



Principles of Programming Languages

Graham Hutton

Department of Computer Science

University of Nottingham

Principles of Programming Languages

Graham Hutton

Lecture 1 - Introduction

1.0.

This course takes a more abstract approach, focussing on some of the mathematical concepts underlying programming languages in general:

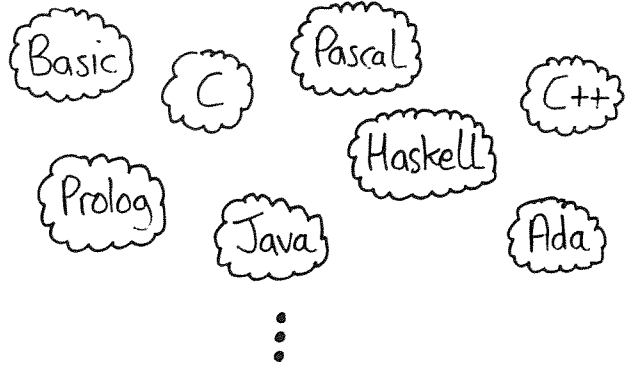
- Syntax;
- Operational semantics;
- Denotational semantics;
- Lambda calculus;
- Type theory;
- Domain theory.

The central topic of the course is semantics.

1.2.

BACKGROUND

There are many different programming languages.



Most courses focus on practical aspects of programming in particular languages, such as:

- Language features;
- Algorithms and data structures;
- Space and time efficiency;
- Design methods.

1.1

WHAT IS SEMANTICS?

Here are a few definitions:

- Oxford Minidictionary:
"Study of meaning";
- Collins Gem English Dictionary:
"Study of linguistic meaning";
- Webster's New Dictionary:
"Science of the meaning of words".

1.3

WHERE IS SEMANTICS STUDIED?

Here are the three main areas:

- Linguistics and Philosophy:

The meaning of sentences in a natural language (e.g. English);

- Mathematics:

The meaning of terms in a formal logic (e.g. predicate logic.);

- ★ Computing Science

The meaning of programs in a programming language (e.g. C.).

1.4.

WHY SEMANTICS MATTERS

Program semantics are useful in a number of areas of computing, including:

- Documentation

A formal semantics provides a complete and precise specification of a language for programmers and compiler writers.

- Formal reasoning

A formal semantics is a prerequisite for proofs about a programming language and programs written in it.

1.5.

- Language design

Semantic issues have a growing influence on the design of programming languages.

- Education

A knowledge of semantics gives a deeper understanding of programming languages, which will be useful when learning new languages, and when comparing different languages.

- Research

The elegance of semantics may inspire you to think about applying for a Ph.D. position in programming language research.

1.6.

COURSE TOPICS

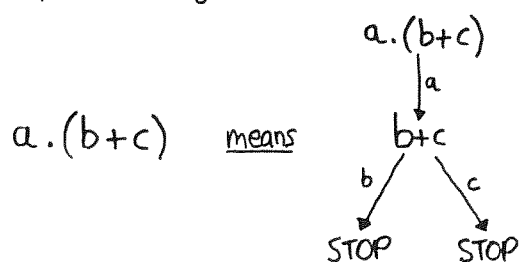
① Syntax

The syntactic structure of programs as trees is specified by a grammar, e.g.:



② Operational semantics

The meaning of programs as transition systems is specified by transition rules, e.g.:



1.7

③ Denotational semantics

The meaning of programs as mathematical functions is specified by recursion equations, e.g.:

$$x := 1 \quad \text{means} \quad f(s) = \text{update}(s, x, 1).$$

④ Lambda calculus

A theory of functions used to study many foundational issues in semantics, e.g.:

$$\text{true } x \ y = x$$

$$\text{false } x \ y = y$$

$$\text{not } x = x \ \text{false} \ \text{true}.$$

⑤ Domain theory

Solves the foundational problems with the semantics of recursive types and values, e.g.:

$$\text{data List} = \text{Nil} \mid \text{Cons Int List}$$

$$\text{len} :: \text{List} \rightarrow \text{Int}$$

$$\text{len Nil} = 0$$

$$\text{len (Cons } n \ ns) = 1 + \text{len } ns$$

COURSE MATERIAL

- Lecture notes: copies of my slides.
- Textbook: none required, but any book on the semantics, principles, or theory of programming languages will be useful for background reading.

