamma \vdash M \rightarrowtriangle_{\mathrm{det}} N : \sigma}{\Gamma \vdash \langle M, N \rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \sigma}$$

$$\frac{\Gamma \vdash M \rightarrowtriangle N : \sigma}{\Gamma \vdash \langle M, N \rangle \in \mathit{Opn}(\mathscr{L}, \lesssim^{\mathscr{L}_0}_{\mathrm{RS}}) : \sigma}$$

$$\frac{\Gamma \vdash M : \mathsf{sum}\,\langle \sigma_n \mid n < \kappa \rangle}{\Gamma \vdash \langle \mathsf{case}\, M\, \mathsf{of}\, \langle x_n.\mathsf{inj}\, n\, \mathsf{of}\, x_n \mid n < \kappa \rangle, M \rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \mathsf{sum}\,\langle \sigma_n \mid n < \kappa \rangle}$$

$$\frac{\Gamma \vdash M : \mathsf{prod}\,\langle \sigma_n \mid n < \kappa \rangle}{\Gamma \vdash \langle \mathsf{tuple}\,\langle \mathsf{proj}\, n\, \mathsf{of}\, M \mid n < \kappa \rangle, M \rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \mathsf{prod}\,\langle \sigma_n \mid n < \kappa \rangle}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad x \notin \mathit{Dom}(\Gamma)}{\Gamma \vdash \langle \lambda x{:}\sigma.\,(M\,x), M \rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \sigma \to \tau}$$

$$\frac{\Gamma \vdash M : \mathsf{P}_\perp(\sigma) \qquad x \notin \mathit{Dom}(\Gamma)}{\Gamma \vdash \langle \mathsf{let}\, x \Leftarrow M\, \mathsf{in}\, [x], M \rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash L : \mathsf{P}_\perp(\sigma) \qquad \Gamma, x : \sigma \vdash M : \mathsf{P}_\perp(\tau) \qquad \Gamma, y : \tau \vdash N : \mathsf{P}_\perp(\rho)}{\begin{array}{l}\Gamma \vdash \langle \mathsf{let}\, y \Leftarrow (\mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, M)\, \mathsf{in}\, N, \\ \qquad \mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, (\mathsf{let}\, y \Leftarrow M\, \mathsf{in}\, N) \rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \mathsf{P}_\perp(\rho)\end{array}}$$

$$\frac{\Gamma \vdash L : \mathsf{P}_\perp(\sigma) \qquad \Gamma \vdash M : \mathsf{P}_\perp(\tau) \qquad \Gamma, x : \sigma, y : \tau \vdash N : \mathsf{P}_\perp(\rho)}{\begin{array}{l}\Gamma \vdash \langle \mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, (\mathsf{let}\, y \Leftarrow M\, \mathsf{in}\, N), \\ \qquad \mathsf{let}\, y \Leftarrow M\, \mathsf{in}\, (\mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, N) \rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \mathsf{P}_\perp(\rho)\end{array}}$$

Figure 5.5: Reduction, η, and sequential composition rules

$$\frac{\Gamma \vdash M : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \langle M \cup M, M\rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash M : \mathsf{P}_\perp(\sigma) \qquad \Gamma \vdash N : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \langle M \cup N, N \cup M\rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash L : \mathsf{P}_\perp(\sigma) \qquad \Gamma \vdash M : \mathsf{P}_\perp(\sigma) \qquad \Gamma \vdash N : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \langle (L \cup M) \cup N, L \cup (M \cup N)\rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\{\Gamma \vdash M_i : \mathsf{P}_\perp(\sigma) \mid 1 \leq i \leq n\} \qquad x \notin \mathit{Dom}(\Gamma)}{\Gamma \vdash \langle M_1 \cup M_2 \cup \ldots \cup M_n, \mathsf{let}\, x \Leftarrow ?\langle M_1, M_2, \ldots, M_n\rangle\, \mathsf{in}\, x\rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\begin{array}{c}\{\Gamma \vdash M_n : \sigma \mid n < \kappa_1\} \\ \{\Gamma \vdash N_n : \sigma \mid n < \kappa_2\} \\ \{M_n \mid n < \kappa_1\} = \{N_n \mid n < \kappa_2\}\end{array}}{\Gamma \vdash \langle ?\langle M_n \mid n < \kappa_1\rangle, ?\langle N_n \mid n < \kappa_2\rangle\rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\begin{array}{c}\{\Gamma \vdash M_n : \sigma \mid n < \kappa_1\} \\ \{\Gamma \vdash N_n : \sigma \mid n < \kappa_2\} \\ \{M_n \mid n < \kappa_1\} \supseteq \{N_n \mid n < \kappa_2\}\end{array}}{\Gamma \vdash \langle ?\langle M_n \mid n < \kappa_1\rangle, ?\langle N_n \mid n < \kappa_2\rangle\rangle \in \mathit{Opn}(\mathscr{L}, \lesssim^{\mathscr{L}_0}_{\mathrm{RS}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash M : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \langle \Omega \cup M, \Omega\rangle \in \mathit{Opn}(\mathscr{L}, \lesssim^{\mathscr{L}_0}_{\mathrm{RS}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash M : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \langle \Omega \cup M, M\rangle \in \mathit{Opn}(\mathscr{L}, \lesssim^{\mathscr{L}_0}_{\mathrm{RS}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash M : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \langle \Omega \cup M, M\rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{LB}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash M : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \langle \Omega \cup M, \Omega\rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{UB}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash \langle M, N\rangle \in \mathit{Opn}(\mathscr{L}, \lesssim^{\mathscr{L}_0}_{\mathrm{LS}}) : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \langle M \cup N, N\rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{LS}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash \langle M, N\rangle \in \mathit{Opn}(\mathscr{L}, \lesssim^{\mathscr{L}_0}_{\mathrm{US}}) : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \langle M, M \cup N\rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{US}}) : \mathsf{P}_\perp(\sigma)}$$

$$\frac{\Gamma \vdash \langle L, M\rangle \in \mathit{Opn}(\mathscr{L}, \lesssim^{\mathscr{L}_0}_{\mathrm{CS}}) : \mathsf{P}_\perp(\sigma) \qquad \Gamma \vdash \langle M, N\rangle \in \mathit{Opn}(\mathscr{L}, \lesssim^{\mathscr{L}_0}_{\mathrm{CS}}) : \mathsf{P}_\perp(\sigma)}{\Gamma \vdash \langle L \cup M \cup N, L \cup N\rangle \in \mathit{Opn}(\mathscr{L}, \simeq^{\mathscr{L}_0}_{\mathrm{CS}}) : \mathsf{P}_\perp(\sigma)}$$

Figure 5.6: Erratic choice rules

2. Suppose there exist terms N_1, N_2 and natural numbers $n_1 \neq n_2$ such that $N_1 \simeq^{\mathscr{L}_0}_{\mathrm{CB}} \underline{n_1}$, $N_2 \simeq^{\mathscr{L}_0}_{\mathrm{CB}} \underline{n_2}$, and:

$$M[?\langle\Omega,[\underline{0}],[\underline{1}]\rangle/x] \Downarrow^{\mathrm{may}} [N_1] \qquad \text{and} \qquad M[?\langle\Omega,[\underline{0}],[\underline{1}]\rangle/x] \Downarrow^{\mathrm{may}} [N_2]$$

Then $M[?\langle\Omega,[\underline{0}],[\underline{1}]\rangle/x] \Uparrow^{\mathrm{may}}$.

Proof More generally, consider a term M such that:

$$\Gamma \vdash M \in \mathscr{L}(\emptyset) : \mathsf{P}_\bot(\mathsf{nat})$$

where $\Gamma = x{:}\mathsf{P}_\bot(\mathsf{P}_\bot(\mathsf{nat})), x_1{:}\mathsf{P}_\bot(\mathsf{nat}), \ldots, x_m{:}\mathsf{P}_\bot(\mathsf{nat})$. By lemma 3.4.6, M is canonical, there exists a unique term N such that $M \rightarrowtriangle N$, or M is blocked on a variable appearing in Γ. If M is canonical, then there exists $n \in \omega$ such that:

$$\Gamma \vdash \langle M,[n]\rangle \in Opn(\mathscr{L},\simeq^{\mathscr{L}_0}_{\mathrm{CB}}) : \mathsf{P}_\bot(\mathsf{nat})$$

The type system prevents variables from Γ from being used in the reduction sequence from the immediate subterm of M, although they may appear as free variables of M. If there exists a unique term N such that $M \rightarrowtriangle N$, then lemma 3.4.10(1) allows substitution of closed terms, such as $?\langle\Omega,[\underline{0}]\rangle$ or $?\langle\Omega,[\underline{0}],[\underline{1}]\rangle$, into both sides. If $M \not\downarrow x$, then $M \neq x$ because they have different types. Therefore there must exist a fresh variable y and terms M' and N such that $M' \not\downarrow y$ and:

$$M = M'[y \mapsto \mathsf{let}\, x_{m+1} \Leftarrow x \,\mathsf{in}\, N]$$

Then:

$$\begin{array}{ll}
& M[?\langle\Omega,[\underline{0}]\rangle/x] \\
= & M'[y \mapsto \mathsf{let}\, x_{m+1} \Leftarrow x \,\mathsf{in}\, N][?\langle\Omega,[\underline{0}]\rangle/x] \\
= & M'[y \mapsto \mathsf{let}\, x_{m+1} \Leftarrow x \,\mathsf{in}\, N][x \mapsto ?\langle\Omega,[\underline{0}]\rangle][?\langle\Omega,[\underline{0}]\rangle/x] \\
= & M'[y \mapsto \mathsf{let}\, x_{m+1} \Leftarrow ?\langle\Omega,[\underline{0}]\rangle \,\mathsf{in}\, N][?\langle\Omega,[\underline{0}]\rangle/x] \\
\rightarrowtriangle & M'[y \mapsto \mathsf{let}\, x_{m+1} \Leftarrow [\Omega] \,\mathsf{in}\, N][?\langle\Omega,[\underline{0}]\rangle/x] \\
\rightarrowtriangle_{\mathrm{det}} & M'[y \mapsto N[\Omega/x_{m+1}]][?\langle\Omega,[\underline{0}]\rangle/x] \\
= & M'[y \mapsto N][?\langle\Omega,[\underline{0}]\rangle/x][\Omega/x_{m+1}]
\end{array}$$

Similarly:

$$M[?\langle\Omega,[\underline{0}]\rangle/x] \rightarrowtriangle^+ M'[y \mapsto N][?\langle\Omega,[\underline{0}]\rangle/x][[\underline{0}]/x_{m+1}]$$

All of the reduction sequences of length two or more from $M[?\langle\Omega,[\underline{0}]\rangle/x]$ pass through instances of $M'[y \mapsto N][?\langle\Omega,[\underline{0}]\rangle/x]$, where x_{m+1} is substituted with either Ω or $[\underline{0}]$. In addition:

$$\Gamma, x_{m+1} : \mathsf{P}_\bot(\mathsf{nat}) \vdash M'[y \mapsto N] \in \mathscr{L}(\emptyset) : \mathsf{P}_\bot(\mathsf{nat})$$

Thus, although reductions are non-deterministic, we do know the general form of the results. A similar argument holds for $M[?\langle\Omega,[\underline{0}],[\underline{1}]\rangle/x]$. Now suppose $M \not\downarrow x_i$, where $1 \leq i \leq m$. If Ω is substituted for x_i in M, then the result always diverges. If $\Delta = \Gamma \setminus \langle x_i, \mathsf{P}_\bot(\mathsf{nat})\rangle$ then:

$$\Delta \vdash M[[\underline{0}]/x_i] \in \mathscr{L}(\emptyset) : \mathsf{P}_\bot(\mathsf{nat})$$
$$\Delta \vdash M[[\underline{1}]/x_i] \in \mathscr{L}(\emptyset) : \mathsf{P}_\bot(\mathsf{nat})$$

Now we can complete both arguments. For $M[?\langle\Omega,[\underline{0}]\rangle/x]$, we iteratively reduce and discard substitutions for new variables that are introduced from $?\langle\Omega,[\underline{0}]\rangle$ until we reach a term that is blocked on a variable x_i. Note that all reductions must be instances of this term. The branch of the reduction tree where Ω is substituted for the blocked variable may be discarded because it never leads to a canonical program, so it is only necessary to consider the branch where $[\underline{0}]$ is substituted for x_i. Therefore all reduction sequences are constrained to either diverge or to lead to canonical programs that are convex bisimilar to one another because the substitutions are no longer relevant. This completes the argument for (1). For $M[?\langle\Omega,[\underline{0}],[\underline{1}]\rangle/x]$, we assume it can converge to at least two canonical programs that are not related by convex bisimilarity and so there must be a branch in the reduction tree. The branch must arise when $[\underline{0}]$ and $[\underline{1}]$ are substituted for some blocked variable x_i, but then Ω can also be substituted for x_i and so the term may diverge. Therefore $M[?\langle\Omega,[\underline{0}],[\underline{1}]\rangle/x]$ may diverge, as required for (2). $\square$

Proposition 5.6.4 Define programs L_1, L_2, L_3, L_4, L_5 by:

$$L_1 \stackrel{\text{def}}{=} ?\langle\underline{0}\rangle$$
$$L_2 \stackrel{\text{def}}{=} \mathsf{let}\, x \Leftarrow ?\langle\Omega,[\underline{0}]\rangle\, \mathsf{in}\, x$$
$$L_3 \stackrel{\text{def}}{=} \mathsf{let}\, x \Leftarrow ?\langle\Omega,[\underline{0}],[\underline{1}]\rangle\, \mathsf{in}\, x$$
$$L_4 \stackrel{\text{def}}{=} ?\langle\underline{0},\underline{1}\rangle$$
$$L_5 \stackrel{\text{def}}{=} ?\underline{\omega}$$

The restrictions of the following relations to $\mathscr{L}_0(\emptyset)$ form a strict chain with respect to inclusion:

$$\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_5)} \quad \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_4)} \quad \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)} \quad \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} \quad \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_1)}$$

Proof The inclusions follow easily from the relative definability results of section 5.5. In each case, we give examples to show that the inclusions are strict. Note that in each fragment the open extension of convex bisimilarity is compatible.

For $\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_1)}$ and $\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)}$ define M and N by:

$$\vdash M \stackrel{\text{def}}{=} \lambda x.\, \mathsf{let}\, y \Leftarrow x\, \mathsf{in}\, [[\underline{0}]] : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat}))$$
$$\vdash N \stackrel{\text{def}}{=} \lambda x.\, \mathsf{let}\, y \Leftarrow x\, \mathsf{in}\, [\mathsf{let}\, z \Leftarrow x\, \mathsf{in}\, [\underline{0}]] : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat}))$$

We prove $M \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_1)} N$ and $M \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} N$. For $M \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_1)} N$, it suffices to show that, for all $L \in \mathscr{L}_0(L_1)(\mathsf{P}_\perp(\mathsf{nat}))$, $ML \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_1)} NL$. Now $L_1 = ?\langle\underline{0}\rangle \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_1)} [\underline{0}]$, and so we obtain a program $L' \in \mathscr{L}_0(\emptyset)$ such that $L \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_1)} L'$ by replacing all occurrences of L_1 with $[\underline{0}]$ and using compatibility. Then $ML \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_1)} ML'$ and $NL \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_1)} NL'$. But L' is deterministic, and so, by examining the possible reductions of both programs and using compatibility, it can be shown that $ML' \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_1)} NL'$. Therefore $M \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_1)} N$. For $M \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} N$, we observe $ML_2 \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} NL_2$. By examining reductions and using compatibility, we can show that $ML_2 \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} \Omega \cup [[\underline{0}]]$ and $NL_2 \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} \Omega \cup [\Omega \cup [\underline{0}]]$, but $\Omega \cup [[\underline{0}]] \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} \Omega \cup [\Omega \cup [\underline{0}]]$. Therefore $M \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} N$.

