PROGRAM SEMANTIC

www.cs.uu.nl/docs/vakken/pv

by Wishnu Prasetya (S.W.B.Prasetya@uu.nl)

FORMAL SEMANTIC

- To provide a formal definition, or at least a model, of what a progarm is. There are several common styles to do this.
- **Natural operational semantic**: defining the meaning of a program in terms of the result of its execution.
- **Structural** (small steps) operational semantic: defining how the program is executed.
- **Denotational semantic**: defining what a program is.
- **Axiomatic semantic**: defining how to infer properties of a program.

RUNNING EXAMPLE : LASG

- *program* → *block*
  block → { *identifier* = *expr* ; *statements* }
  statements → one or more stmt seprated by ";"
  stmt → *identifier* := *expr* | *block*

  expr → 0 | 1 | 2 | …
  | *identifier*
  | *expr* + *expr*

- A program produces/returns the final value of the (single) variable it declares at the root.

EXAMPLES

- A program that returns 11: {x=10; x:=x+1}

- A program with nested blocks, returning 21:

  {x=10; {y=x+10; {x=y ; y:=x+1} ; x:=y }}

OPERATIONAL SEMANTIC

- Notation: $\langle e,s\rangle \rightarrow v$

  "executing an expression *e* on the state *s* results in the value *v*"

- How do you want to represent a "state" ?
- Let's use list of "pairs" e.g. s = [$x \longmapsto 0$, $y \longmapsto 9$]
- Duplicate mapping is allowed, e.g. [$x \longmapsto 0$, $x \longmapsto 1$, $y \longmapsto 9$]
  The first mapping of *x* in s is also called the "latest" or the "most recent".

OPERATIONS ON STATE

- **query**, denoted by *s x* : querying the value of *the most recent mapping* of *x* in *s*. It is only defined if *s* contains *x*.

- **update** *s x v* : update *x*'s most recent mapping to *v*; it assumes that *x* is defined in *s*.

- **remove** *x s* : remove the most recent mapping of *x* in *s*.

A NATURAL OPERATIONAL SEMANTIC OF EXPR

- "executing" an expr results in a value.

- $\langle c,s \rangle \to c$
- $\langle x,s \rangle \to s\ x$, assuming s contains x
- $\langle e_1 + e_2 , s \rangle \to \langle e_1,s \rangle + \langle e_2,s \rangle$

- Example: $\langle x{+}1,s \rangle$ where s = [$x \longmapsto 0$, $x \longmapsto 1$, $y \longmapsto 9$]

AND FOR STMT (ASSIGNMENT FIRST)

- Executing a statement on a state results in a new state.

- For assignment :

$$\frac{\langle e,s \rangle \rightarrow v}{\langle x{:=}e,\ s \rangle \rightarrow \text{update } s\ x\ v}$$

OTHER STATEMENTS

- S_1 ; S_2 , can you define a semantic for this ?

- Block is a bit more involved:

$$\frac{\langle e,s\rangle \rightarrow v \qquad \langle S,\ x \mapsto v : s \rangle \rightarrow t}{\langle \{\ x=e\ ;\ S\ \}\ ,\ s\ \rangle \ \rightarrow\ \text{remove}\ x\ t}$$

FINALLY, THE SEMANTIC OF PROGRAM

- A program returns the final value of the variable it declares.
- Furthermore, any LAsg program starts from **the initial state**, which is [ ].
- Let P = { x=e ; S } be a **program**.

$$\dfrac{\langle e, [\,] \rangle \rightarrow v \qquad \langle S, [x \longmapsto v] \rangle \rightarrow t}{\langle P, [\,] \rangle \rightarrow t\,x}$$

STRUCTURAL OPERATIONAL SEMANTIC

- Structural, aka "small steps" semantic wants to describe the execution itself.
- Suppose we have an imaginary interpreter for LAsg, which "executes" an expr with the help of a stack. The following semantic describes how this works:
  - $\langle c,\sigma,s\rangle \;\Rightarrow\; \langle c{:}\sigma\;,\;s\rangle$
  - $\langle x,\sigma,s\rangle \;\Rightarrow\; \langle s\;x : \sigma\;,\;s\rangle$, assuming *s* contains *x*
  - $$\frac{\langle e_1,\sigma,s\rangle \;\Rightarrow\; \langle v_1 : \sigma\;,\;s\rangle \qquad \langle e_2\;,\;v_1{:}\sigma\;,s\rangle \;\Rightarrow\; \langle v_2{:}v_1{:}\sigma\;,s\rangle}{\langle e_1{+}e_2\;,\;\sigma\;,\;s\rangle \;\Rightarrow\; \langle v_1{+}v_2 : \sigma\;,\;s\rangle}$$

STMT (ASSIGNMENT FIRST)

- A stament may change the state.
  - It does not change the stack (it may temporarily change the stack, but will restore it to what it was).
  - For example, the semantic of assignement:

$$\frac{\langle e,\sigma,s\rangle \;\Rightarrow\; \langle v{:}\sigma\ ,\ s\rangle}{\langle x{:=}e\ ,\ \sigma\ ,\ s\rangle \;\Rightarrow\; \langle \sigma\ ,\ \text{update}\ s\ x\ v\rangle}$$

  - How about seq and block ?
  - How about the semantic of LAsg programs?