ASSESSMENT

- Non-assessed exercises for each lecture;
- One written examination (100%).

HASKELL

Throughout the course, the language Haskell will be used for implementing semantic concepts.

- Haskell is well-suited to the kind of symbolic manipulation used in semantics;
- Implementing the concepts helps to make them easier to understand, and also allows them to be executed;
- Automatic type checking in Haskell helps to avoid mistakes in our definitions;
- Lecture 2 revises the basics of Haskell.

A TASTE OF SEMANTICS

In most languages, the Booleans (false and true) are built-in to the language.

Question: is this theoretically necessary?

Answer: no, they can be encoded in a simple way using functions!

The type of Booleans:

$$\text{type Bool} = a \rightarrow a \rightarrow a$$

The basic values:

true, false :: Bool

true x y = x

false x y = y

The "logical negation" function:

not :: Bool \rightarrow Bool

not x = x false true

Examples:

not false = false false true
= true

not true = true false true
= false

1.12.

1.13

EXERCISES

① In a similar manner to the not example, define a "logical and" function:

and :: Bool \rightarrow Bool \rightarrow Bool

② Show that your definition has the expected behaviour for all possible combinations of Boolean arguments.

③ Repeat both exercises above for the "logical or" function:

or :: Bool \rightarrow Bool \rightarrow Bool

1.14.

Principles of Programming Languages

Graham Hutton

Lecture 2 - Review of Haskell

INTRODUCTION

This course assumes some experience with the functional programming language Haskell.

For example, see:

Lecture notes in functional programming,
Graham Hutton, Univ. of Nottingham, 1997.

We'll be using the Hugs 1.3 version of Haskell, available for most systems from:

[http://www.cs.nott.ac.uk/
Department/Staff/mpj/hugs.html](http://www.cs.nott.ac.uk/Department/Staff/mpj/hugs.html).

This lecture reviews the basics of Haskell.

2.0

2.1

STARTING HUGS FROM UNIX

hugs Start Hugs;
hugs <filenames> Start and load files.

USEFUL COMMANDS WITHIN HUGS

:l <filenames> Load files;
:r Repeat last load;
:a <filenames> Load extra files;
:e <filename> Edit file;
:e Edit last file;
:t <expression> Type an expression;
:? List commands;
:q Quit

2.2

LAYOUT

In a sequence of definitions, each definition must begin in precisely the same column.

$a = b+c$ where $b=10$ $c=15-5$ $d = a*2$

means

$\{a = b+c$ where $\{b=10;$ $c=15-5$ $d = a*2 \}$

COMMENTS

-- A single-line comment.

{- A multiple-line comment,
which can be nested. -}

2.3

CONDITIONALS

$\text{abs } x = \text{if } x \geq 0 \text{ then } x \text{ else } -x$

GUARDS

$\text{signum } x \mid \begin{array}{l} x < 0 = -1 \\ x = 0 = 0 \\ x > 0 = 1 \end{array}$

PATTERN MATCHING

$\text{not False} = \text{True}$

$\text{not True} = \text{False}$

CASE ANALYSIS

$\text{not } x = \text{case } x \text{ of}$
 $\text{False} \rightarrow \text{True}$
 $\text{True} \rightarrow \text{False}$

2.4

BASIC TYPES

Bool Booleans: False True

Int Integers: ... -1 0 1 ...

Char Characters: ... 'a' 'b' ...