For $\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)}$ and $\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)}$ define M and N by:

$$\vdash M \stackrel{\mathrm{def}}{=} \lambda x.\,\mathsf{let}\, y \Leftarrow x\,\mathsf{in}\,[\mathsf{plus}\,(y,y)] : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{nat})$$
$$\vdash N \stackrel{\mathrm{def}}{=} \lambda x.\,\mathsf{let}\, y \Leftarrow x\,\mathsf{in}\,\mathsf{let}\, z \Leftarrow x\,\mathsf{in}\,[\mathsf{plus}\,(y,z)] : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{nat})$$

We prove $M \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} N$ and $M \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)} N$. For $M \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} N$, it suffices to show that, for all $L \in \mathscr{L}_0(L_2)(\mathsf{P}_\perp(\mathsf{nat}))$, $M\,L \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} N\,L$. Clearly, $M\,L \Uparrow^{\mathrm{may}}$ if and only if $N\,L \Uparrow^{\mathrm{may}}$. Also, there must exist a term L' such that $L = L'[?\langle \Omega, \underline{0}\rangle / x]$ and:

$$x : \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat})) \vdash L' \in \mathscr{L}_0(\emptyset) : \mathsf{P}_\perp(\mathsf{nat})$$

If L may converge at all, by lemma 5.6.3(1), there is a unique natural number n such that, for all K, $L \Downarrow^{\mathrm{may}} K$ implies $K \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} [\underline{n}]$. By compatibility, it follows that if:

$$\mathsf{let}\, y \Leftarrow L\,\mathsf{in}\,[\mathsf{plus}\,(y,y)] \Downarrow^{\mathrm{may}} [\mathsf{plus}\,(N_1,N_1)]$$
$$\mathsf{let}\, y \Leftarrow L\,\mathsf{in}\,\mathsf{let}\, z \Leftarrow L\,\mathsf{in}\,[\mathsf{plus}\,(y,z)] \Downarrow^{\mathrm{may}} [\mathsf{plus}\,(N_2,N_3)]$$

Then $[\mathsf{plus}\,(N_1,N_1)] \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} [\mathsf{plus}\,(N_2,N_3)]$. Therefore $M \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_2)} N$. For $M \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)} N$, we observe $M\,L_3 \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)} N\,L_3$. By examining reductions and using compatibility, we have:

$$M\,L_3 \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)} \Omega \cup ?\langle \mathsf{plus}\,(\underline{0},\underline{0}), \mathsf{plus}\,(\underline{1},\underline{1})\rangle$$
$$N\,L_3 \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)} \Omega \cup ?\langle \mathsf{plus}\,(\underline{0},\underline{0}), \mathsf{plus}\,(\underline{0},\underline{1}), \mathsf{plus}\,(\underline{1},\underline{0}), \mathsf{plus}\,(\underline{1},\underline{1})\rangle$$

They are not related by convex bisimilarity, and therefore $M \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)} N$.

For $\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)}$ and $\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_4)}$ define M and N by:

$$\vdash M \stackrel{\mathrm{def}}{=} \lambda x.\,\mathsf{let}\, y \Leftarrow x\,\mathsf{in}\,[\underline{0}] : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{nat})$$
$$\vdash N \stackrel{\mathrm{def}}{=} \lambda x.\,\mathsf{let}\, y \Leftarrow x\,\mathsf{in}\,\mathsf{let}\, z \Leftarrow x\,\mathsf{in}\,\mathsf{if}\,\mathsf{eq}\,(y,z)\,\mathsf{then}\,[\underline{0}]\,\mathsf{else}\,\Omega : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{nat})$$

We prove $M \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)} N$ and $M \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_4)} N$. For $M \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)} N$, it suffices to show that, for all $L \in \mathscr{L}_0(L_3)(\mathsf{P}_\perp(\mathsf{nat}))$, $M\,L \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)} N\,L$. It is clear that both $M\,L$ and $N\,L$ may converge to $[\underline{0}]$ if and only if L may converge, and neither one can converge to another canonical program. Now $M\,L \Uparrow^{\mathrm{may}}$ if and only if $L \Uparrow^{\mathrm{may}}$. Also, $L \Uparrow^{\mathrm{may}}$ implies $N\,L \Uparrow^{\mathrm{may}}$. Thus we only have to show $N\,L \Uparrow^{\mathrm{may}}$ implies $L \Uparrow^{\mathrm{may}}$. The only case to consider is when the conditional reduces to Ω in the else branch because L may converge to programs representing two different numbers. By lemma 5.6.3(2), $L \Uparrow^{\mathrm{may}}$ and we are done. Therefore $M \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_3)} N$. For $M \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_4)} N$, we observe $M\,L_4 \not\simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_4)} N\,L_4$. By examining reductions, we have:

$$M\,L_4 \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_4)} [\underline{0}]$$
$$N\,L_4 \simeq_{\mathrm{CB}}^{\mathscr{L}_0(L_4)} \Omega \cup [\underline{0}]$$

They are not related by convex bisimilarity, and therefore $M \not\simeq^{\mathscr{L}_0(L_4)}_{\mathrm{CB}} N$.

For $\simeq^{\mathscr{L}_0(L_4)}_{\mathrm{CB}}$ and $\simeq^{\mathscr{L}_0(L_5)}_{\mathrm{CB}}$ define M and N by:

$$\vdash M \stackrel{\text{def}}{=} \lambda x.\,\mathsf{let}\, y \Leftarrow x\,\mathsf{in}\,[\underline{0}] : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{nat})$$
$$\vdash N \stackrel{\text{def}}{=} \lambda x.\,((\mathsf{ffix}\, f.\,\lambda y.\,\mathsf{let}\, z \Leftarrow x\,\mathsf{in}\,\mathsf{if}\,\mathsf{lt}\,(y,z)\,\mathsf{then}\, f\, z\,\mathsf{else}\,[\underline{0}])\,\underline{0}) : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{nat})$$

We prove $M \simeq^{\mathscr{L}_0(L_4)}_{\mathrm{CB}} N$ and $M \not\simeq^{\mathscr{L}_0(L_5)}_{\mathrm{CB}} N$. For $M \simeq^{\mathscr{L}_0(L_4)}_{\mathrm{CB}} N$, it suffices to show that, for all $L \in \mathscr{L}_0(L_4)(\mathsf{P}_\perp(\mathsf{nat}))$, $M\,L \simeq^{\mathscr{L}_0(L_4)}_{\mathrm{CB}} N\,L$. Now $M\,L \Uparrow^{\mathrm{may}}$ if and only if $L \Uparrow^{\mathrm{may}}$, $M\,L \Downarrow^{\mathrm{may}} [\underline{0}]$ if and only if there exists K such that $L \Downarrow^{\mathrm{may}} K$, and $M\,L$ cannot converge to any other program. If $L \Uparrow^{\mathrm{may}}$ then $N\,L \Uparrow^{\mathrm{may}}$, and if L cannot converge to any program, then neither can $N\,L$. If L can converge to at least one program, then $N\,L \Downarrow^{\mathrm{may}} [\underline{0}]$. This leaves the possibility that $N\,L \Uparrow^{\mathrm{may}}$ and $L \Downarrow^{\mathrm{must}}$. But then the true branch of the conditional must be chosen at each iteration within $N\,L$, and this is only possible if L can converge to a set of programs that represent an infinite set of natural numbers, because y must take values from a strictly increasing sequence of natural numbers. Lemma 3.4.8(2) rules this out because $L \Downarrow^{\mathrm{must}}$. So we have $N\,L \Uparrow^{\mathrm{may}}$ if and only if $L \Uparrow^{\mathrm{may}}$, $N\,L \Downarrow^{\mathrm{may}} [\underline{0}]$ if and only if there exists K such that $L \Downarrow^{\mathrm{may}} K$, and $N\,L$ cannot converge to any other program. Therefore $M \simeq^{\mathscr{L}_0(L_4)}_{\mathrm{CB}} N$. For $M \not\simeq^{\mathscr{L}_0(L_5)}_{\mathrm{CB}} N$, we observe $M\,L_5 \not\simeq^{\mathscr{L}_0(L_5)}_{\mathrm{CB}} N\,L_5$. The program $N\,L_5$ may diverge. To see this define N' by:

$$N' \stackrel{\text{def}}{=} \mathsf{if}\,\mathsf{lt}\,(y,z)\,\mathsf{then}\, f\, z\,\mathsf{else}\,[\underline{0}]$$

Then, for all $n \in \omega$:

$$\begin{array}{ll}
 & (\mathsf{ffix}\, f.\,\lambda y.\,\mathsf{let}\, z \Leftarrow ?\underline{\omega}\,\mathsf{in}\, N')\,\underline{n} \\
\rightarrowtriangle^{+} & (\lambda y.\,\mathsf{let}\, z \Leftarrow ?\underline{\omega}\,\mathsf{in}\, N'[\mathsf{ffix}\, f.\,\lambda y.\,\mathsf{let}\, z \Leftarrow ?\underline{\omega}\,\mathsf{in}\, N'/f])\,\underline{n} \\
\rightarrowtriangle & \mathsf{let}\, z \Leftarrow ?\underline{\omega}\,\mathsf{in}\, N'[\mathsf{ffix}\, f.\,\lambda y.\,\mathsf{let}\, z \Leftarrow ?\underline{\omega}\,\mathsf{in}\, N'/f][\underline{n}/y] \\
\rightarrowtriangle & \mathsf{let}\, z \Leftarrow [\underline{n+1}]\,\mathsf{in}\, N'[\mathsf{ffix}\, f.\,\lambda y.\,\mathsf{let}\, z \Leftarrow ?\underline{\omega}\,\mathsf{in}\, N'/f][\underline{n}/y] \\
\rightarrowtriangle & \mathsf{if}\,\mathsf{lt}\,(\underline{n},\underline{n+1})\,\mathsf{then}\,(\mathsf{ffix}\, f.\,\lambda y.\,\mathsf{let}\, z \Leftarrow ?\underline{\omega}\,\mathsf{in}\, N')\,\underline{n+1}\,\mathsf{else}\,[\underline{0}] \\
\rightarrowtriangle & (\mathsf{ffix}\, f.\,\lambda y.\,\mathsf{let}\, z \Leftarrow ?\underline{\omega}\,\mathsf{in}\, N')\,\underline{n+1}
\end{array}$$

By examining other reductions, we have:

$$M\,L_5 \simeq^{\mathscr{L}_0(L_5)}_{\mathrm{CB}} [\underline{0}]$$
$$N\,L_5 \simeq^{\mathscr{L}_0(L_5)}_{\mathrm{CB}} \Omega \cup [\underline{0}]$$

They are not related by convex bisimilarity, and therefore $M \not\simeq^{\mathscr{L}_0(L_5)}_{\mathrm{CB}} N$. □

Perhaps more surprisingly, convex bisimilarity for language fragments more expressive than $\mathscr{L}_0(?\underline{\omega})$ can be finer than $\simeq^{\mathscr{L}_0(?\underline{\omega})}_{\mathrm{CB}}$. The example in proposition 5.6.5 was suggested by Alan Jeffrey.

Proposition 5.6.5 If $A \subseteq_{\mathrm{ne}} \omega$ is not recursively enumerable, then the restriction of $\simeq^{\mathscr{L}_0(?\underline{A},?\underline{\omega})}_{\mathrm{CB}}$ to $\mathscr{L}_0(?\underline{\omega})$ is strictly finer than $\simeq^{\mathscr{L}_0(?\underline{\omega})}_{\mathrm{CB}}$.

Proof There are countably many terms in $\mathscr{L}_0(?\underline{\omega})$ (up to α-equivalence) and we assume an enumeration $\langle M_n \mid n < \kappa\rangle$ of programs in $\mathscr{L}_0(?\underline{\omega})(\mathsf{P}_\perp(\mathsf{nat}))$ (possibly containing repetitions). Define M and N by:

$$\vdash M \stackrel{\text{def}}{=} \lambda x.\, ?\langle M_n \mid n < \kappa\rangle : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat}))$$
$$\vdash N \stackrel{\text{def}}{=} \lambda x.\, [x] \cup ?\langle M_n \mid n < \kappa\rangle : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat}))$$

For any $L \in \mathscr{L}_0(?\underline{\omega})(\mathsf{P}_\perp(\mathsf{nat}))$, there exists $m \in \omega$ such that $L = M_m$. Therefore:

$$M\,L \simeq_{\text{CB}}^{\mathscr{L}_0(?\underline{\omega})} ?\langle M_n \mid n < \kappa\rangle \simeq_{\text{CB}}^{\mathscr{L}_0(?\underline{\omega})} [M_m] \cup ?\langle M_n \mid n < \kappa\rangle \simeq_{\text{CB}}^{\mathscr{L}_0(?\underline{\omega})} N\,L$$

But, M and N can be distinguished by $?\underline{A}$ because, by lemma 5.5.6(4), there is no $m \in \omega$ such that $M_m \simeq_{\text{CB}}^{\mathscr{L}_0(?\underline{\omega})} ?\underline{A}$. Thus:

$$M\,(?\underline{A}) \simeq_{\text{CB}}^{\mathscr{L}_0(?\underline{A},?\underline{\omega})} ?\langle M_n \mid n < \kappa\rangle \not\simeq_{\text{CB}}^{\mathscr{L}_0(?\underline{A},?\underline{\omega})} [?\underline{A}] \cup ?\langle M_n \mid n < \kappa\rangle \simeq_{\text{CB}}^{\mathscr{L}_0(?\underline{A},?\underline{\omega})} N\,(?\underline{A})$$

This does not complete the proof because M and N are not in $\mathscr{L}_0(?\underline{\omega})$. However, we can construct programs in $\mathscr{L}_0(?\underline{\omega})$ that are convex bisimilar to M and N. To see this, note that not only is $\mathscr{L}(?\underline{\omega})$ countable, but type checking and the reduction relation are recursive on a suitable encoding of terms as natural numbers. We denote the encoding of a term $L \in \mathscr{L}(?\underline{\omega})$ by $\widetilde{L} \in \omega$. Now there exists a program interp such that:

$$\vdash \mathsf{interp} \in \mathscr{L}(?\underline{\omega}) : \mathsf{nat} \to \mathsf{P}_\perp(\mathsf{nat})$$

and:

$$\mathsf{interp}\,\underline{m} \simeq_{\text{CB}}^{\mathscr{L}_0(?\underline{\omega})} \begin{cases} L & \text{if } \exists L \in \mathscr{L}_0(?\underline{\omega})(\mathsf{P}_\perp(\mathsf{nat})).\, m = \widetilde{L} \\ [\underline{0}] & \text{otherwise} \end{cases}$$

If $?\langle M_n \mid n < \kappa\rangle \Downarrow^{\text{may}} [M_m]$ then:

$$\begin{array}{ll} & \mathsf{let}\, y \Leftarrow ?\underline{\omega}\, \mathsf{in}\, [\mathsf{interp}\, y] \\ \rightarrow & \mathsf{let}\, y \Leftarrow [\underline{\widetilde{M_m}}]\, \mathsf{in}\, [\mathsf{interp}\, y] \\ \rightarrow & [\mathsf{interp}\, \underline{\widetilde{M_m}}] \end{array}$$

And it follows that:

$$?\langle M_n \mid n < \kappa\rangle \simeq_{\text{CB}}^{\mathscr{L}_0(?\underline{\omega})} \mathsf{let}\, y \Leftarrow ?\underline{\omega}\, \mathsf{in}\, [\mathsf{interp}\, y]$$

The argument above holds when we redefine M and N by:

$$\vdash M \stackrel{\text{def}}{=} \lambda x.\, \mathsf{let}\, y \Leftarrow ?\underline{\omega}\, \mathsf{in}\, [\mathsf{interp}\, y] : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat}))$$
$$\vdash N \stackrel{\text{def}}{=} \lambda x.\, [x] \cup \mathsf{let}\, y \Leftarrow ?\underline{\omega}\, \mathsf{in}\, [\mathsf{interp}\, y] : \mathsf{P}_\perp(\mathsf{nat}) \to \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat}))$$

Therefore $M \simeq_{\text{CB}}^{\mathscr{L}_0(?\underline{\omega})} N$ and $M \not\simeq_{\text{CB}}^{\mathscr{L}_0(?\underline{A},?\underline{\omega})} N$, where $M, N \in \mathscr{L}_0(?\underline{\omega})$. □

5.7 Fixed-Points

In this section we prove the Scott induction principle for the lower, upper, and convex variants of similarity, i.e., fixed-point terms are least fixed-points with respect to those relations.

