Lecture 11: Denotational Semantics (Part I)

Books

- “Denotational Semantics: a Methodology for Language Development”, by D.A. Schmidt, Allyn and Bacon, 1986.
- “The Structure of Typed Programming Languages”, by D.A. Schmidt, MIT Press, 1994.
- “The Formal Semantics of Programming Languages”, by G. Winskel, MIT Press, 1993.
- “Semantics of Programming Languages. Structures and Techniques”, by C.A. Gunther, MIT Press, 1992.

Semantics

The symbol sequence defined by the abstract syntax does not have meaning by itself.

Why is a formal semantics useful?

- A formal semantics is a contract between the language designer and implementor.
- The formal semantics should abstract over machine specific issues, to guarantee that programs preserve their meaning when moving to other architectures.
- A formal semantics is needed to justify that two programs produce the same result.

Styles of Semantics

Different styles of defining semantics:

- *Operational Semantics* specifies the meaning of a program in terms of an abstract interpreter.
- *Denotational Semantics* specifies the meaning of a program in terms of an abstract function mapping programs to values.
- *Axiomatic Semantics* specifies the meaning of a program via axioms and inference rules, like a mathematical logic.

The Basics of Denotational Semantics

In a denotational semantics the meaning of a program is defined as a mathematical function mapping language constructs into semantic domains.

Meaning = Function

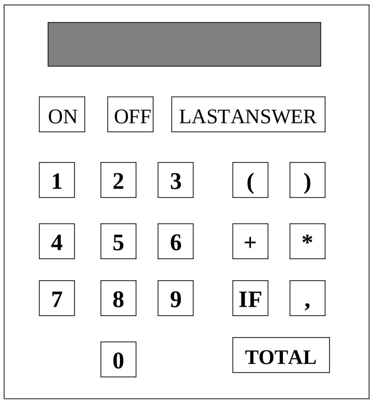
To model the meaning of languages with global variables, the semantics uses an additional argument: the state of the computation.

The denotational semantics of a language is very close to a functional program interpreting the language.

Example I: A Calculator

We want to define the semantics of a simple pocket calculator consisting of

- Output: display screen,
- Input: buttons (numbers, operators, and conditional),
- State: single memory cell.



An Example "Program"

Input	Display
`ON`	
(4 + 1 2) * 2	
`TOTAL`	32
1 + `LASTANSWER`	
`TOTAL`	33
`IF LASTANSWER` + 1 , 0 , 2 + 4	
`TOTAL`	6
`OFF`	

Roadmap of Defining a Denotational Semantics

1. Define the *abstract syntax* of the language.
2. Specify the *semantic domains* describing the values that can be produced.
3. Specify one or more *valuation functions* mapping program expressions to values.

The denotational semantics of a program is the result of applying the valuation function to the program.

Typically we write $\mathbf{E}[\![E]\!]e$ for the semantics of the expression E in the environment e. $\mathbf{E}$ is the valuation function. The funny brackets separate the "program" from the mathematical objects.

Step 1: Abstract Syntax

Typically we use some form of BNF notation.

$$
\begin{array}{ll}
P & \in \mathit{Program} \\
S & \in \mathit{ExprSeq} \\
E & \in \mathit{Expression} \\
N & \in \mathit{Numeral} \\
D & \in \mathit{Digit}
\end{array}
$$

$$
\begin{array}{ll}
P & ::= \texttt{ON}\ S \\
S & ::= E\ \texttt{TOTAL}\ S \mid E\ \texttt{TOTAL OFF} \\
E & ::= E_1 + E_2 \mid E_1 * E_2 \mid \texttt{IF}\ E_1\ ,\ E_2\ ,\ E_3 \mid \texttt{LASTANSWER} \mid (E) \mid N \\
N & ::= N\ D \mid D \\
D & ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

Step 2: Semantic Domains

Semantic domains are modeled as sets of values together with operators.