$a \ b \ c \ \dots$ Variables

COMPOUND TYPES

$[a]$ Lists

(a, b) Pairs

$a \rightarrow b$ Functions

2.6

LIST COMPREHENSIONS

$\text{pairs } xs = [(x, y) \mid x \leftarrow xs, y \leftarrow xs]$

$\text{positives } ns = [n \mid n \leftarrow ns, n > 0]$

$\text{concat } xss = [x \mid xs \leftarrow xss, x \leftarrow xs]$

ABBREVIATIONS

$f \ x \ y \ z$ means $((f \ x) \ y) \ z$

$a \rightarrow b \rightarrow c \rightarrow d$ means $a \rightarrow (b \rightarrow (c \rightarrow d))$

$[1, 2, 3, 4]$ means $1 : (2 : (3 : (4 : [])))$

"Hello" means $['H', 'e', 'l', 'l', 'o']$

2.5

TYPE DEFINITIONS

$\text{type String} = [\text{Char}]$

$\text{type Table } k \ v = [(k, v)]$

DATA DEFINITIONS

$\text{data Bool} = \text{False} \mid \text{True}$

$\text{data Result} = \text{Error} \mid \text{OK Int}$

$\text{data List } a = \text{Nil} \mid \text{Cons } a \ (\text{List } a)$

OVERLOADING

$(+)$:: $\text{Num } a \Rightarrow a \rightarrow a \rightarrow a$

$(=)$:: $\text{Eq } a \Rightarrow a \rightarrow a \rightarrow \text{Bool}$

2.7

RECURSION

fac :: Int → Int

fac 0 = 1

fac (n+1) = (n+1) * fac n

length :: [a] → Int

length [] = 0

length (x:xs) = 1 + length xs

flatten :: Tree a → [a]

flatten (Leaf x) = [x]

flatten (Node l r) = flatten l ++ flatten r

2.8

INPUT/OUTPUT

The library file "IO.hs" defines:

IO a - the type of interactive programs that return results of type a.

together with:

getChar :: IO Char

getLine :: IO String

putChar :: Char → IO ()

putStr :: String → IO ()

putStrLn :: String → IO ()

sequence :: [IO a] → IO ()

return :: a → IO a

2.9

Examples

getChar :: IO Char

getChar = do x ← getCh
putChar x
return x

putStr :: String → IO ()

putStr [] = return ()

putStr (x:xs) = do putChar x
putStr xs

getLine :: IO String

getLine = do x ← getChar
if x == '\n' then
return ""
else
do xs ← getLine
return (x:xs)

2.10

LIBRARY FUNCTIONS

For reference, here are some of the most commonly used library functions:

Booleans

(&&) :: Bool → Bool → Bool

(||) :: Bool → Bool → Bool

not :: Bool → Bool

Lists

(:) :: a → [a] → [a]

(++) :: [a] → [a] → [a]

head :: [a] → a

tail :: [a] → [a]

last :: [a] → a

init :: [a] → [a]

2.11

(!!) :: [a] → Int → a
 null :: [a] → Bool
 length :: [a] → Int
 elem :: Eq a ⇒ a → [a] → Bool
 reverse :: [a] → [a]
 zip :: [a] → [b] → [(a,b)]

Pairs

fst :: (a,b) → a
 snd :: (a,b) → b

Higher-order functions

(.) :: (b → c) → (a → b) → (a → c)
 map :: (a → b) → [a] → [b]
 filter :: (a → Bool) → [a] → [a]

EXAMPLE

Let us write a program for displaying a table showing the distribution of a list of integer exam marks in the range 0-99.

For example:

```
? showtable [57,68,41,61,...
0:
10: *
20:
30: ****
40: ****
50: ****
60: ****
70: ****
80: ***
90:
```

Displaying a mark:

showmark :: Int → IO ()
 showmark 0 = putStr "0"
 showmark n = putStr (show n)

Displaying a line of a table:

showline :: Int → [Int] → IO ()
 showline m ms =
 do showmark m
 putStr ": "
 putStrLn stars

where

stars = ['*' | _ ← [1..num]]
 num = length (filter valid ms)
 valid x = (m <= x) && (x <= m+9)

Displaying a table:

showtable :: [Int] → IO ()
 showtable ms = sequence [showline (m*10) ms / m | m ← [0..9]]

EXERCISES

① Modify the showtable program to also display the min, max, and average mark.