With deterministic λ-calculi it is often possible to define finite approximations to a fixed-point term and prove that their least upper bound is equivalent to the fixed-point (see [MST96, Pit97, San97]). The situation with non-deterministic λ-calculi is more complex because ω-continuity usually fails and the variants of mutual similarity and bisimilarity do not coincide. There are a number of approaches to resolving the lack of ω-continuity in a denotational setting, including lower, upper, and convex powerdomain models [Plo76, Plo83, Gun92] and categorical powerdomain models [Leh76, Abr83, PR88, Rus90]. These approaches are not entirely satisfactory because they make unwanted identifications [Plo83, Ong93, MW95] or do not make necessary identifications such as capturing the idempotency of binary erratic choice. In addition, the results of section 5.6 suggest that a range of models will be required to model languages with different forms of erratic non-determinism (with respect to relative definability).

Lassen [Las98b] proves positive and negative results about unwinding, continuity, and Scott induction for contextual preorders upon λ-calculi with finite and countable erratic non-determinism, as well as a continuity result for upper similarity in the presence of finite non-determinism. He also gives a number of examples, one of which demonstrates the failure of ω-continuity for lower similarity. This is rephrased for $\mathscr{L}$ in example 5.7.1.

Example 5.7.1 Define the term M by:

$$x : \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat})) \vdash M \stackrel{\text{def}}{=} [[\underline{0}] \cup (\mathsf{let}\, y \Leftarrow x\, \mathsf{in}\, \mathsf{let}\, z \Leftarrow y\, \mathsf{in}\, [\mathsf{plus}\,(z, \underline{1})])] : \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat}))$$

It can be shown that:

$$\vdash \mathsf{fix}\, x.\, M \simeq^{\mathscr{M}_0}_{\mathrm{CB}} [\Omega \cup ?\underline{\omega}] : \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat}))$$

As an aside, varying the placement of the unit term constructors in M changes the convex bisimilarity equivalence class.

Now, for $n \in \omega$, define the nth unwinding $\mathsf{fix}^{(n)}\, x.\, M$ of the fixed-point program $\mathsf{fix}\, x.\, M$ by:

$$\begin{aligned} &\vdash \mathsf{fix}^{(0)}\, x.\, M \stackrel{\text{def}}{=} \Omega : \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat})) \\ &\vdash \mathsf{fix}^{(n+1)}\, x.\, M \stackrel{\text{def}}{=} M[\mathsf{fix}^{(n)}\, x.\, M/x] : \mathsf{P}_\perp(\mathsf{P}_\perp(\mathsf{nat})) \end{aligned}$$

The finite unwindings of $\mathsf{fix}\, x.\, M$ are related by convex bisimilarity to simple programs:

$$\begin{aligned} \mathsf{fix}^{(0)}\, x.\, M &\simeq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega \\ \mathsf{fix}^{(1)}\, x.\, M &\simeq^{\mathscr{M}_0}_{\mathrm{CB}} [\Omega \cup ?\langle \underline{0} \rangle] \\ \mathsf{fix}^{(2)}\, x.\, M &\simeq^{\mathscr{M}_0}_{\mathrm{CB}} [\Omega \cup ?\langle \underline{0}, \underline{1} \rangle] \\ &\vdots \\ \mathsf{fix}^{(n+1)}\, x.\, M &\simeq^{\mathscr{M}_0}_{\mathrm{CB}} [\Omega \cup ?\langle \underline{0}, \underline{1}, \ldots, \underline{n} \rangle] \end{aligned}$$

The program $N \stackrel{\text{def}}{=} \Omega \cup ?\langle \Omega \cup ?\langle \underline{m} \mid m \leq n \rangle \mid n < \omega \rangle$ is a least upper bound of the finite unwindings with respect to the preorder lower similarity, but it is not a fixed-point of M with respect to mutual lower similarity because $M[N/x] \not\lesssim^{\mathscr{M}_0}_{\text{LS}} N$. However, the additional unwinding does coincide with the fixed-point:

$$M[N/x] \simeq^{\mathscr{M}_0}_{\text{CB}} \mathsf{fix} x. M \simeq^{\mathscr{M}_0}_{\text{CB}} [\Omega \cup ?\underline{\omega}]$$

If the unwinding for the limit ordinal ω is defined by $\mathsf{fix}^{(\omega)} x. M \stackrel{\text{def}}{=} N$, and the previous definition is extended for successor ordinals, then the closure ordinal of the function that takes a term and substitutes it into M for x is $\omega + 1 = Succ(\omega)$. In general, least upper bounds with respect to lower similarity are definable in $\mathscr{L}$ only if the set of terms is countable, and so unwindings cannot be defined for uncountable limit ordinals. Note also that N is not a least upper bound with respect to upper or convex similarity.

The usual technique for proving a syntactic formulation of Scott induction relies on ω-continuity, but this does not hold in general for erratic non-determinism. In the remainder of this section, we develop a novel technique that does not depend upon ω-continuity, and prove the Scott induction principle for the lower, upper, and convex variants of similarity upon $\mathscr{L}$ and all of the language fragments. However, we first observe that Scott induction does not hold for refinement similarity.

Example 5.7.2 If $M \stackrel{\text{def}}{=} x$ and $N \stackrel{\text{def}}{=} [\star]$, then $\vdash M[N/x] \lesssim^{\mathscr{M}_0}_{\text{RS}} N : \mathsf{P}_\perp(\sigma)$, but:

$$\mathsf{fix} x. M \simeq^{\mathscr{M}_0}_{\text{CB}} \Omega \not\lesssim^{\mathscr{M}_0}_{\text{RS}} [\star] \simeq^{\mathscr{M}_0}_{\text{CB}} N$$

The components of the Scott induction result make use of a form of compatibility result where related terms are also related when placed in the same reduction context.

Lemma 5.7.3 Consider a compatible relation $\mathcal{R} \in Rel(\mathcal{E})$ and terms L, M, N such that $L \not\downarrow x$ and:

$$\Gamma, x : \sigma \vdash L \in \mathscr{L}(E) : \tau$$
$$\Gamma, x : \sigma \vdash \langle M, N \rangle \in \mathcal{R} : \sigma$$

Then $\Gamma, x : \sigma \vdash \langle L[x \mapsto M], L[x \mapsto N] \rangle \in \mathcal{R} : \tau$

Proof Reflexivity of $\mathcal{R}$ follows from compatibility by induction and lemma 5.3.8(2). The result follows by induction on the derivation of $L \not\downarrow x$. The coproduct and product cases require the fragment $\mathscr{L}(\mathcal{E})$ to be closed under blocked substitution. $\square$

Lemma 5.7.4 is more specific, and is used to replace a substituted term only at the blocked occurrence of a variable.

Lemma 5.7.4 Consider a compatible relation $\mathcal{R} \in Rel(\mathcal{E})$ and terms L, M, N such that $L \not\downarrow x$ and:

$$\mathcal{R}[Id(\mathscr{L}(\mathcal{E}))] \subseteq \mathcal{R}$$
$$\Gamma, x : \sigma \vdash L \in \mathscr{L}(\mathcal{E}) : \tau$$
$$\Gamma, x : \sigma \vdash M \in \mathscr{L}(\mathcal{E}) : \sigma$$
$$\Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \sigma$$
$$\Gamma \vdash \langle M[N/x], N \rangle \in \mathcal{R} : \sigma$$

Then $\Gamma \vdash \langle L[x \mapsto M][N/x], L[N/x] \rangle \in \mathcal{R} : \tau$.

Proof By lemma 5.7.3:

$$\Gamma, x : \sigma \vdash \langle L[x \mapsto M[N/x]], L[x \mapsto N] \rangle \in \mathcal{R} : \tau$$

With $\Gamma \vdash \langle N, N \rangle \in Id(\mathscr{L}(\mathcal{E})) : \sigma$ and $\mathcal{R}[Id(\mathscr{L}(\mathcal{E}))] \subseteq \mathcal{R}$, this implies:

$$\Gamma \vdash \langle L[x \mapsto M[N/x]][N/x], L[x \mapsto N][N/x] \rangle \in \mathcal{R} : \tau$$

By lemma 3.4.11(1,2):

$$L[x \mapsto M][N/x] = L[x \mapsto M[N/x]][N/x]$$
$$L[N/x] = L[x \mapsto N][N/x]$$

Therefore $\Gamma \vdash \langle L[x \mapsto M][N/x], L[N/x] \rangle \in \mathcal{R} : \tau$ □

We now make the following assumptions and definitions until proposition 5.7.8 (inclusively):

1. There is a preorder $\mathcal{R} \in Rel_0(\mathcal{E})$ such that the open extension of $\mathcal{R}$ is compatible:

   $$Cmp(\mathcal{E}, Opn(\mathcal{E}, \mathcal{R})) \subseteq Opn(\mathcal{E}, \mathcal{R})$$

   By lemma 5.3.4(2):

   $$Opn(\mathcal{E}, \mathcal{R})[Id(\mathscr{L}(\mathcal{E}))] \subseteq Opn(\mathcal{E}, \mathcal{R})$$

   The relation $\mathcal{R}$ is intended to be the lower, upper, or convex variant of similarity upon $\mathscr{L}_0(\mathcal{E})$.

2. There is a term M and a program N such that:

$$x : \mathsf{P}_\perp(\sigma) \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\sigma)$$
$$\vdash N \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\sigma)$$
$$\vdash \langle M[N/x], N\rangle \in \mathcal{R} : \mathsf{P}_\perp(\sigma)$$

   With additional constraints on $\mathcal{R}$ we prove in theorem 5.7.9 that:

$$\vdash \langle \mathsf{fix}\, x.\, M, N\rangle \in \mathcal{R} : \mathsf{P}_\perp(\sigma)$$

3. Define the relation $\mathcal{S} \in Rel_0(\mathcal{E})$ by:

$$\mathcal{S} \stackrel{\text{def}}{=} \{\langle D[\mathsf{fix}\, x.\, M/x], D[N/x]\rangle \mid x{:}\mathsf{P}_\perp(\sigma) \vdash D \in \mathscr{L}(\mathcal{E}) : \tau\}$$

   The relation $\mathcal{S}$ is used for a coinductive proof. The meta-variable D ranges over terms that function as contexts in the following proofs. However, we only substitute programs for x. Consequently, contextual substitution with variable capture is unnecessary, and we may restrict ourselves to ordinary substitution.

The relation $\mathcal{S}$ cannot be defined by relational substitution, but does satisfy a property similar to that of lemma 5.3.6(5).

Lemma 5.7.5

$$\mathcal{S} \subseteq \{\langle \mathsf{fix}\, x.\, M, N\rangle\} \cup Cls(Cmp(\mathcal{E}, Opn(\mathcal{E}, \mathcal{S})))$$

Proof By a case analysis on $x : \mathsf{P}_\perp(\sigma) \vdash D \in \mathscr{L}(\mathcal{E}) : \tau$. If $D = x$ then:

$$\langle D[\mathsf{fix}\, x.\, M/x], D[N/x]\rangle = \langle \mathsf{fix}\, x.\, M, N\rangle$$

and we are done. Otherwise D is not a variable and we have to show:

$$\vdash \langle D[\mathsf{fix}\, x.\, M/x], D[N/x]\rangle \in Cmp(\mathcal{E}, Opn(\mathcal{E}, \mathcal{S})) : \tau$$

In each case, substituting $\mathsf{fix}\, x.\, M$ and N for x into the immediate subterms of D results in terms that are related by $Opn(\mathcal{E}, \mathcal{S})$ (for variable binding term constructors we use the fact that $\mathsf{fix}\, x.\, M$ and N are closed to commute substitutions). Finally, $D[\mathsf{fix}\, x.\, M/x], D[N/x] \in \mathscr{L}(\mathcal{E})$ because fragments are closed under substitution, and therefore:

$$\vdash \langle D[\mathsf{fix}\, x.\, M/x], D[N/x]\rangle \in Cmp(\mathcal{E}, Opn(\mathcal{E}, \mathcal{S})) : \tau$$

$\square$

We now use the preceding lemmas to prove propositions dealing with the behaviour of terms resulting from a substitution of $\mathsf{fix}x.M$ and N into the same term D. In each proposition, the idea is to use the fact that a reduction of $D[\mathsf{fix}x.M/x]$ or $D[N/x]$ does not involve either substitution (so there is a term E such that $D \rightharpoonup E$) or $D \not\downarrow x$. For the former case, $D[\mathsf{fix}x.M/x] \rightharpoonup E[\mathsf{fix}x.M/x]$ and $D[N/x] \rightharpoonup E[N/x]$. In the latter case, $D[\mathsf{fix}x.M/x] \rightharpoonup_{\mathrm{det}} D[x \mapsto M][\mathsf{fix}x.M/x]$ and we have the relationship

$$\vdash \langle D[x \mapsto M][N/x], D[N/x] \rangle \in \mathcal{R} : \tau$$

And then the proofs continue working with the terms resulting from substituting $\mathsf{fix}x.M$ and N into $D[x \mapsto M]$. Proposition 5.7.6 handles the case for the function $\langle \cdot \rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$. Propositions 5.7.7 and 5.7.8 establish the two clauses in the alternative formulation of $\langle \cdot \rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$ on page 91.

Proposition 5.7.6 Suppose $\mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$ and there is a term D and a program K_1 such that $x : \mathsf{P}_\perp(\sigma) \vdash D \in \mathscr{L}(\mathcal{E}) : \tau$ and $D[\mathsf{fix}x.M/x] \Downarrow^{\mathrm{may}} K_1$. Then there exists a program K_2 such that $D[N/x] \Downarrow^{\mathrm{may}} K_2$ and $\vdash \langle K_1, K_2 \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{S}; \mathcal{R})) : \tau$.

