SE580: Lecture 3

Overview

Dynamic semantics

Operational semantics

Substitution

Implementation

Homework

Dynamic semantics

What is dynamic semantics? How can we specify the dynamic semantics of a programming language?

What is an interpreter?

Operational semantics

Hobbes is specified using *operational semantics*.

The semantics is given as a collection of rules such as:

thread *a* { let *X* = *i* + *j*; *B* } *P*
–>
thread *a* { [*X* := *i*+*j*] *B* } *P*

Read this as:

Any program which contains a thread *a*
starting with let *X* = *i* + *j*; then executing block *B*

can become in one step

the same program, except that now thread *a*
is executing block *B* with *X* bound to the result of adding *i* and *j*

Operational semantics

For example:

```
  thread Main { let foo = 1+2; let bar = foo+foo; return Nothing; }
-->
  ????
```

Operational semantics

What rules do we need to add to the Hobbes specification to deal with this program?

```
  thread Main {
    let foo = 1 + 2;
    let bar = foo < 4;
    if (bar) { let baz = foo + 2; } else { let baz = foo + 5; }
  }
-->
  ????
```

Operational semantics

What rules do we need to add to the Hobbes specification to deal with this program?

```
  class Ref { field contents : Integer; }
  object Foo : Ref { contents = 5 }
  thread Main {
    let x = Foo.contents;
    let y = x + 1;
    Foo.contents := y;
  }
-->
  ????
```

Operational semantics

What rules do we need to add to the Hobbes specification to deal with this program?

```
  class Ref { field contents : Integer; }
  thread Main {
    let w = new Ref { contents = 5 }
    let x = w.contents;
    let y = x + 1;
    w.contents := y;
  }
-->
  ????
```

Operational semantics

We have (almost) dealt with all Hobbes straight-line code.

To-do list:

What does the substitution [$X := V$] B mean?

What do we do about method calls?

Substitution

We write [X := V] B for B with every *free occurrence* of variable X replaced by V.

For example, what is:

```
[ foo := 3 ] ( let bar = foo + foo; return Nothing; )
[ foo := 3 ] ( let bar = foo < 4; if (bar) { let baz = foo + 2; } else { let baz = foo + 5; } )
```

Slightly trickier, what is:

```
[ foo := 3 ] ( let foo = 1 + foo; let bar = 2 + foo; )
```

Substitution

Imagine we could write the following program:

```
class Ref { field contents : Integer; }
object foo : Ref { contents = 5 }
thread Main { let x = foo; let foo = "fred"; x.contents := 7; }
```

What would happen when we execute this program?

What can we do about it?

Substitution

Formal definition of a substitution σ:

```
Substitution ::=  Subst "," ... "," Subst
Subst ::= Var ":=" Val
```

For example, are the following valid substitutions?

```
[ x := 37 ]
[ x := Foo ]
[ x := 37, y := Foo ]
[ x := 37, x := Foo ]
[ ]
```

Substitution

Formal definition of $[\sigma]V$ (replacing any free variable X in V by $\sigma(X)$).

What is $[\sigma]i$ (i an integer)?

What is $[\sigma]a$ (a a global variable?)

What is $[\sigma]x$ (x a local variable)?

For example, what are the following?

```
[ x := 37, y := Foo ]5
[ x := 37, y := Foo ]Foo
[ x := 37, y := Foo ]x
[ x := 37, y := Foo ]y
[ x := 37, x := Foo ]x
```

Substitution

Formal definition of [σ]*B* (replacing any free variable *X* in *B* by σ(*X*)).

What is [σ](return *V*;)?

What is [σ](let *X* = *E*; *B*)?

For example, what are the following?

```
[ x := 37, y := Foo ] ( return x; )
[ x := 37, y := Foo ] ( let w = x + x; let v = x + w; )
[ x := 37, y := Foo ] ( let y = x + 1; let z = x + y; )
```

Note: we only define [σ]*B* for *closed* σ. What does this mean? Why?

Implementation

The Hobbes implementation follows the specification very closely.

How could we implement a Substitution class?

How could we implement a Step class (which performs one step of the operational semantics)?

Homework

Complete the interpreter for Hobbes (currently missing some of the cases for `Block`).

Same procedure as before!

Next week

Specification of static semantics.

Read the static semantic specification of Hobbes.