THE SEMANTIC OF A PROGRAM

- Let P = { x=e ; S } be a **program**.
- A program is executed on a fresh/empty stack and state.
- $$\frac{\langle e, [\,], [\,] \rangle \Rightarrow \langle [v], [\,] \rangle \qquad \langle S, [\,], [x \longmapsto v] \rangle \Rightarrow \langle [\,], t \rangle}{\langle P, [\,], [\,] \rangle \Rightarrow t\,x}$$

PROPERTIES

- Having formal semantics allows us to more precisely discuss about properties of programs, or of the semantics themselves, and to prove them.
- Examples:
  - *After executing an expression e, the stack will grow with exactly one:* $\langle e,\sigma,s\rangle \Rightarrow \langle \tau,s\rangle$ implies $|\tau| = |\sigma| + 1$
  - *The stack after executing any statemet S will be the same as it was at the start of the execusion.*
  - *The natural and structural semantics of LAsg's expr are equivalent*:

    For all σ : $\langle e,s \rangle \rightarrow v$ *if and only if* $\langle e,\sigma,s \rangle \Rightarrow \langle v{:}\sigma,s \rangle$

-

DENOTATIONAL SEMANTIC

- What is a program? Or at least, define a (mathematical) model of what a program is.
- **Notation**: $\mathcal{E}[\![e]\!]$ = the meaning of *e*
  - $\mathcal{E}$: expr → some domain
  - is called *valuation function*.
- We will need :
  - $\mathcal{E}$ for expr
  - $\mathcal{S}$ for statements
  - $\mathcal{P}$ for programs

DEFINING THE DOMAINS TO USE

- **State** = the domain of all possible states, each represented as a map as introduced before.
- **Int** = the domain of all integers.
- We will model an expr as a function of type **State** → **Int**.
- and a statement as a function of type **State** → **State**.

DENOTATIONAL SEM. OF EXPR

- $\mathcal{E}$: expr → (**State** → **Int**)

- $\mathcal{E}[\![c]\!] \;=\; (\lambda s.\ c)$
- $\mathcal{E}[\![x]\!] \;=\; (\lambda s.\ s\ x)$ ……… what if x is undefined in s ?
- $\mathcal{E}[\![e_1 + e_2]\!] = (\lambda s.\ \mathcal{E}[\![e_1]\!]s \;+ \mathcal{E}[\![e_2]\!]s)$

DENOTATIONAL SEM. OF STATEMENTS AND PROGRAMS

- $\mathcal{S}$: stmt → (**State** → **State**)

- For example, assignment:

  $\mathcal{S}[\![x := e]\!] \;=\; (\lambda s.\;\; \text{update}\;\; s\;\; \mathrm{x}\;\; (\mathcal{E}[\![e]\!]s)\;)$

- $\mathcal{P}$: program → **Int**

  $\mathcal{P}[\![\;\{\; x{=}e\;;\; \mathrm{S}\;\}\;]\!] \;=\; \textbf{let}\; \mathrm{t} = \mathcal{S}[\![\mathrm{S}]\!][\mathrm{x} \longmapsto \mathcal{E}[\![e]\!][\;]]\;\textbf{in}\;\mathrm{t}\;\mathrm{x}$

AXIOMATIC SEMANTIC

- Defining the meaning of a program in terms of its properties of interest.
  - More precisely, in terms of which properties hold (and implicitly, which do not).
  - Commonly formulated in terms of inference rules.

- Example: property of interest is the functional correctness of a program, in terms of Hoare triples.

SPECIFYING STATES

- We need a slightly richer language than expr. Call it **Pred** (predicates) :
  - $e_1 = e_2$, $e_1 > e_2$
  - $\neg P$, $P_1 \wedge P_2$, $P_1 \Rightarrow P_2$
  - $\forall x.\ P$

- Examples (on which states they hold ?) :
  - $x > 0$
  - $\forall x.\ x > 0 \Rightarrow \neg(x = 0)$

HOARE TRIPLES

- **Note**: as before, we limit our discussion to LAsg.
- Let P and Q be predicates.
- For statement: {* *P* *} *stmt* {* *Q* *}
- For LAsg program: {* *P* *} *prog* {* *Q* *}
  - *P* contains no free variable ; effectively the only sensical *P* is the predicate *true*.
  - *Q* only has "return" as its free variable

- Examples:
  - {* *x*>0 *} x:=x+1 {* x>1 *}
  - {* true *} {x=10 ; x:=x+1} {* return>10 *}

"AXIOMS" FOR STATEMENTS

- $P \Rightarrow Q[e/x]$ is valid

  $\{*P*\}\ x{:=}e\ \{*Q*\}$

- $\{*P*\}\ x'{:=}e\ ;\ x_{old}{:=}x\ ;\ x{:=}x'\ ;\ S\ ;\ x{:=}x_{old}\ \{*Q*\}$

  $\{*P*\}\ \{x{=}e\ ;\ S\ \}\ \{*Q*\}$

x' , x'', x_{old} are all fresh variables.

EXAMPLE

- {*true*}   {x:=2; y:=x−1}   {*¬y<0*}