Proof By induction on the length of the reduction sequence $D[\mathsf{fix}x.M/x] \rightharpoonup^* K_1$. For the base case, $D[\mathsf{fix}x.M/x] = K_1$. By lemma 3.2.2, D is canonical because $\mathsf{fix}x.M$ is not canonical. Hence $D[N/x]$ is also canonical. In addition, by lemma 5.7.5:

$$\vdash \langle D[\mathsf{fix}x.M/x], D[N/x] \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{S})) : \tau$$

Now $\mathcal{R}$ is a preorder, hence reflexive, so $\mathcal{S} \subseteq \mathcal{S}; \mathcal{R}$, and the open extension and compatible refinement are monotone, so:

$$\mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{S})) \subseteq \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{S}; \mathcal{R}))$$

Therefore:

$$\vdash \langle D[\mathsf{fix}x.M/x], D[N/x] \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{S}; \mathcal{R})) : \tau$$

For the inductive step, suppose that $D[\mathsf{fix}x.M/x] \rightharpoonup L \rightharpoonup^* K_1$. By lemma 3.4.10(2), either $D \not\downarrow x$ or there exists a term E such that $D \rightharpoonup E$ and $L = E[\mathsf{fix}x.M/x]$. Suppose $D \not\downarrow x$. By lemma 3.4.11(4), $D[\mathsf{fix}x.M/x]$ has precisely one reduction:

$$D[\mathsf{fix}x.M/x] \rightharpoonup_{\mathrm{det}} D[x \mapsto M][\mathsf{fix}x.M/x]$$

So $L = D[x \mapsto M][\mathsf{fix}x.M/x]$. Applying the induction hypothesis to (note that language fragments are closed under blocked substitution):

$$D[x \mapsto M][\mathsf{fix}x.M/x] \Downarrow^{\mathrm{may}} K_1$$

yields a program K_2 such that $D[x \mapsto M][N/x] \Downarrow^{\mathrm{may}} K_2$ and:

$$\vdash \langle K_1, K_2 \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{S}; \mathcal{R})) : \tau$$

By lemma 5.7.4:

$$\vdash \langle D[x \mapsto M][N/x], D[N/x]\rangle \in Opn(\mathcal{E},\mathcal{R}) : \tau$$

So $\langle D[x \mapsto M][N/x], D[N/x]\rangle \in \mathcal{R} \subseteq \langle \mathcal{R}\rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$. By lemma 5.3.9(1), there exists a program K_3 such that $D[N/x] \Downarrow^{\mathrm{may}} K_3$ and:

$$\vdash \langle K_2, K_3\rangle \in Cmp(\mathcal{E}, Opn(\mathcal{E},\mathcal{R})) : \tau$$

The relation $\mathcal{R}$ is transitive, so $\mathcal{S};\mathcal{R};\mathcal{R} \subseteq \mathcal{S};\mathcal{R}$. Using lemma 5.3.4(3) and 5.3.6(4) we have:

$$Cmp(\mathcal{E}, Opn(\mathcal{E},\mathcal{S};\mathcal{R})); Cmp(\mathcal{E}, Opn(\mathcal{E},\mathcal{R})) \subseteq Cmp(\mathcal{E}, Opn(\mathcal{E},\mathcal{S};\mathcal{R}))$$

Therefore:

$$\vdash \langle K_1, K_3\rangle \in Cmp(\mathcal{E}, Opn(\mathcal{E},\mathcal{S};\mathcal{R})) : \tau$$

This completes the case when $D \nmid x$. Now consider the case when $D \twoheadrightarrow E$ and $L = E[\mathsf{fix}x.M/x]$. Applying the induction hypothesis to $E[\mathsf{fix}x.M/x] \Downarrow^{\mathrm{may}} K_1$, gives us a program K_2 such that $E[N/x] \Downarrow^{\mathrm{may}} K_2$ and:

$$\vdash \langle K_1, K_2\rangle \in Cmp(\mathcal{E}, Opn(\mathcal{E},\mathcal{S};\mathcal{R})) : \tau$$

Using lemma 3.4.10(1), $D[N/x] \twoheadrightarrow E[N/x]$, so $D[N/x] \Downarrow^{\mathrm{may}} K_2$, and we are done. $\square$

Proposition 5.7.7 If $\mathcal{R} \subseteq \langle \mathcal{R}\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$ and there is a term D such that $x : \mathsf{P}_\perp(\sigma) \vdash D \in \mathscr{L}(\mathcal{E}) : \tau$ and $D[N/x] \Uparrow^{\mathrm{may}}$. Then $D[\mathsf{fix}x.M/x] \Uparrow^{\mathrm{may}}$.

Proof By coinduction it suffices to show that there exists a term E such that $D[\mathsf{fix}x.M/x] \twoheadrightarrow E[\mathsf{fix}x.M/x]$ where $x : \mathsf{P}_\perp(\sigma) \vdash E \in \mathscr{L}(\mathcal{E}) : \tau$ and $E[N/x] \Uparrow^{\mathrm{may}}$. By lemma 3.4.10(2), $D[N/x] \Uparrow^{\mathrm{may}}$ implies either $D \nmid x$ or there exists a term E such that $D \twoheadrightarrow E$ and $E[N/x] \Uparrow^{\mathrm{may}}$. Suppose $D \nmid x$. By lemma 5.7.4:

$$\vdash \langle D[x \mapsto M][N/x], D[N/x]\rangle \in Opn(\mathcal{E},\mathcal{R}) : \tau$$

By using $\langle D[x \mapsto M][N/x], D[N/x]\rangle \in \mathcal{R} \subseteq \langle \mathcal{R}\rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$ and $D[N/x] \Uparrow^{\mathrm{may}}$ we may deduce that $D[x \mapsto M][N/x] \Uparrow^{\mathrm{may}}$. By lemma 3.4.11(4), $D[\mathsf{fix}x.M/x]$ has one reduction:

$$D[\mathsf{fix}x.M/x] \twoheadrightarrow_{\mathrm{det}} D[x \mapsto M][\mathsf{fix}x.M/x]$$

Therefore, we may take $E = D[x \mapsto M]$, and this concludes the case when $D \nmid x$. Now consider the case when $D \twoheadrightarrow E$ and $E[N/x] \Uparrow^{\mathrm{may}}$. By lemma 3.4.10(1), $D[\mathsf{fix}x.M/x] \twoheadrightarrow E[\mathsf{fix}x.M/x]$, and we are done. $\square$

Proposition 5.7.8 Suppose $\mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\mathrm{US}}^{\mathcal{L}_0(\mathcal{E})}$ and there is a term D and a program K_2 such that $x : \mathsf{P}_\perp(\sigma) \vdash D \in \mathcal{L}(\mathcal{E}) : \tau$ and $D[N/x] \Downarrow^{\mathrm{may}} K_2$. Then $D[\mathsf{fix}x.M/x] \Uparrow^{\mathrm{may}}$ or there exists a program K_1 such that $D[\mathsf{fix}x.M/x] \Downarrow^{\mathrm{may}} K_1$ and:

$$\vdash \langle K_1, K_2 \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{S}; \mathcal{R})) : \tau$$

Proof Suppose that D is canonical, so $D \neq x$ and both $D[\mathsf{fix}x.M/x]$ and $D[N/x]$ are canonical. By lemma 5.7.5:

$$\vdash \langle D[\mathsf{fix}x.M/x], D[N/x] \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{S})) : \tau$$

Now suppose that D is not canonical. We claim that $D[\mathsf{fix}x.M/x] \Uparrow^{\mathrm{may}}$ or there exists a term E and a program K_3 such that $D[\mathsf{fix}x.M/x] \rightarrowtriangle E[\mathsf{fix}x.M/x]$, $E[N/x] \Downarrow^{\mathrm{may}} K_3$, and:

$$\vdash \langle K_3, K_2 \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{R})) : \tau$$

Either $D = x$ and N is canonical, so $D \not\searrow x$, or $D[N/x]$ is not canonical and there exists L such that $D[N/x] \rightarrowtriangle L \rightarrowtriangle^* K_2$. In the latter case, by lemma 3.4.10(2), either $D \not\searrow x$ or there exists a term E such that $D \rightarrowtriangle E$ and $L = E[N/x]$. Suppose $D \not\searrow x$. By lemma 5.7.4:

$$\vdash \langle D[x \mapsto M][N/x], D[N/x] \rangle \in \mathit{Opn}(\mathcal{E}, \mathcal{R}) : \tau$$

Using $\langle D[x \mapsto M][N/x], D[N/x] \rangle \in \mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\mathrm{US}}^{\mathcal{L}_0(\mathcal{E})}$ and $D[N/x] \Downarrow^{\mathrm{may}} K_2$, by lemma 5.3.9(2), we have that $D[x \mapsto M][N/x] \Uparrow^{\mathrm{may}}$ or there exists K_3 such that $D[x \mapsto M][N/x] \Downarrow^{\mathrm{may}} K_3$ and:

$$\vdash \langle K_3, K_2 \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{R})) : \tau$$

If $D[x \mapsto M][N/x] \Uparrow^{\mathrm{may}}$ then proposition 5.7.7 implies that $D[x \mapsto M][\mathsf{fix}x.M/x] \Uparrow^{\mathrm{may}}$. Also, by lemma 3.4.11(4):

$$D[\mathsf{fix}x.M/x] \rightarrowtriangle_{\mathrm{det}} D[x \mapsto M][\mathsf{fix}x.M/x]$$

Hence $D[x \mapsto M][N/x] \Uparrow^{\mathrm{may}}$ implies $D[\mathsf{fix}x.M/x] \Uparrow^{\mathrm{may}}$. Otherwise, define $E = D[x \mapsto M]$, so $D[\mathsf{fix}x.M/x] \rightarrowtriangle_{\mathrm{det}} E[\mathsf{fix}x.M/x]$ and $E[N/x] \Downarrow^{\mathrm{may}} K_3$. This establishes the claim when $D \not\searrow x$. Now suppose there exists a term E such that $D \rightarrowtriangle E$ and $E[N/x] \Downarrow^{\mathrm{may}} K_2$. By lemma 3.4.10(1), $D[\mathsf{fix}x.M/x] \rightarrowtriangle E[\mathsf{fix}x.M/x]$. Therefore we may take $K_3 = K_2$, because:

$$\vdash \langle K_2, K_2 \rangle \in \mathit{Cmp}(\mathcal{E}, \mathit{Opn}(\mathcal{E}, \mathcal{R})) : \tau$$

We can now deduce the result. By iterating the above argument, we obtain a sequence of reductions:

$$D[\mathsf{fix}x.M/x] \rightarrowtriangle E_1[\mathsf{fix}x.M/x] \rightarrowtriangle E_2[\mathsf{fix}x.M/x] \rightarrowtriangle \ldots$$

where, for $i \geq 1$, $E_i[N/x] \Downarrow^{\mathrm{may}} K_{i+2}$ and:

$$\vdash \langle K_{i+2}, K_{i+1} \rangle \in Cmp(\mathcal{E}, Opn(\mathcal{E}, \mathcal{R})) : \tau$$

The sequence may terminate when $E_n[\mathsf{fix}x.M/x] \Uparrow^{\mathrm{may}}$, in which case $D[\mathsf{fix}x.M/x] \Uparrow^{\mathrm{may}}$, or a term E_n is canonical. Otherwise the sequence is infinite, so $D[\mathsf{fix}x.M/x] \Uparrow^{\mathrm{may}}$. Thus, we need only consider the case when E_n is canonical. We have:

$$D[\mathsf{fix}x.M/x] \Downarrow^{\mathrm{may}} E_n[\mathsf{fix}x.M/x]$$
$$\vdash \langle E_n[\mathsf{fix}x.M/x], E_n[N/x] \rangle \in Cmp(\mathcal{E}, Opn(\mathcal{E}, \mathcal{S})) : \tau$$
$$E_n[N/x] = K_{n+2}$$
$$\vdash \langle K_{n+2}, K_2 \rangle \in Cmp(\mathcal{E}, Opn(\mathcal{E}, \mathcal{R})) : \tau$$

Therefore:

$$\vdash \langle E_n[\mathsf{fix}x.M/x], K_2 \rangle \in Cmp(\mathcal{E}, Opn(\mathcal{E}, \mathcal{S}; \mathcal{R})) : \tau$$

We may set $K_1 = E_n[\mathsf{fix}x.M/x]$, and we are done. □

The Scott induction result for the variants of lower, upper, and convex variants of similarity upon every language fragment can be established using propositions 5.7.6, 5.7.7, and 5.7.8. This shows that Scott induction is a robust principle that requires no restrictions upon the form of erratic non-determinism.

Theorem 5.7.9 Suppose that $\mathcal{R} \in Rel_0(\mathcal{E})$ is lower similarity $\lesssim_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$, upper similarity $\lesssim_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$, or convex similarity $\lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}$, and there are terms M and N such that:

$$\Gamma, x : \mathsf{P}_\perp(\sigma) \vdash M \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\sigma)$$
$$\Gamma \vdash N \in \mathscr{L}(\mathcal{E}) : \mathsf{P}_\perp(\sigma)$$
$$\Gamma \vdash \langle M[N/x], N \rangle \in Opn(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)$$

Then $\Gamma \vdash \langle \mathsf{fix}x.M, N \rangle \in Opn(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)$.

Proof Suppose that Γ is the empty environment. Then the assumptions on page 171 apply to $\mathcal{R}$, M, and N for propositions 5.7.6, 5.7.7, and 5.7.8. It suffices to prove $\mathcal{S} \subseteq \mathcal{R}$. By the "up to" result of lemma 2.3.10, this follows from:

1. $\mathcal{S} \subseteq \langle \mathcal{S}; \mathcal{R} \rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$ when $\mathcal{R} = \lesssim_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})}$.
2. $\mathcal{S} \subseteq \langle \mathcal{S}; \mathcal{R} \rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$ when $\mathcal{R} = \lesssim_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$.
3. $\mathcal{S} \subseteq \langle \mathcal{S}; \mathcal{R} \rangle_{\mathrm{LS}}^{\mathscr{L}_0(\mathcal{E})} \cap \langle \mathcal{S}; \mathcal{R} \rangle_{\mathrm{US}}^{\mathscr{L}_0(\mathcal{E})}$ when $\mathcal{R} = \lesssim_{\mathrm{CS}}^{\mathscr{L}_0(\mathcal{E})}$.

Using lemma 5.3.9, these inclusions follow immediately from propositions 5.7.6, 5.7.7, and 5.7.8. This completes the case for the empty environment.

Now consider $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ and programs $L_i \in \mathscr{L}_0(\mathcal{E})(\sigma_i)$, for all $1 \leq i \leq n$. If $\vec{L} = L_1, \ldots, L_n$, we have to show:

$$\vdash \langle (\mathsf{fix}\, x. M)[\vec{L}/\vec{x}], N[\vec{L}/\vec{x}] \rangle \in \mathcal{R} : \mathsf{P}_\perp(\sigma)$$

Now $\Gamma \vdash \langle M[N/x], N \rangle \in Opn(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)$ implies:

$$\vdash \langle M[N/x][\vec{L}/\vec{x}], N[\vec{L}/\vec{x}] \rangle \in \mathcal{R} : \mathsf{P}_\perp(\sigma)$$

By lemma 3.3.4:

$$M[N/x][\vec{L}/\vec{x}] = M[\vec{L}/\vec{x}][N[\vec{L}/\vec{x}]/x]$$

And so:

$$\vdash \langle M[\vec{L}/\vec{x}][N[\vec{L}/\vec{x}]/x], N[\vec{L}/\vec{x}] \rangle \in \mathcal{R} : \mathsf{P}_\perp(\sigma)$$

By the argument above for the empty environment:

$$\vdash \langle \mathsf{fix}\, x. (M[\vec{L}/\vec{x}]), N[\vec{L}/\vec{x}] \rangle \in \mathcal{R} : \mathsf{P}_\perp(\sigma)$$

But $\mathsf{fix}\, x. (M[\vec{L}/\vec{x}]) = (\mathsf{fix}\, x. M)[\vec{L}/\vec{x}]$, and so:

$$\vdash \langle (\mathsf{fix}\, x. M)[\vec{L}/\vec{x}], N[\vec{L}/\vec{x}] \rangle \in \mathcal{R} : \mathsf{P}_\perp(\sigma)$$

Therefore $\Gamma \vdash \langle \mathsf{fix}\, x. M, N \rangle \in Opn(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\sigma)$. $\square$

Chapter 6

Discussion

In this section we summarise the work presented in this dissertation and then discuss applications and future research.

