SE580: Lecture 2

Overview

Language design choices in Hobbes

Syntax

Grammars

Parse trees

Hobbes specification

Abstract syntax trees

Syntax sugar

Homework

Language design choices in Hobbes

Easy to parse (mostly LL(1), some LL(2)).

Explicitly typed.

No overloading or field shadowing.

No imperative variables, while loops, ...

Top-level `object` and `thread` declarations.

Lots of 'real-world' features missing (`private` annotations, constructors, packages,...)

Has a specification!

Hobbes is big enough to be interesting, but small enough to be usable in the classroom (only ~12000 loc).

Syntax

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

What is a parser? What is an abstract syntax tree?

Grammars

Hobbes is specified using *Extended Backus Naur Form* grammar.

What language is recognized by the following grammar: `Foo ::= "a"`

What about: `Foo ::= "a" "b"`

What about: `Foo ::= "a" | "b"`

What about: `Foo ::= "a" | "b" Foo`

What about: `Foo ::= ("a")*`

What about: `Foo ::= ("a")+`

What about: `Foo ::= ("a")?`

What about: `Foo ::= ['a'-'z']`

What about: `Foo ::= "a" ("," "a")*`

What about: `Foo ::= ( "a" ("," "a")* )?`

Parse trees

What is a parse tree?

Given the grammar:

```
Bar ::= Foo | "a" | "b"
Foo ::= "a" Bar "a" | "b" Bar "b"
```

What are the parse trees for the following Bars?

```
a
b
a a a
a b a
a b a b a
```

Hobbes specification: Identifiers

LocalId ::= Lower (Alpha | Digit | "_")*

GlobalId ::= Upper (Alpha | Digit | "_")*

Lower ::= [’a’-’z’]

Upper ::= [’A’-’Z’]

Digit ::= [’0’-’9’]

Alpha ::= Lower | Upper

For example, are these LocalId or GlobalIds?

aBC_DEFgh
Abc_defGH
_abc_def
ábc_dèf
$abcdef
abc-def

Hobbes specification: Values

Val ::= ValGlobal | ValLocal | ValInteger | ValString

ValGlobal ::= GlobalId

ValLocal ::= LocalId

ValInteger ::= *...an integer constant...*

ValString ::= *...a string constant...*

For example, are these Vals?

x
True
37
"abc-def"
abc-def
3.1415

Hobbes specification: Expressions

Exp ::= Val | ExpFieldAccess | ExpCall | *...other kinds of expression...*

ExpFieldAccess ::= Val "." LocalId

ExpCall ::= Val "." LocalId "(" Val "," ... "," Val) // synonym for (Val ("," Val)*)?

For example, are these Vals?

x.y
x.y()
x.y(a,b)
x.y.z
"abc".y
x."def"

Hobbes specification: Threads

DecThread ::= "thread" GlobalId "{" Block "}"

Block ::= BlockLet | BlockReturn | *...other kinds of block...*

BlockLet ::= "let" Var "=" Exp ";" Block

BlockReturn ::= "return" Val ";"

Var ::= LocalId (":" Type)?

Type ::= GlobalId

For example, are these DecThreads?

```
thread Main { return 3; }
thread Main { let x = 3; return x; }
thread Main { let x:Integer = 3; return x; }
thread Main { let x = 3; let y = x; return y; }
thread Main { let x:String = 3; return x; }
thread Main { let X = 3; return X; }
thread Main { let x = 3; }
```

Hobbes specification: Programs

Program ::= Dec ... Dec // synonym for Dec*

Dec ::= DecClass | DecObject | DecThread

DecClass ::= *...a class declaration...*

DecObject ::= *...an object declaration...*

Hobbes specification

The full syntactic specification is available on-line.

Lexers and parsers

What is a lexer (lexical analyser)?

What is a parser?

What is a parser generator?

For more information, take CSC448!

Abstract syntax trees

What is an AST?

How can we define an AST representation for a Hobbes Val?

Val ::= ValGlobal | ValLocal | ValInteger | ValString

ValGlobal ::= GlobalId

ValLocal ::= LocalId

ValInteger ::= *...an integer constant...*

ValString ::= *...a string constant...*

Abstract syntax trees

How can we define an AST representation for a Hobbes Block?

Block ::= BlockLet | BlockReturn | *...other kinds of block...*

BlockLet ::= "let" Var "=" Val ";" Block

BlockReturn ::= "return" Val ";"

Var ::= LocalId (":" Type)?

Type ::= GlobalId

Abstract syntax trees

How can we define an AST representation for a Hobbes Program?

Program ::= Dec ... Dec // synonym for Dec*

Dec ::= DecClass | DecObject | DecThread

DecClass ::= *...a class declaration...*

DecObject ::= *...an object declaration...*

DecThread ::= "thread" GlobalId "{" Block "}"

Abstract syntax trees

Good ideas for AST design...

1. Program immutably. (What does this mean? Why?)

2. Use visitors. (What does this mean? Why?)

3. Program to an interface, not an implementation. (What does this mean? Why?)

Syntax sugar

So far, we've only mentioned the Hobbes *core* language. There's also a *surface* language. For example:

```
thread Main { 1+2*3; }
```

is translated to:

```
thread Main {
  let tmp1 = 1.infix +(2);
  let tmp2 = tmp1.infix *(3);
  return Nothing;
}
```

Why do this?

Homework

Complete the AST for Hobbes (currently missing `DecObject` and `DecClass`.

Same procedure as before!

Next week

Specification of dynamic semantics.

Read the dynamic semantic specification of Hobbes.