The specification of a semantic domain consists of

- a name;
- its values (typically specified as a set);
- its operators (typically specified as functions over this set).

Step 2: Semantic Domains

First we define the basic mathematical objects we need: natural numbers and boolean values.

Natural Numbers :

Domain : $\mathbb{N} = \{0, 1, \ldots\}$

Operators : $+, * \;::\; \mathbb{N} \to \mathbb{N} \to \mathbb{N}$

$== \;::\; \mathbb{N} \to \mathbb{N} \to \mathbb{B}$

Boolean Values :

Domain : $\mathbb{B} = \{tt, ff\}$

Operators : $not \;::\; \mathbb{B} \to \mathbb{B}$

$and, or \;::\; \mathbb{B} \to \mathbb{B} \to \mathbb{B}$

Step 3: Valuation Functions

For each non-terminal symbol in the abstract syntax we need one valuation function mapping it to the semantic domains.

The construct $\mathbb{N}^*$ denotes a finite list, i.e. a finite sequence of values from the set $\mathbb{N}$, with the operators $nil, cons, hd, tl$.

$$\mathbf{P} :: \texttt{Program} \to \mathbb{N}^*$$

$$\mathbf{P}\,[\![\texttt{ON}\ S]\!] \;=\; \mathbf{S}[\![S]\!]0$$

Note that the memory cell is initialised with the value 0.

Step 3: Valuation Function for `ExprSeq`

Typically a valuation function is defined as a case distinction over the structure of the program, with (recursive) calls to valuation functions for determining the semantics of the components.

$$\mathbf{S} :: \texttt{ExprSeq} \rightarrow \mathbb{N} \rightarrow \mathbb{N}^*$$
$$\mathbf{S}\,[\![E\ \texttt{TOTAL}\ S]\!]\ n \;=\; \mathit{let}\ n' = \mathbf{E}[\![E]\!]n\ \mathit{in}\ \mathit{cons}\ n'\ (\mathbf{S}[\![S]\!]n')$$
$$\mathbf{S}\,[\![E\ \texttt{TOTAL}\ \texttt{OFF}\ ]\!]\ n \;=\; \mathit{cons}\ (\mathbf{E}[\![E]\!]n)\ \mathit{nil}$$

The second argument to **S** specifies the "state" of the calculator. For this simple example the state is just the current value of the memory cell.

The sequence of values produced by the calculator is constructed by using the *cons* constructor of the semantic domain "Finite List".

Step 3: Valuation Functions for `Expression`

The meaning of an expression is a single $\mathbb{N}$ value.

$$\mathbf{E} :: \texttt{Expression} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$$
$$\mathbf{E}\,[\![E_1 + E_2]\!]\ n \;=\; \mathbf{E}\,[\![E_1]\!]\ n\ +\ \mathbf{E}\,[\![E_2]\!]\ n$$
$$\mathbf{E}\,[\![E_1 * E_2]\!]\ n \;=\; \mathbf{E}\,[\![E_1]\!]\ n\ *\ \mathbf{E}\,[\![E_2]\!]\ n$$
$$\mathbf{E}\,[\![\texttt{IF}\ \texttt{E}_1\ ,\ \texttt{E}_2\ ,\ \texttt{E}_3]\!]\ n \;=\; \mathit{if}\ \mathbf{E}\,[\![E_1]\!]\ n == 0\ \mathit{then}\ \mathbf{E}\,[\![E_2]\!]\ n\ \mathit{else}\ \mathbf{E}\,[\![E_3]\!]\ n$$
$$\mathbf{E}\,[\![\texttt{LASTANSWER}]\!]\ n \;=\; n$$
$$\mathbf{E}\,[\![(E)]\!]\ n \;=\; \mathbf{E}\,[\![E]\!]\ n$$
$$\mathbf{E}\,[\![N]\!]\ n \;=\; \mathbf{N}\,[\![N]\!]$$

NB: in the case of `LASTANSWER` we have to access the value of the memory cell.