6.1 Summary

Chapter 2 reviews and demonstrates relationships between ordinals (including recursive ordinals), well-founded relations, trees, transition systems (labelled and unlabelled, with and without divergence), and the coinductively-defined similarity and bisimilarity relations. The variants of similarity and bisimilarity upon transition systems with divergence serve as an introduction to the more complex typed transition systems defined in chapter 4, and are used to compare different binary choice operators from the literature.

Chapter 3 introduces a λ-calculus $\mathscr{L}$ that exhibits a general form of erratic non-determinism. To allow more general definitions and results, types and terms may be infinite objects and are not restricted to recursive trees. The operational semantics is presented as a reduction semantics and an evaluation semantics. The latter makes use of the duality between least and greatest fixed-points to present the inductively-defined must convergence predicate as the complement of the coinductively-defined may divergence predicate, and this relationship turns out to be useful for the compatibility theorems of chapter 5. A family of non-deterministic λ-calculi with uniformly defined operational semantics is generated using closure conditions upon sets of well-typed terms. This allows consideration of λ-calculi with more restrictive forms of erratic non-determinism, which is important because some similarity and bisimilarity relations are sensitive to the forms of erratic non-determinism present in the language.

Chapter 4 identifies typed transition systems as the appropriate abstract structures representing the behaviour of $\mathscr{L}$ and its language fragments. Typed transition systems are a special case of labelled transition systems with divergence, and so the definitions of the variants of similarity and bisimilarity from chapter 2 can be replayed. General inclusions are established between the relations.

A typed transition system $\mathscr{S}$ is defined by interpreting each type constructor by its set-theoretic counterpart, where the computation type $\mathsf{P}_{\perp}(\sigma)$ is interpreted by the set of non-empty subsets of the lift of the interpretation of σ. This is possible because the type system for $\mathscr{L}$, and hence typed transition systems, does not permit recursive types. The definition is determined by the structure of typed transition systems, but (unsurprisingly) cannot be used as a denotational model because it lacks general fixed-points. It is nevertheless useful for constructing examples that distinguish the variants of similarity, mutual similarity, and bisimilarity, and for formalising properties of typed transition systems. In addition, a category of maps between typed transition systems is defined and every typed transition system that can be quotiented by convex bisimilarity, and that satisfies another minor condition, is the source of a map with target $\mathscr{S}$. Intuitively, the target of a map is a partial quotient of the source typed transition system, with additional elements.

Chapter 5 defines typed transition systems for $\mathscr{L}$ and its fragments. The variants of similarity and bisimilarity correspond to their usual definitions for non-deterministic λ-calculi. The lower, upper, and convex variants of similarity are shown to be compatible for all fragments using an extension (due to Lassen and discovered independently by the author) of techniques due to Howe and Ong. In particular, the compatibility results apply to fragments exhibiting finite non-determinism $?\langle\underline{0},\underline{1}\rangle$, countable non-determinism $?\underline{\omega}$, and more complex forms of indexed erratic non-determinism such as $?\underline{A}$, where $A \subseteq_{\mathrm{ne}} \omega$ is a non-recursively enumerable set.

Another technique, also due to Howe, is used to prove compatibility of the variants of bisimilarity. However, an additional assumption is required to handle infinitary terms and "incomplete" fragments. This is unfortunate, but not overly restrictive because the excluded infinitary terms are rarely used (and the assumption holds for the common forms of indexed erratic non-determinism mentioned above), and term constructors that cause fragments to be "incomplete" would be replaced by term constructors of lower arity, e.g., $?\langle\underline{0},\underline{1}\rangle$ would be a term constructor of arity 0 rather than a term constructor of arity 2 with immediate subterms $\underline{0}$ and $\underline{1}$.

A novel technique is used to prove compatibility of Lassen's refinement similarity by relating the dual of the transitive closure of Howe's congruence candidate to the transitive closure of Howe's congruence candidate for the dual of the underlying relation.

The compatibility proofs apply not to a single language, but to collections of languages obtained as fragments of $\mathscr{L}$. The proofs themselves are greatly simplified by the use of Lassen's relational operators and the fact that the variants of similarity and bisimilarity are defined in terms of only two simulation functions, $\langle\cdot\rangle^{\mathscr{T}}_{\mathrm{LS}}$ and $\langle\cdot\rangle^{\mathscr{T}}_{\mathrm{US}}$, on relations.

Relative definability of programs with respect to convex bisimilarity is considered, with the focus on programs of type $\mathsf{P}_{\perp}(\mathtt{nat})$. This is a robust notion because convex bisimilarity at this type does not depend upon the fragment. The following chain exists with respect to relative definability (where $\Omega\cup?\langle\underline{0},\underline{1}\rangle =^{\mathscr{M}_0}_{\mathrm{CB}} \Omega\cup?\underline{\omega}$):

$$?\langle\underline{0}\rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega\cup?\langle\underline{0}\rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} \Omega\cup?\langle\underline{0},\underline{1}\rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\langle\underline{0},\underline{1}\rangle \leq^{\mathscr{M}_0}_{\mathrm{CB}} ?\underline{\omega}$$

For each consecutive pair, the second program can be used to distinguish programs from the deterministic, recursive fragment $\mathscr{L}(\emptyset)$ that the first program cannot. In addition, if $A \subseteq_{\mathrm{ne}} \omega$ is a non-recursively enumerable set, then $?\underline{A}$ can be used to distinguish programs from $\mathscr{L}(?\underline{\omega})$ that $?\underline{\omega}$ cannot. This is relevant to the study of denotational models for erratic non-determinism because a model for a more expressive form of erratic non-determinism will be too discriminating for a programming language with less expressive forms of erratic non-determinism.

Finally, the Scott induction principle is proven for the lower, upper, and convex variants of similarity in all language fragments. Results for lower similarity for finite non-determinism and countable non-determinism, and upper similarity for finite non-determinism, are unpublished but were known to Lassen and can be proven using techniques similar to those developed in his dissertation [Las98b] for contextual approximation. The novel technique introduced here applies to all three relations by examining the reductions of terms with substitutions of programs, applying single instances of substitutions only when they are in a reduction context, and avoiding the use of contextual substitution.

6.2 Further Work

There are several avenues for future work, and we divide them roughly into applications for non-deterministic/concurrent higher-order languages and to denotational models.

Non-Deterministic λ-Calculi

The operational theory of λ-calculi exhibiting erratic non-determinism is well-developed, and it is tempting to look for a specification and refinement formalism that makes use of structured data types and the higher-order nature of the λ-calculus, as opposed to Dijkstra's more frugal Guarded Command Language (GCL). As a first step, we discuss some of the issues involved in translating the GCL to $\mathscr{L}$.

Dijkstra's GCL [Dij76, dB80, Kal90][1] is a minimal imperative language with constructs for non-deterministic alternation and iteration. Variables are often booleans, natural numbers, integers, or arrays. For the sake of argument, we consider two variables, a a natural number and b an array of length 5 of natural numbers. Define the type σ by:

$$\sigma \stackrel{\text{def}}{=} \mathsf{nat} \times (\mathsf{sum}\,\langle \mathsf{unit} \mid n < 5 \rangle \to \mathsf{nat})$$

Following the usual approach to state transformer semantics of simple imperative languages [Plo83], we would like to transform GCL programs using the variables a and b into programs of $\mathscr{L}$ with type $\sigma \to \mathsf{P}_\perp(\sigma)$. If P and Q are GCL programs and their translations are $[\![P]\!], [\![Q]\!] \in \mathscr{L}_0(\sigma \to \mathsf{P}_\perp(\sigma))$, then the translation of the sequential composition $P;Q$ is given by:

$$[\![P;Q]\!] \stackrel{\text{def}}{=} \lambda x.\, \mathsf{let}\, y \Leftarrow [\![P]\!]\, x\, \mathsf{in}\, [\![Q]\!]\, y \in \mathscr{L}_0(\sigma \to \mathsf{P}_\perp(\sigma))$$

The translations of alternation and iteration statements must evaluate all guards (boolean expressions that must not diverge) and then non-deterministically choose between the statements whose guards are true. Alternation introduces the possibility of a program failing when no guards are true [AO91]. Failure can be identified with divergence or modelled using a coproduct type, in which case it must be propagated by translations.

From a precondition and a postcondition we can define a program that is the minimal program, with respect to upper similarity, satisfying the precondition and postcondition. If we consider

[1] de Bakker's book [dB80] contains a formal treatment of the GCL that is particularly useful in this context.

only the variable a, as well as a precondition predicate $pre \subseteq \omega$ and a postcondition relation $post \subseteq \omega \times \omega$, then one such program is:

$$\lambda x.\,\mathsf{case}\,x\,\mathsf{of}\,\langle y_m.M_m \mid m < \omega\rangle \in \mathscr{L}_0(\mathsf{nat} \to \mathsf{P}_\perp(\mathsf{nat}))$$

where, for $m \in \omega$, we define $M_m \in \mathscr{L}_0(\mathsf{P}_\perp(\mathsf{nat}))$ and $A_m \subseteq \omega$ by:

$$M_m \stackrel{\text{def}}{=} \begin{cases} \Omega & \text{if } m \notin pre \text{ or } A_m = \emptyset \\ ?\underline{A_m} & \text{otherwise} \end{cases}$$
$$A_m \stackrel{\text{def}}{=} \{n \mid \langle m, n\rangle \in post\}$$

Given a representation of a natural number as an initial state, this program diverges if the initial state does not satisfy the precondition or there are no terminal states that satisfy the postcondition with this initial state.

In general, it is necessary to use a case statement indexed by ω to make use of variables of type nat, because variables of $\mathscr{L}$ cannot appear in the indexing expression of an erratic choice term constructor. Consequently, it may be difficult to specify properties of lists of unbounded size. For example, a similar program for preconditions and postconditions over the variables a and b must have 6 nested case statements, 1 for a and 5 for b.

A second program satisfies a precondition and postcondition if it is greater than or equal to, with respect to upper similarity, the program derived from those conditions. The finer relations, refinement similarity, convex similarity, upper bisimilarity, and convex bisimilarity, may be useful for identifying stronger relationships between programs. If the bisimilarity relations are used then it would be sensible to restrict attention from $\mathscr{L}$ to the fragment $\mathscr{M}$ in order to have compatibility.

The next step would be to investigate how proof principles for GCL programs, particularly loops, can be carried over to their translations into $\mathscr{L}$ programs. If this proves successful, it would be interesting to generalise those principles for developing functional programs from non-deterministic functional programs representing specifications. This would allow structured (coproduct and product) and function types to be used for program development. Such types are not normally available in the GCL, perhaps because preconditions and postconditions use program variables within the terms of well-formed formulae. If the types of program variables are not flat, then it is tempting to introduce a more expressive term language, but this can lead to partially-defined terms with their associated problems [CJ91].

It should be straightforward to extend $\mathscr{L}$ with algebraic data types, re-prove the compatibility and Scott induction results, and construct a structure that corresponds to $\mathscr{S}$. With such a language, it may be possible to adapt some of Bird and de Moor's development techniques [BdM97]. However, the settings are mismatched because Bird and de Moor's model allows a program to have no outcome and does not incorporate divergence and general recursion. Instead primitive recursion is used to define functions over (well-founded) algebraic data types. Removing the fixed-point term constructor or creating a subtype $\mathsf{P}(\sigma)$ of $\mathsf{P}_\perp(\sigma)$ that permits non-determinism but not divergence solves the latter problem. The possibility of no outcome follows from a more serious problem: much of Bird and de Moor's work is based upon specifying the behaviour of a program in terms of the dual of a function. It seems unlikely that it would be possible to give an operational semantics for the dual of a higher-order program. In the special case of a function between algebraic data types, it may be possible to define a program that acts as the dual of a program up to convex bisimilarity. For example, the dual of a program M of type $\mathsf{nat} \to \mathsf{P}(\mathsf{nat})$ could be a program M^{op}, also of type $\mathsf{nat} \to \mathsf{P}(\mathsf{nat})$, defined by:

$$M^{\mathrm{op}} \stackrel{\mathrm{def}}{=} \lambda x.\, \mathsf{case}\, x\, \mathsf{of}\, \langle y_n.?\underline{A_n} \mid n < \omega \rangle$$

where $A_n \subseteq \omega$ is defined by:

$$A_n \stackrel{\mathrm{def}}{=} \{m \in \omega \mid \exists N. M\,\underline{m} \Downarrow^{\mathrm{may}} [N] \wedge N \simeq_{\mathrm{CB}}^{\mathscr{L}_0(\mathcal{E})} \underline{n}\}$$

In addition, there will be countably many convex bisimilarity equivalence classes of programs with a fixed algebraic data type such as finite lists or trees, and so we can expect to be able to define the dual of functions operating on such data types. However, in general, it would not be possible to define the dual of an open term or to internalise the dual operation within the programming language.

The addition of recursive types would cause several problems. The duals suggested above would have no analogue for a non-deterministic program with type $\mathsf{nat} \to \mathsf{P}(\mathsf{list}(\mathsf{nat}))$, where $\mathsf{list}(\mathsf{nat})$ represents finite or infinite lists of natural numbers, because there is no way to recognise infinite lists. Countable non-determinism may be useful for specifying such programs by testing that all finite prefixes of lists satisfy some property (see the take lemma in [BdM93]). Recursive types would also introduce divergence at all types, not just computation types, and would prevent the definition of a structure corresponding to $\mathscr{S}$. Similarity and bisimilarity could be defined, and the compatibility and Scott induction results should hold with few changes. However, a new strategy may be required to prove compatibility of the variants of bisimilarity if the possibility of divergence at value types weakens lemma 4.2.5.