② Define as many of the library functions from slides 2.11 and 2.12 as you can, without looking up their definitions.

Hint: when testing your definitions in Hugs, you'll need to rename the functions to avoid clashing with the built-in definitions.

Principles of Programming Languages

Graham Hutton

Lecture 3 - Syntax I

3.0

INTRODUCTION

At the lowest level, a program can be viewed simply as a string of characters.

In order to study the meaning of programs at a higher level, we first need to study how programs are formed as strings:

meaning = semantics;

form = syntax.

The next two lectures introduce the basic mathematics underlying program syntax.

3.1

LOGIC

We begin by reviewing the operators of propositional (or Boolean) logic:

<u>Notation</u>	<u>Meaning</u>
$A \wedge B$	"and";
$A \vee B$	"or";
$\neg A$	"not";
$A \Rightarrow B$	"implies";
$A \Leftrightarrow B$	"equivalent to".

Note: the \Leftrightarrow operator is sometimes read as "if and only if", abbreviated by "iff".

3.2

Predicate logic extends propositional logic by introducing two quantifiers:

<u>Notation</u>	<u>Meaning</u>
$\forall x. P(x)$	"for all";
$\exists x. P(x)$	"there exists".

Examples

$\forall x. (\text{Man}(x) \Rightarrow \text{Person}(x))$ means
"Every man is a person";

$\exists x. (\text{Person}(x) \wedge \neg \text{Man}(x))$ means
"Not every person is a man".

3.3

SETS

A set is a collection of objects, such that:

- ① The order of objects is not important;
- ② Duplicate objects are not important.

The objects are usually called "elements" or "members".

<u>Notation</u>	<u>Meaning</u>
\emptyset	"empty set";
$\{a_1, a_2, \dots, a_n\}$	"explicitly defined set";
$\{x \mid P(x)\}$	"implicitly defined set";
$x \in S$	"is a member of";
$x \notin S$	"is not a member of".

34

STRINGS

A string over a finite set S is a finite sequence of elements of S , with duplicates permitted.

<u>Notation</u>	<u>Meaning</u>
ϵ	"empty string";
$a_1 a_2 \dots a_n$	"explicitly defined string";
$s \wedge t$	"concatenation".

The underlying set of a string is usually called the "alphabet" of the string.

Examples

1011 and 00101 are both strings over the alphabet $\{0,1\}$ of binary digits.

36

Useful operations on sets:

$$\text{Union} : S \cup T = \{x \mid x \in S \vee x \in T\}$$

$$\text{Intersection} : S \cap T = \{x \mid x \in S \wedge x \in T\}$$

$$\text{Subset} : S \subseteq T \Leftrightarrow \forall x. x \in S \Rightarrow x \in T$$

$$\text{Equality} : S = T \Leftrightarrow S \subseteq T \wedge T \subseteq S$$

$$\text{Product} : S \times T = \{(x,y) \mid x \in S \wedge y \in T\}$$

$$\text{Powerset} : P(S) = \{T \mid T \subseteq S\}$$

We can also extend \cup and \cap to sets of sets:

$$\bigcup X = \{x \mid \exists S. S \in X \wedge x \in S\}$$

$$\bigcap X = \{x \mid \forall S. S \in X \Rightarrow x \in S\}$$

35

Useful sets of strings

$$A^n : \text{All strings of length } n \text{ over the alphabet } A;$$

$$A^+ : \text{All non-empty strings over } A;$$

$$A^* : \text{ALL strings over } A.$$

Examples

$$\{0,1\}^0 = \{\epsilon\}$$

$$\{0,1\}^1 = \{0,1\}$$

$$\{0,1\}^2 = \{00,01,10,11\}$$

$$\{0,1\}^+ = \{0,1,00,01,10,11, \dots\}$$

$$\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$$

37