Step 3: Valuation Functions for `Numerals`

The valuation functions for `Numerals` and `Digits` just maps (sequences of) digits to their corresponding mathematical values.

We use the valuation functions `mi` and `md` shown earlier in this course (Sec. 11.4 in the lecture notes).

$\mathbf{N} :: \texttt{Numeral} \rightarrow \mathbb{N}$

$\mathbf{N}\ [\![N\ D]\!] \ = \ 10 * \mathbf{N}\ [\![N]\!] \ + \mathbf{D}\ [\![D]\!]$

$\mathbf{N}\ [\![D]\!] \ = \ \mathbf{D}\ [\![D]\!]$

$\mathbf{D} :: \texttt{Digit} \rightarrow \mathbb{N}$

$\mathbf{D}\ [\![0]\!] \ = \ 0$

$\mathbf{D}\ [\![1]\!] \ = \ 1$

$\ldots$

Simplification

What is the meaning of the program:

```
ON 2+1 TOTAL IF LASTANSWER , 2 , 0 TOTAL OFF
```

$\mathbf{P}\ [\![\texttt{ON}\ 2+1\ \texttt{TOTAL IF LASTANSWER , 2 , 0 TOTAL OFF}]\!] \ =$

$\mathbf{S}\ [\![2+1\ \texttt{TOTAL IF LASTANSWER , 2 , 0 TOTAL OFF}]\!]\ 0 \ =$

$let\ n' \ = \ \mathbf{E}[\![2+1]\!]0\ in\ cons\ n'\ (\mathbf{S}\ [\![\texttt{IF LASTANSWER , 2 , 0 TOTAL OFF}]\!]\ n' =$

$let\ n' = 3\ in\ cons\ n'\ (\mathbf{S}\ [\![\texttt{IF LASTANSWER , 2 , 0 TOTAL OFF}]\!]\ n') =$

$cons\ 3\ (\mathbf{S}\ [\![\texttt{IF LASTANSWER , 2 , 0 TOTAL OFF}]\!]\ 3) =$

$cons\ 3\ (cons\ (\mathbf{E}\ [\![\texttt{IF LASTANSWER , 2 , 0 TOTAL}]\!]\ 3)\ nil) =$

Simplification (cont'd)

The IF expression denotes the following value:

$\mathbf{E}\,[\![\texttt{IF LASTANSWER , 2 , 0 TOTAL OFF}]\!]\;3 =$

$\mathit{if}\ \mathbf{E}\,[\![\texttt{LASTANSWER}]\!]\;3 == 0\ \mathit{then}\ \mathbf{E}\,[\![2]\!]\;3\ \mathit{else}\ \mathbf{E}\,[\![0]\!]\;3\ =$

$\mathit{if}\ 3 == 0\ \mathit{then}\ 2\ \mathit{else}\ 0$

$\mathit{if}\ \mathit{ff}\ \mathit{then}\ 2\ \mathit{else}\ 0$

0

When we use this value in the overall expression we get:

$\mathit{cons}\ 3\ (\mathit{cons}\ (\mathbf{E}\,[\![\texttt{IF LASTANSWER , 2 , 0 TOTAL}]\!]\;3)\ \mathit{nil} =$

$\mathit{cons}\ 3\ (\mathit{cons}\ 0\ \mathit{nil})$

Observations on the Simplification

Each of the simplification steps preserves the meaning of the program.

The goal is to find a "simpler" expression that denotes the same value as the original program.

The specification of the valuation functions are already very close to a functional program — an interpreter for this language.

In general, an *implementation* of the language consists of a *specification* (e.g. as a denotational semantics) and a *simplification strategy*.

Summary

In a denotational semantics the **meaning** of a program is defined as a **function**, mapping language constructs into semantic domains.

In order to define a denotational semantics you have to

1. Define the *abstract syntax* of the language.
2. Specify the *semantic domains* describing the values that can be produced.
3. Specify one or more *valuation functions* mapping program expressions to values.