Concurrent λ-Calculi

The operational semantics derived here for the binary erratic choice operator $M \cup N$ differs from Jeffrey's [Jef99] operational semantics for internal choice:

$$\frac{M \rightarrowtriangle M'}{M \,[\!]\, N \rightarrowtriangle M' \,[\!]\, N} \qquad \frac{N \rightarrowtriangle N'}{M \,[\!]\, N \rightarrowtriangle M \,[\!]\, N'}$$

$$K \,[\!]\, N \rightarrowtriangle K \qquad M \,[\!]\, K \rightarrowtriangle K$$

The variants of similarity and bisimilarity considered here cannot distinguish between $M \cup N$ and $M \,[\!]\, N$, because although their reduction trees are different, they do have the same leaves (labelled with canonical programs). However, higher-order weak similarity and bisimilarity can distinguish them because those relations are sensitive to reduction behaviour. If higher-order weak similarity is denoted by $\preceq$, then, for programs M, N of the same computation type, we have $M \preceq N$ if and only if:

$$(\forall M'.M \rightarrowtriangle M' \Longrightarrow \exists N'.N \rightarrowtriangle^* N' \wedge M' \preceq N') \wedge$$
$$(\forall M'.M = [M'] \Longrightarrow \exists N'.N \rightarrowtriangle^* [N'] \wedge M' \preceq N')$$

Higher-order weak bisimilarity is defined in the obvious way. For example, convex bisimilarity relates the programs $[\star] \cup \Omega$ and $\mathsf{fix}x.[\star] \cup x$, both of type $\mathsf{P}_\perp(\mathsf{unit})$, but they are not related by higher-order weak bisimilarity because Ω and $\mathsf{fix}x.[\star] \cup x$ are not related by higher-order weak bisimilarity. The interest in higher-order weak similarity and bisimilarity is motivated by applications to languages, such as CML, with concurrency and communication primitives.

It would be useful to understand the relationship between languages with binary erratic choice with respect to the lower, upper, and convex variants of similarity and bisimilarity and languages with internal choice with respect to higher-order weak similarity and bisimilarity.

Denotational Models

The variants of bisimilarity are strictly finer than the corresponding variants of mutual similarity (see example 4.4.5 and the discussion in section 5.2). This prompts the question, are partial orders appropriate for modelling the variants of bisimilarity, and, if not, which properties characterise the fixed-point operator with respect to the variants of bisimilarity? There are a number of related equational treatments of fixed-point operators: iteration and iterative theories [Blo89, Wag94], traced monoidal categories [JSV96, Has97], dinatural fixed-point operators [Sim93, PS00], and FLR_0 [Mos89, Mos95]. It would be interesting to see whether the variants of bisimilarity satisfy such properties, because then the quotient of the language fragment by the bisimilarity would be a model of the equational axioms that does not arise directly from a partial order, in contrast to the usual domain-theoretic examples.

For example, the dinaturality property is that, for a relation $\mathcal{R} \in Rel_0(\mathcal{E})$ and terms $\Gamma, x : \mathsf{P}_\perp(\sigma) \vdash M : \mathsf{P}_\perp(\tau)$ and $\Gamma, y : \mathsf{P}_\perp(\tau) \vdash N : \mathsf{P}_\perp(\sigma)$, we have:

$$\Gamma \vdash \langle \mathsf{fix}y.M[N/x], M[\mathsf{fix}x.N[M/y]/x] \rangle \in Opn(\mathcal{E}, \mathcal{R}) : \mathsf{P}_\perp(\tau)$$

It may be possible to give a direct proof of dinaturality with respect to the variants of bisimilarity using techniques similar to those used for the Scott induction result (theorem 5.7.9). An alternative is to try to adapt Lassen and Moran's [LM99] proof of dinaturality with respect to mutual cost similarity for a λ-calculus with ambiguous choice.

Bibliography

[Abr83] S. Abramsky. On semantic foundations for applicative multiprogramming. In J. Diaz, editor, *Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*. Springer-Verlag, 1983. 57, 169

[Abr87a] S. Abramsky. *Domain Theory and the Logic of Observable Properties*. PhD thesis, University of London, 1987. 57

[Abr87b] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53:225–241, 1987. 33, 39

[Abr89] S. Abramsky. A generalized Kahn principle for abstract asynchronous networks. In M. Mislove M. Main, A. Melton and D. Schmidt, editors, *Proceedings of the 5th Conference on Mathematical Foundations of Programming Semantics*, volume 442 of *Lecture Notes in Computer Science*, pages 1–21. Springer-Verlag, 1989. 16

[Abr90] S. Abramsky. The lazy lambda calculus. In Turner [Tur90b], pages 65–117. 9, 10, 11, 20, 21, 39, 88, 88, 88, 88, 120, 134

[Abr91] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92(2):161–218, 1991. 8, 39

[AC98] R. M. Amadio and P.-L. Curien. *Domains and Lambda-Calculi*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998. 10, 44

[Acz77] P. Aczel. An introduction to inductive definitions. In Barwise [Bar77], pages 739–782. 9, 23, 35, 35, 42

[Acz88] P. Aczel. *Non-Well-Founded Sets*, volume 14 of *CSLI Lecture Notes*. Center for the Study of Language and Information, Stanford University, 1988. 9, 32, 43

[Acz94] P. Aczel. Final universes of processes. *Lecture Notes in Computer Science*, 802:1–28, 1994. 9

[AD95] R. M. Amadio and M. Dam. Reasoning about higher-order processes. In P. D. Mosses, M. Nielsen, and M. I. Schwarzbach, editors, *TAPSOFT '95: Theory and Practice of Software Development, 6th International Joint Conference CAAP/FASE*, volume 915 of *Lecture Notes in Computer Science*, pages 202–216. Springer-Verlag, 1995. Full version as SICS Research Report RR:94/18, October 1994. 17

[AGM94] S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors. *Semantic Structures*, volume 3 of *Handbook of Logic in Computer Science*. Clarendon Press, 1994. 186, 198

[AJ94] S. Abramsky and A. Jung. Domain theory. In Abramsky et al. [AGM94]. 44

[AJM94] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (extended abstract). In M. Hagiya and J. C. Mitchell, editors, *Theoretical Aspects of Computer Software TACS'94*, volume 789 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 1994. 9

[ALT95] R. M. Amadio, L. Leth, and B. Thomsen. From a concurrent λ-calculus to the π-calculus. In H. Reichel, editor, *Fundamentals of Computation Theory (FCT '95, 10th International Conference, Dresden, Germany)*, volume 965 of *Lecture Notes in Theoretical Computer Science*, pages 106–115. Springer-Verlag, 1995. Full version as Technical Report ECRC-95-18. 17

[Ama93] R. M. Amadio. On the reduction of CHOCS bisimulation to π-calculus bisimulation. In E. Best, editor, *CONCUR '93: 4th International Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 112–126. Springer-Verlag, August 1993. 17

[Ama94] R. M. Amadio. Translating core Facile. Technical Report ECRC-TR-3-94, European Computer-Industry Research Center, GmbH, Munich, 1994. Also available as a technical report from CRIN(CNRS)-Inria (Nancy). 17

[AO91] K. R. Apt and E.-R. Olderog. *Verification of Sequential and Concurrent Programs*. Texts and Monographs in Computer Science. Springer-Verlag, 1991. 5, 6, 181

[AO93] S. Abramsky and C.-H. L. Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105(2):159–267, 1993. 10, 20, 21, 88

[AP86] K. R. Apt and G. D. Plotkin. Countable nondeterminism and random assignment. *Journal of the ACM*, 33(4):724–767, October 1986. 6, 20, 81, 81, 157

[BA81] J. D. Brock and W. B. Ackerman. Scenarios: A model of non-determinate computation. In J. Díaz and I. Ramos, editors, *Proceedings of the International Colloquium on Formalization of Programming Concepts*, volume 107 of *Lecture Notes in Computer Science*, pages 252–259. Springer-Verlag, 1981. 16

[Bac80] R. J. R. Back. Semantics of unbounded nondeterminism. In J. W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium*, volume 85 of *Lecture Notes in Computer Science*, pages 51–63. Springer-Verlag, 1980. 6, 6

[Bac88] R. J. R. Back. A calculus of refinements for program derivations. *Acta Informatica*, 25:593–624, 1988. 6

[Bar77] J. Barwise, editor. *Handbook of Mathematical Logic*. Number 90 in Studies in Logic and the Foundations of Mathematics. North-Holland, 1977. 185, 192, 192

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, revised edition, 1984. 9, 10, 58, 69, 88

[Bar92] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Background: Computational Structures*, volume 2 of *Handbook of Logic in Computer Science*. Clarendon Press, 1992. 55

[BdM93] R. S. Bird and O. de Moor. Solving optimisation problems with catamorphisms. In R. S. Bird, C. C. Morgan, and J. C. P. Woodcock, editors, *2nd International Conference on the Mathematics of Program Construction*, volume 669 of *Lecture Notes in Computer Science*, pages 45–66. Springer-Verlag, 1993. 183

[BdM97] R. S. Bird and O. de Moor. *Algebra of Programming*. International Series in Computer Science. Prentice Hall, 1997. 182

[Ber98] K. L. Bernstein. A congruence theorem for structured operational semantics of higher-order languages. In *13th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1998. 11

[BL95] G. Boudol and C. Laneve. λ-calculus, multiplicities and the π-calculus. Technical Report INRIA Res. Report 2581, INRIA Sophia-Antopolis, June 1995. 13

[Blo89] S. L. Bloom. The equational logic of iterative processes. In J. Csirik, J. Demetrovics, and F. Gécseg, editors, *Fundamentals of Computation Theory*, volume 380 of *Lecture Notes in Computer Science*, pages 47–57. Springer-Verlag, 1989. 184

[BM96] J. Barwise and L. Moss. *Vicious Circles: on the Mathematics of Non-Wellfounded Phenomena*. Number 60 in CSLI Lecture Notes. CSLI publications, 1996. 9, 32, 42, 43

[Bou93] G. Boudol. The λ-calculus with multiplicities. Technical Report INRIA Res. Report 2025, INRIA, September 1993. 13

[Bou94a] G. Boudol. The discriminating power of multiplicities in the λ-calculus. Technical Report INRIA Res. Report 2441, INRIA, December 1994. 13

[Bou94b] G. Boudol. Lambda-calculi for (strict) parallel functions. *Information and Computation*, 108:51–127, 1994. 13

[Bou97a] G. Boudol. The π-calculus in direct style. In *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 228–241, January 1997. 17

[Bou97b] G. Boudol. Typing the use of resources in a concurrent calculus. In R. K. Shyamasundar and K. Ueda, editors, *Advances in Computing Science, Proceedings of ASIAN '97, the Asian Computing Science Conference (Kathmandu, Nepal)*, volume 1345 of *Lecture Notes in Computer Science*, pages 239–253. Springer-Verlag, 1997. 17

[BPR90] D. B. Benson, P. Panangaden, and J. R. Russell. Defining fair merge as a colimit: Towards a fixed-point theory for indeterminate dataflow. In M. Z. Kwiatkowska, M. W. Shields, and R. M. Thomas, editors, *Semantics for Concurrency, Leicester*, Workshops in Computing, pages 175–184. Springer-Verlag, 1990. 16

[Bro86] M. Broy. A theory for nondeterminism, parallelism, communication, and concurrency. *Theoretical Computer Science*, 45:1–61, 1986. 16

[Bro88] M. Broy. Nondeterministic data flow programs: How to avoid the merge anomaly. *Science of Computer Programming*, 10:65–85, 1988. 16

[BRW88] S. D. Brookes, A. W. Roscoe, and D. J. Walker. An operational semantics for CSP. Technical report, Programming Research Group, University of Oxford, 1988. 39

[Buc97] A. Bucciarelli. Degrees of parallelism in the continuous type hierarchy. *Theoretical Computer Science*, 177(1):59–71, April 1997. 18, 77

[Bur88] F. W. Burton. Nondeterminism with referential transparency in functional programming languages. *The Computer Journal*, 31(3):243–247, June 1988. 16

[CC92] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretations. In *Conference Record of the 19th ACM Symposium on Principles of Programming Languages*, pages 83–94, 1992. 15, 53, 73

[CG95] R. L. Crole and A. D. Gordon. A sound metalogical semantics for input/output effects. In L. Pacholski and J. Tiuryn, editors, *Computer science logic: 8th workshop, CSL '94*, volume 933 of *Lecture Notes in Computer Science*, pages 339–353. Springer-Verlag, 1995. 19, 59, 59

[Chu38] C. A. Church. The constructive second number class. *Bull. Amer. Math. Soc.*, 44:224–232, 1938. 46

[CJ91] J. H. Cheng and C. B. Jones. On the usability of logics which handle partial functions. In C. Morgan and J. C. P. Woodcock, editors, *Proceedings of the Third Refinement Workshop*, Workshops in Computing Series, pages 51–69, Berlin, 1991. Springer-Verlag. 182

[CK37] C. A. Church and S. C. Kleene. Formal definitions in the theory of ordinal numbers. *Fund. Math.*, 28:11–21, 1937. 46

[Cli82] W. Clinger. Nondeterministic call by need is neither lazy nor by name. In *ACM Symposium on Lisp and Functional Programming*, pages 226–234, 1982. 17

[CM93] P. Cenciarelli and E. Moggi. A syntactic approach to modularity in denotational semantics. In *Proceedings of the Conference on Category Theory and Computer Science, Amsterdam*, CWI Technical Report, 1993. 59

[CP92] R. L. Crole and A. M. Pitts. New foundations for fixpoint computations: FIX-hyperdoctrines and the FIX-logic. *Information and Computation*, 98(2):171–210, 1992. 19

[Cro93] R. L. Crole. *Categories for Types*. Cambridge Mathematical Textbooks. Cambridge University Press, 1993. 58, 59, 127

[Cut80] N. Cutland. *Computability: An Introduction to Recursive Function Theory*. Cambridge University Press, 1980. 77

[dB80] J. de Bakker. *Mathematical Theory of Program Correctness*. International Series in Computer Science. Prentice Hall, 1980. 4, 181, 181

[Den84] J. B. Dennis. Data flow computation. In M. Broy, editor, *Control Flow and Data Flow: Concepts of Distributed Programming.*, volume 14 of *NATO Advanced Science Institutes Series F: Computer and System Sciences*. Springer-Verlag, 1984. A collection of 5 essays. 16

[dGHLP94] P. di Gianantonio, F. Honsell, S. Liani, and G. D. Plotkin. Countable nondeterminism and uncountable limits. In B. Jonsson and J. Parrow, editors, *CONCUR '94: Concurrency Theory, 5th International Conference*, volume 836 of *Lecture Notes in Computer Science*, pages 130–145. Springer-Verlag, August 1994. 6

[Dij76] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976. 4, 5, 181

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks. Cambridge University Press, 1990. 30, 33

[Ear97] B. Earl. Tools for domain theory. Master's thesis, University of Oxford, Computing Laboratory, September 1997. 101

[Fau82] A. A. Faustini. *The Equivalence of an Operational and a Denotational Semantics for Pure Dataflow*. PhD thesis, University of Warwick, 1982. 16

[FF86] M. Felleisen and D. P. Friedman. Control operators, the SECD-machine and the λ-calculus. In *Formal Description of Programming Concepts III*, pages 193–217. North-Holland, 1986. 65

[FH83] M. Forti and F. Honsell. Set theory with free construction principles. *Annali Scuola Normale Supeiore di Pisa, Classe di Scienze*, 10:493–522, 1983. 32, 43

[FHJ95] W. Ferreira, M. Hennessy, and A. Jeffrey. A theory of weak bisimulation for core CML. Technical Report 05/95, University of Sussex, September 1995. 17, 68, 120

[FHL94] M. Forti, F. Honsell, and M. Lenisa. Processes and hyperuniverses. In I. Prívara, B. Rovan, and P. Ruzicka, editors, *Mathematical Foundations of Computer Science 1994 19th International Symposium*, volume 841 of *Lecture Notes in Computer Science*, pages 352–363. Springer-Verlag, 1994. 9, 32, 43

[Fio94] M. P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. PhD thesis, University of Edinburgh, 1994. 59

[Fra86] N. Francez. *Fairness*. Texts and Monographs in Computer Science. Springer-Verlag, 1986. 5

[Gal91] J. H. Gallier. What's so special about Kruskal's theorem and the ordinal Γ_0? A survey of some results in proof theory. *Annals of Pure and Applied Logic*, 53(3):199–260, September 1991. 23

[Gir87] J.-Y. Girard. *Proof Theory and Logical Complexity*, volume 1 of *Studies in Proof Theory*. Bibliopolis, 1987. 23, 45, 46

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Number 7 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989. 75

[Gor94] A. D. Gordon. *Functional Programming and Input/Output*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1994. 11, 19, 21, 58, 59, 59, 75, 89, 120

[Gor95a] A. D. Gordon. Bisimilarity as a theory of functional programming. BRICS Notes Series NS-95-3, Department of Computer Science, University of Aarhus, 1995. 11, 38, 89, 120

[Gor95b] A. D. Gordon. A tutorial on co-induction and functional programming. In *Proceedings of the 1994 Glasgow Workshop on Functional Programming*, Workshops in Computing, 1995. 11, 39, 120

[Gri81] D. Gries. *The Science of Programming*. Texts and Monographs in Computer Science. Springer-Verlag, 1981. 3, 6

[Gru93] J. Grundy. *A Method of Program Refinement*. PhD thesis, University of Cambridge, 1993. 4

[Gun92] C. A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing. MIT Press, 1992. 9, 10, 11, 33, 44, 72, 76, 88, 112, 169

[HA80] M. C. B. Hennessy and E. A. Ashcroft. A mathematical semantics for a nondeterministic typed λ-calculus. *Theoretical Computer Science*, 11(3):227–245, 1980. 12

[Has97] M. Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of Let and Letrec*. PhD thesis, University of Edinburgh, 1997. 184

[Hen82] P. Henderson. Purely functional operating systems. In J. Darlington, P. Henderson, and D. Turner, editors, *Functional Programming and its Applications*, pages 177–192. Cambridge University Press, 1982. 16

[Hen94] M. Hennessy. Higher-order process and their models. In S. Abiteboul and E. Shamir, editors, *Automata, Languages and Programming, 21st International Colloquium*, volume 820 of *Lecture Notes in Computer Science*, pages 286–303, Jerusalem, Israel, July 1994. Springer-Verlag. 17

[HM95] J. Hughes and A. K. Moran. Making choices lazily. In *Functional Programming and Computer Architecture*, pages 108–119. ACM Press, June 1995. 12, 14, 17, 19, 73

[HO89] J. Hughes and J. T. O'Donnell. Expressing and reasoning about non-deterministic functional programs. In *Functional Programming (Glasgow)*, Workshops in Computing, pages 308–328. Springer-Verlag, 1989. 16

[HO00] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF. *Information and Computation*, 163:285–408, 2000. 9

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969. 3

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice Hall, 1985. 7

[How89] D. J. Howe. Equality in lazy computation systems. In *Proceedings, 4th Annual Symposium on Logic in Computer Science*, pages 198–203. IEEE Computer Society Press, 1989. 11, 12, 21, 89, 129, 134, 135

[How96] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996. 11, 12, 18, 21, 134, 142

[HS86] J. R. Hindley and J. P. Seldin. *Introduction to Combinators and λ-calculus*. Number 1 in London Mathematical Society Student Texts. Cambridge University Press, 1986. 69, 88

[IS98] H. Ibraheem and D. A. Schmidt. Adapting big-step semantics to small-step style: Coinductive interpretations and "higher-order" derivations. In A. D. Gordon, A. M. Pitts, and C. Talcott, editors, *Proc. 2nd Workshop on Higher Order Operational Techniques in Semantics, Stanford, December 1997*, volume 10 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1998. 73

[Jef95] A.S.A. Jeffrey. A fully abstract semantics for a concurrent functional language with monadic types. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 255–264, San Diego, California, 26–29 June 1995. IEEE Computer Society Press. 17, 120

[Jef99] A.S.A. Jeffrey. A fully abstract semantics for a higher-order functional language with nondeterministic computation. *Theoretical Computer Science*, 228(1-2):105–150, 1999. 13, 19, 59, 59, 68, 183

[Joh87] P. T. Johnstone. *Notes on Logic and Set Theory*. Cambridge Mathematical Textbooks. Cambridge University Press, 1987. 23, 25

[JR97] B. Jacobs and J.J.M.M. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of EATCS*, 62:222–259, 1997. 10, 33

[JSV96] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Math. Proc. Camb. Phil. Soc.*, 119:447–468, 1996. 184

[Kah74] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information Processing '74: Proceedings of the IFIP Congress*, pages 471–475. North-Holland, 1974. 16

[Kah87] G. Kahn. Natural semantics. In F. J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, editors, *STACS 87*, volume 247 of *Lecture Notes in Computer Science*, pages 22–39. Springer-Verlag, 1987. 9, 72

[Kal90] A. Kaldewaij. *Programming: the derivation of algorithms*. International Series in Computer Science. Prentice Hall, 1990. 6, 181

[Kei77] H. J. Keisler. Fundamentals of model theory. In Barwise [Bar77]. 57

[Kle38] S. C. Kleene. On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3(4):150–155, December 1938. 46

[KSS99] A. Kutzner and M. Schmidt-Schauß. A non-deterministic call-by-need lambda calculus. *ACM SIGPLAN Notices*, 34(1):324–335, January 1999. 12, 17

[Kun77] K. Kunen. Combinatorics. In Barwise [Bar77]. 23

[Kun80] K. Kunen. *Set Theory: an introduction to independence proofs*, volume 102 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1980. 23, 58

[Las97] S. B. Lassen. Action semantics reasoning about functional programs. *Math. Struct. in Comp. Science*, 7(5):557–589, 1997. 12, 12, 13, 73, 135

[Las98a] S. B. Lassen. Relational reasoning about contexts. In A. D. Gordon and A. M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, pages 91–135. Cambridge University Press, 1998. 38

[Las98b] S. B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, Department of Computer Science, University of Aarhus, Denmark, 1998. 12, 12, 12, 18, 21, 44, 69, 75, 80, 100, 125, 129, 135, 169, 181

[Lau93] J. Launchbury. A natural semantics for lazy evaluation. In *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 144–154, 1993. 17

[Lav93] C. Lavatelli. Non-deterministic lazy λ-calculus vs π-calculus. Technical Report Technical Report 93-15, DMI, LIENS, 1993. 13, 13

[Leh76] D. J. Lehmann. *Categories for Fixpoint Semantics*. PhD thesis, Department of Computer Science, University of Warwick, 1976. 57, 169

[Lev79] A. Levy. *Basic Set Theory*. Perspectives in Mathematical Logic. Springer-Verlag, 1979. 23, 35

[Lic96] B. Lichtenthäler. *Degrees of Parallelism*. Masters thesis, Informatik-Bericht, January 1996. Abridged English version. 18, 77

[LM99] S. B. Lassen and A. K. Moran. Unique fixed point induction for McCarthy's Amb. In *Proc. of MFCS'99, the 26th Symposium on Mathematical Foundations of Computer Science*, volume 1672 of *Lecture Notes in Computer Science*, pages 198–208. Springer-Verlag, September 1999. 184

[LP98] S. B. Lassen and C. S. Pitcher. Similarity and bisimilarity for countable non-determinism and higher-order functions. In A. D. Gordon, A. M. Pitts, and C. Talcott, editors, *Proc. 2nd Workshop on Higher Order Operational Techniques in Semantics, Stanford, December 1997*, volume 10 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1998. 12, 13, 135

[LS86] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. Number 7 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1986. 59

[McC63] J. McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–70. North-Holland, 1963. 14, 16

[Mil89] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989. 7, 8, 9, 30, 35, 38, 39, 57

[Mil90] R. Milner. Functions as processes. In M. S. Paterson, editor, *Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 166–180. Springer-Verlag, 1990. 13

[Mil91] R. Milner. The polyadic π-calculus: A tutorial. Technical report, LFCS, University of Edinburgh, 1991. 7

[Mit96] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996. 9, 21, 88, 99

[Mog89a] E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, University of Edinburgh, 1989. Lecture Notes for course CS 359, Stanford University. 19, 55

[Mog89b] E. Moggi. Computational lambda-calculus and monads. In *Proceedings, 4th Annual Symposium on Logic in Computer Science*, pages 14–23. IEEE Computer Society Press, 1989. 19, 55, 59

[Mog91] E. Moggi. Notions of computations and monads. *Information and Computation*, 93(1):55–92, 1991. 19, 55, 59

[Mor90] C. C. Morgan. *Programming from Specifications*. International Series in Computer Science. Prentice Hall, 1990. 6

[Mor94] A. K. Moran. Natural semantics for non-determinism. Licentiate thesis, Chalmers University of Technology and University of Göteborg, May 1994. 12, 12, 14

[Mor98] A. K. Moran. *Call-by-name, Call-by-need, and McCarthy's Amb*. PhD thesis, Department of Computing Science, Chalmers University of Technology, Göteborg, Sweden, September 1998. 12, 12, 14, 16, 16, 17, 19, 70, 75

[Mos74] Y. N. Moschovakis. *Elementary Induction on Abstract Structures*. Studies in Logic and the Foundations of Mathematics. North-Holland, 1974. 35

[Mos89] Y. N. Moschovakis. The formal language of recursion. *The Journal of Symbolic Logic*, 54(4):1216–1252, December 1989. 184

[Mos90] Y. N. Moschovakis. *Descriptive Set Theory*. North-Holland, 1990. 46

[Mos91] Y. N. Moschovakis. A model of concurrency with fair merge and full recursion. *Information and Computation*, 93(1):114–171, 1991. 16

[Mos95] Y. N. Moschovakis. Computable concurrent processes. *Theoretical Computer Science*, 139(1–2):243–273, 1995. 16, 184

[Mos98] Y. N. Moschovakis. A game-theoretic, concurrent and fair model of the typed lambda-calculus, with full recursion. In M. Nielsen and W. Thomas, editors, *CSL '97*, volume 1414 of *Lecture Notes in Computer Science*, pages 341–359. Springer-Verlag, 1998. 16

[MP86] A. Moitra and P. Panangaden. Finitary choice cannot express fairness: A metric space technique. Technical Report TR86-788, Cornell University, 1986. 16

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992. 7

[MST96] I. A. Mason, S. F. Smith, and C. L. Talcott. From operational semantics to domain theory. *Information and Computation*, 128(1):26–47, 1996. 169

[MT88] R. Milner and M. Tofte. Co-induction in relational semantics. Technical Report ECS-LFCS-88-58, LFCS, University of Edinburgh, 1988. 8

[MW95] Y. N. Moschovakis and G. T. Whitney. Powerdomains, powerstructures and fairness. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic '94*, volume 933 of *Lecture Notes in Computer Science*, pages 382–396. Springer-Verlag, 1995. 6, 169

[Odi89] P. Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1989. 18, 23, 45, 46, 77, 158

[Ong92a] C.-H. L. Ong. Concurrent lambda calculus, and a general pre-congruence theorem for applicative bisimulation. Preliminary version, August 1992. 13, 21, 21, 89, 89, 134

[Ong92b] C.-H. L. Ong. Functions, non-determinism and concurrency. Working draft, January 1992. 13, 89, 89

[Ong93] C.-H. L. Ong. Non-determinism in a functional setting. In *Proceedings, 8th Annual Symposium on Logic in Computer Science*, pages 275–286. IEEE Computer Society Press, 1993. 12, 12, 18, 89, 89, 169

[OP93] C.-H. L. Ong and A. M. Pitts. Systematic programming semantics, 1993. Case for support. 21, 89

[Par79] D. M. Park. On the semantics of fair parallelism. In D. Bjørner, editor, *Abstract Software Specification*, volume 86 of *Lecture Notes in Computer Science*, pages 504–526. Springer-Verlag, 1979. 6, 8, 35, 39

[Par81] D. M. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981. 6, 8, 35, 39

[PGM90] S. Prasad, A. Giacalone, and P. Mishra. Operational and algebraic semantics for Facile: A symmetric integration of concurrent and functional programming. In M. S. Paterson, editor, *Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 765–780. Springer-Verlag, 1990. 17

[Pit91] A. M. Pitts. Evaluation logic. In G. Birtwistle, editor, *4th Higher Order Workshop, Banff 1990*, Workshops in Computing, pages 162–189. Springer-Verlag, Berlin, 1991. 19

[Pit97] A. M. Pitts. Operationally-based theories of program equivalence. In P. Dybjer and A. M. Pitts, editors, *Semantics and Logics of Computation*. Cambridge University Press, 1997. Lectures given at the CLICS-II Summer School on Semantics and Logics of Computation, Isaac Newton Institute for Mathematical Sciences, Cambridge, UK, September 1995. 9, 65, 69, 169

[Plo75] G. D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975. 9

[Plo76] G. D. Plotkin. A powerdomain construction. *SIAM J. Comput.*, 5(3):452–487, 1976. 44, 169

[Plo77] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977. 9

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981. 7, 9, 11, 30, 65

[Plo82] G. D. Plotkin. A powerdomain for countable non-determinism (extended abstract). In M. Nielson and E. M. Schmidt, editors, *International Colloquium on Automata, Languages and Programs*, volume 140 of *Lecture Notes in Computer Science*, pages 418–428. Springer-Verlag, 1982. 6, 14

[Plo83] G. D. Plotkin. Domains. Course notes, 1983. 44, 169, 169, 181

[Plo85] G. D. Plotkin. Denotational semantics with partial functions. Lecture notes, C.S.L.I. Summer School, Stanford, 1985. 9

[Pot90] M. D. Potter. *Sets: An Introduction*. Oxford Science Publications. Oxford University Press, 1990. 23, 45

[PR88] P. Panangaden and J. Russell. A category-theoretic semantics for unbounded indeterminacy. Technical Report 88-957, Cornell University, December 1988. 57, 169

[PS87] P. Panangaden and V. Shanbhogue. On the expressive power of indeterminate network primitives. Technical Report 87-891, Cornell University, December 1987. 16

[PS88a] P. Panangaden and V. Shanbhogue. McCarthy's amb cannot implement fair merge. Technical Report 88-913, Cornell University, May 1988. 16

[PS88b] P. Panangaden and E. W. Stark. Computations, residuals, and the power of indeterminacy. In T. Lepistö and A. Saloman, editors, *15th ICALP*, volume 317 of *Lecture Notes in Computer Science*, pages 439–454. Springer-Verlag, 1988. 16

[PS00] G. Plotkin and A. K. Simpson. Complete axioms for categorical fixed-point operators. In *Fifteenth Annual IEEE Symposium on Logic in Computer Science*, pages 30–44, 2000. 184

[Rep92] J. H. Reppy. *Higher-order Concurrency*. PhD thesis, Department of Computer Science, Cornell University, June 1992. TR 92-1285. 17, 17

[Rep99] J. H. Reppy. *Concurrent Programming in ML*. Cambridge University Press, 1999. 17

[Rog67] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill Series in Higher Mathematics. McGraw-Hill, 1967. 18, 45, 77, 153, 158

[Ros88] A. W. Roscoe. Two papers on CSP. Technical Report PRG-67, Programming Resarch Group, Oxford University Computing Laboratory, July 1988. (An alternative order for the failures model & Unbounded nondeterminism in CSP). 57

[Ros98] A. W. Roscoe. *The Theory and Practice of Concurrency*. International Series in Computer Science. Prentice Hall, 1998. 7, 8, 39, 57

[Rus90] J. R. Russell. *Full Abstraction and Fixed-Point Principles for Indeterminate Computation*. PhD thesis, Department of Computer Science, Cornell University, April 1990. Available as TR90-1120. 16, 57, 169

[San93] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1993. 17

[San94] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Information and Computation*, 111(1):120–153, 1994. 13

[San97] D. Sands. From SOS rules to proof principles: An operational metatheory for functional languages. In *Proceedings of the 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 428–441, January 1997. 169

[Saz75] V. Yu. Sazonov. Sequentially and parallel computable functionals. In G. Goos and J. Hartmanis, editors, *λ-calculus and Computer Science Theory*, volume 37 of *Lecture Notes in Computer Science*, pages 312–319. Springer-Verlag, 1975. 18, 77

[Sco93] D. S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121(1–2):411–440, 1993. Reprint of an unpublished manuscript written in 1969. 9

[Sha90] V. Shanbhogue. *The Expressiveness of Indeterminate Dataflow Primitives*. PhD thesis, Cornell University, 1990. Available as TR90-1147. 16, 16, 16

[Sho71] J. R. Shoenfield. *Degrees of Unsolvability*. North-Holland, 1971. 18

[Sie93] K. Sieber. Call-by-value and nondeterminism. In *Proceedings of the Conference on Typed Lambda Calculus and its Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 376–390. Springer-Verlag, 1993. 12

[Sim93] A. K. Simpson. A characterisation of the least-fixed-point operator by dinaturality. *Theoretical Computer Science*, 118(2):301–314, September 1993. 184

[Smy78] M. B. Smyth. Power domains. *Journal of Computer and System Sciences*, 16(1):23–36, 1978. 44

[Spi89] J. M. Spivey. A categorical approach to the theory of lists. In J. L. A. van de Snepscheut, editor, *Mathematics of Program Construction*, volume 375 of *Lecture Notes in Computer Science*, pages 399–408. Springer-Verlag, 1989. 59

[Spi90] J. M. Spivey. A functional theory of exceptions. *Science of Computer Programming*, 14:25–42, 1990. 59

[SS92] H. Søndergaard and P. Sestoft. Non-determinism in functional languages. *Computer Journal*, 35(5):514–523, October 1992. 17

[Sta87] E. W. Stark. Concurrent transition system semantics of process networks. In *Conference Record of the 14th ACM Symposium on Principles of Programming Languages*, pages 199–210, 1987. 16

[Sta90] E. W. Stark. A simple generalisation of Kahn's principle to indeterminate dataflow networks. In M. Z. Kwiatkowska, M. W. Shields, and R. M. Thomas, editors, *Semantics for Concurrency, Leicester*, pages 157–176. Springer-Verlag, 1990. 16

[Sti97] C. Stirling. Bisimulation, model-checking and other games. Notes for Mathfit instructional meeting on games and computation, Edinburgh, June 1997. 9, 42

[Tho89] B. Thomsen. A calculus of higher order communicating systems. In *Conference Record of the 16th ACM Symposium on Principles of Programming Languages*, pages 143–154, 1989. 17

[Tho91] S. Thompson. *Type Theory and Functional Programming*. International Computer Science Series. Addison Wesley, 1991. 58

[Tho93] B. Thomsen. Plain CHOCS: A second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, January 1993. 17

[Tho95] B. Thomsen. A Theory of Higher Order Communication Systems. *Information and Computation*, 116(1):38–57, 1995. 17

[Tur90a] D. A. Turner. An approach to functional operating systems. In *Research Topics in Functional Programming* [Tur90b]. 16

[Tur90b] D. A. Turner, editor. *Research Topics in Functional Programming*. The UT Year of Programming Series. Addison-Wesley, 1990. 185, 198

[Van90] R. J. Van Glabbeek. The linear time – branching time spectrum. Report CS-R9029, CWI, 1990. 8, 39

[Van93] R. J. Van Glabbeek. The linear time — branching time spectrum II (the semantics of sequential systems with silent moves). In E. Best, editor, *Proceedings* CONCUR'93*, 4th International Conference on Concurrency Theory, Hildesheim, Germany*, volume 715, pages 66–81, 1993. 8, 39

[Van94] R. J. Van Glabbeek. What is branching time semantics and why to use it? In M. Nielsen, editor, *The Concurrency Column*, pages 191–198. *Bulletin of the EATCS 53*, 1994. 8

[Wad92] P. L. Wadler. Comprehending monads. *Mathematical Structures in Computer Science*, 2:461–493, 1992. 19, 59

[Wag94] E. G. Wagner. Algebraic semantics. In Abramsky et al. [AGM94]. 184

[Wal90] D. J. Walker. Bisimulation and divergence. *Information and Computation*, 85(2):202–241, 1990. 33, 39

[Whi94] G. T. Whitney. *Recursion Structures for Non-Determinism and Concurrency*. PhD thesis, University of California, Los Angeles, March 1994. 16

[Win93] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. Foundations of Computing Series. MIT Press, February 1993. 72

[WN95] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Semantic Modelling*, volume 4 of *Handbook of Logic in Computer Science*. Clarendon Press, 1995. 7, 32

Glossary of Symbols

Sets

ω . . . . natural numbers (including 0), 23
ω_1^{CK} . . . . least non-recursive ordinal, 46
ω_1 . . . . least non-countable ordinal
$A+B$ . . . . disjoint union of sets A and B
$A_\perp$ . . . . disjoint union of sets A and $\{\perp\}$
$Succ(A)$ . . . . . . successor of a set A, $Succ(A) = A \cup \{A\}$, 23
$Dom(f)$ . . . . . . domain of function f
$Im(f)$ . . . . image of function f
$\mathcal{P}(A)$ . . . . powerset of set A
$\mathcal{P}_{\mathrm{ne}}(A)$ . . . . set of all non-empty subsets of set A

Sequences

$\langle\rangle$ . . . . empty sequence
$\vec{a}$, $\langle a_0, a_1, \ldots, a_n\rangle$ . . . finite sequences
$\langle a_i \mid i \in I\rangle$ . . . . . indexed sequence
$\sqsubseteq$ . . . . prefix order
$\preceq_{\mathrm{LX}}$ . . . . lexicographic order, 46
$\preceq_{\mathrm{KB}}$ . . . . Kleene-Brouwer order, 46

Relations

$Id(A)$ . . . . identity or diagonal relation of set A, 127
$\mathcal{R}^*$ . . . . reflexive, transitive closure of relation $\mathcal{R}$
$\mathcal{R}^+$ . . . . transitive closure of relation $\mathcal{R}$
$\mathcal{R};\mathcal{S}$ . . . . diagrammatic composition of relations $\mathcal{R}$ and $\mathcal{S}$, 127
$\mathcal{R}^{\mathrm{op}}$ . . . . dual of relation $\mathcal{R}$, 127

Partial Orders

$\perp$ . . . . least element of partial order (or an urelement), 34
$\top$ . . . . greatest element of partial order, 34
$a \sqcap b$ . . . . binary meet of a and b, 34
$a \sqcup b$ . . . . binary join of a and b, 34
$\bigsqcap A$ . . . . meet of set A, 33
$\bigsqcup A$ . . . . join of set A, 33
$\overline{a}$ . . . . complement of a, 34
$\mu a.F(a)$ . . . . . . least fixed-point of function F, 34
$\nu a.F(a)$ . . . . . . greatest fixed-point of function F, 34

Trees

$Rank(a,<)$ . . . . . rank of a with respect to well-founded relation $<$, 27
$Len(A,\leq)$ . . . . . length of tree $\langle A,\leq\rangle$, 28
$Tree(A)$ . . . . . . $\in$-tree of set A, 27
$ST(\langle S,\rightarrow\rangle,s)$ . . . . synchronisation tree of transition system $\langle S,\rightarrow\rangle$ rooted at state s, 32

Transition Systems

$s \rightarrow t$ . . . . unlabelled transition from state s to state t, 30
$s \xrightarrow{a} t$ . . . . transition from state s to state t, labelled with a, 30, 86
$s \Uparrow^{\mathrm{may}}$ . . . . state s may diverge, 33, 86
$s \Downarrow^{\mathrm{must}}$ . . . . state s must converge, 33, 86
$TS(A)$ . . . . $\in$-transition system of set A, 31
$TSWD(A)$ . . . . . . $\in$-transition system with divergence of set A, 33

Typed Transition Systems

$\mathscr{L}_0$ . . . . typed transition system, 120
$\mathscr{L}_0(\mathcal{E})$ . . . . typed transition system, 120
$\mathscr{M}_0$ . . . . typed transition system, 147
$\mathscr{S}$ . . . . typed transition system, 99
$Total(\mathscr{T})$ . . . . . . hereditarily total elements of typed transition system $\mathscr{T}$, 87
$Det(\mathscr{T})$ . . . . . . hereditarily deterministic elements of typed transition system $\mathscr{T}$, 87
PR . . . . category of maps between typed transition systems, 109
$Ext(\mathscr{T})$ . . . . . . extensional collapse of typed transition system $\mathscr{T}$, 111
$\mathscr{T} \restriction X$ . . . . restriction of TTS $\mathscr{T}$ to set of states X, 112
$\xi(\sigma)$ . . . . choice function at type σ, 115

Simulation Operators

$\langle \mathcal{R} \rangle_{\mathrm{S}}$ . . . . simulation operator for transition systems, 40

$\langle \mathcal{R} \rangle_{\mathrm{LS}}$ . . . . lower simulation operator for transition systems with divergence, 43

$\langle \mathcal{R} \rangle_{\mathrm{US}}$ . . . . upper simulation operator for transition systems with divergence, 43

$\langle \mathcal{R} \rangle_{\mathrm{LS}}^{\mathscr{T}}$ . . . . lower simulation operator for typed transition system $\mathscr{T}$, 90

$\langle \mathcal{R} \rangle_{\mathrm{US}}^{\mathscr{T}}$ . . . . upper simulation operator for typed transition system $\mathscr{T}$, 90

Similarity and Bisimilarity for Transition Systems

$\lesssim_{\mathrm{S}}$ . . . . similarity, 40

$\simeq_{\mathrm{S}}$ . . . . mutual similarity, 40

$\simeq_{\mathrm{B}}$ . . . . bisimilarity, 40

Similarity and Bisimilarity for Transition Systems with Divergence

$\lesssim_{\mathrm{LS}}, \lesssim_{\mathrm{US}}, \lesssim_{\mathrm{CS}}, \lesssim_{\mathrm{RS}}$. lower, upper, convex, refinement similarity, 44

$\simeq_{\mathrm{LS}}, \simeq_{\mathrm{US}}, \simeq_{\mathrm{CS}}, \simeq_{\mathrm{RS}}$. lower, upper, convex, refinement mutual similarity, 44

$\simeq_{\mathrm{LB}}, \simeq_{\mathrm{UB}}, \simeq_{\mathrm{CB}}, \simeq_{\mathrm{RB}}$. lower, upper, convex, refinement bisimilarity, 44

Similarity and Bisimilarity for Typed Transition System $\mathscr{T}$

$\lesssim_{\mathrm{LS}}^{\mathscr{T}}, \lesssim_{\mathrm{US}}^{\mathscr{T}}, \lesssim_{\mathrm{CS}}^{\mathscr{T}}, \lesssim_{\mathrm{RS}}^{\mathscr{T}}$. lower, upper, convex, refinement similarity, 91

$\simeq_{\mathrm{LS}}^{\mathscr{T}}, \simeq_{\mathrm{US}}^{\mathscr{T}}, \simeq_{\mathrm{CS}}^{\mathscr{T}}, \simeq_{\mathrm{RS}}^{\mathscr{T}}$. lower, upper, convex, refinement mutual similarity, 91

$\simeq_{\mathrm{LB}}^{\mathscr{T}}, \simeq_{\mathrm{UB}}^{\mathscr{T}}, \simeq_{\mathrm{CB}}^{\mathscr{T}}, \simeq_{\mathrm{RB}}^{\mathscr{T}}$. lower, upper, convex, refinement bisimilarity, 91

Binary Choice Operators

$GAng(A,B)$ . . . . . global angelic choice of sets A and B, 50

$Amb(A,B)$ . . . . . ambiguous choice of sets A and B, 50

$Err(A,B)$ . . . . . . erratic choice of sets A and B, 50

$LDem(A,B)$ . . . . . local demonic choice of sets A and B, 50

$GDem(A,B)$ . . . . . global demonic choice of sets A and B, 50

Programming Language

$\mathscr{L}$ . . . . set of terms of programming language $\mathscr{L}$, 60
$\mathscr{L}_0$ . . . . set of programs of $\mathscr{L}$, 60
Can_0 . . . . set of canonical programs of $\mathscr{L}$, 60
$\mathscr{L}(\mathcal{E})$ . . . . set of terms of smallest fragment containing set of terms $\mathcal{E}$, 77
$\mathscr{L}_0(\mathcal{E})$ . . . . set of programs of smallest fragment containing set of terms $\mathcal{E}$, 77
$\mathscr{M}$ . . . . set of terms of fragment $\mathscr{M}$, 147
$\mathscr{M}_0$ . . . . set of programs of fragment $\mathscr{M}$, 147
$POrd(\sigma)$ . . . . . . P-order of type σ, 56
Var . . . . set of variables, 57
$Fv(M)$ . . . . free variables of term M, 57
$M[N_1,\ldots,N_n/x_1,\ldots,x_n]$ simultaneous substitution of terms $N_1,\ldots,N_n$ for variables $x_1,\ldots,x_n$, 58
$Red(\sigma)$ . . . . reducibility candidates of type σ, 76

Operational Semantics

$M \rightarrowtriangle_{\mathrm{det}} N$ . . . . . . term M reduces in one step to program N (deterministic), 65
$M \rightarrowtriangle N$ . . . . . . term M reduces in one step to program N, 65
$M \rightarrowtriangle^{\omega}$ . . . . term M diverges, 66
$M \not\downarrow x$ . . . . term M blocked at variable x, 66
$M[x \mapsto N]$ . . . . . . substitution of term N for occurrence of variable x blocking term M, 66
$M \Downarrow^{\mathrm{may}} K$ . . . . . . program M may converge to canonical program K, 72
$M \Uparrow^{\mathrm{may}}$ . . . . . . program M may diverge, 73
$M \Downarrow^{\mathrm{must}}$ . . . . . . program M must converge, 74
$M \Downarrow^{\mathrm{must}} A$ . . . . . program M has must convergence rank A, 80

Relations on Terms

$Rel_0(\mathcal{E})$ . . . . . . set of binary relations on programs of $\mathscr{L}(\mathcal{E})$, 123
$Rel(\mathcal{E})$ . . . . set of binary relations on terms of $\mathscr{L}(\mathcal{E})$, 127
$Cls(\mathcal{R})$ . . . . . . closed restriction of relation $\mathcal{R}$, 128
$Opn(\mathcal{E},\mathcal{R})$ . . . . . open extension of relation $\mathcal{R}$, 128
$\mathcal{R}[\mathcal{S}]$ . . . . relational substition of relation $\mathcal{S}$ into relation $\mathcal{R}$, 128
$Cmp(\mathcal{E},\mathcal{R})$ . . . . . compatible refinement of $\mathcal{R}$, 129
$Cand(\mathcal{E},\mathcal{R})$ . . . . congruence candidate of $\mathcal{R}$, 134

Index