dynamic semantics — for describing meanings of exprs, statements, program units, etc.

Park-note

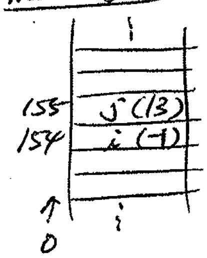
— Semantic domains for programming Languages

environment (γ) — set of pairs (var, mem location)
name addr.

memory (μ) — set of pairs (mem location, value)
addr.
(can be int, boolean, char values)
semantic domain
scope of values

locations (N) — natural numbers
addr.

ex) memory



$\gamma = \{ \langle i, 154 \rangle, \langle j, 155 \rangle \}$

$\mu = \{ \langle 0, \text{undefined} \rangle, \ldots \langle 154, -1 \rangle, \langle 155, 13 \rangle, \ldots \}$

— state transformation

— state of a program (σ) — set of pairs $\langle v, val \rangle$

sigma $\neq \gamma$ (gamma)

active var at that time

currently assigned value

ex) $\sigma = \{ \langle v1, val1 \rangle, \langle v2, val2 \rangle, \ldots \}$

$\sigma(v)$ — func. returning the current value of v.

ex) X=1 ; Y=2 ; Z=3 ;

current state — $\sigma = \{ \langle X, 1 \rangle, \langle Y, 2 \rangle, \langle Z, 3 \rangle \}$

$\sigma(Y) = 2$

→ Y=2*Z+3 ;

$\sigma' = \{ \langle X, 1 \rangle, \underline{\langle Y, 9 \rangle}, \langle Z, 3 \rangle \}$

→ W=4 ;

$\sigma'' = \{ \langle X, 1 \rangle, \langle Y, 9 \rangle, \langle Z, 3 \rangle, \underline{\langle W, 4 \rangle} \}$

— State transformation ($\overline{\cup}$) — overriding union (a kind of set operation)

ex) state of prog.
$\sigma_1 = \{ \langle x,1 \rangle, \langle y,2 \rangle, \langle z,3 \rangle \}$
$\sigma_2 = \{ \langle y,9 \rangle, \langle w,4 \rangle \}$

$\sigma_1 \overline{\cup} \sigma_2 = \{ \langle x,1 \rangle, \langle y,9 \rangle, \langle z,3 \rangle, \langle w,4 \rangle \}$ — transformed state (new state)

when union,
search for var's of σ_2 in σ_1,
for matched ones, replace values

— $\overline{\cup}$ represents the state after assign-st.
⇒ a formal model of assignment operation in a program.

semantics

id := expr ; ——— Semantics

Syntax — what it looks like   what it does.

used in operational semantics, & denotational ".

— 3 ways of defining semantics of programs

- operational semantics — execute statements, state change defines the meaning of the statements
- axiomatic "
- △ denotational " — the most abstract semantics description method. — complexity.

axiomatic
based on formal logic (predicate calculus)
formal program verification

Operational Semantics (of a program)

provides a definition of program meaning by simulating the program's behavior on a simple machine model (can be virtual machine)

ex) semantics of LISP on "SECD machine" — 1966

- Uses 2 notations

  - $\sigma(e) \Rightarrow v$ — computation of value v from expr. e in state σ.

    σ: current state, e: expr., v: value

    evaluation of expression e in σ

  - execution rules — $\dfrac{\text{premise}}{\text{conclusion}}$ (if premise is True, then conclusion also is True)

- Addition operation semantic is defined by:

$$\boxed{\dfrac{\sigma(e_1) \Rightarrow v_1, \quad \sigma(e_2) \Rightarrow v_2}{\sigma(e_1 + e_2) \Rightarrow v_1 + v_2}}$$

sub-expr. eval. first

ex) $\sigma = \{\cdots, \langle x, 5\rangle \cdots\}$

$$\dfrac{\sigma(x) \Rightarrow 5, \quad \sigma(1) \Rightarrow 1}{\sigma(x+1) \Rightarrow 6}$$

- Assignment statement (s.target = s.source) — right side eval. first.

  Semantic is defined by:

$$\boxed{\dfrac{\sigma(s.source) \Rightarrow v}{\sigma(s.target = s.source;) \Rightarrow \sigma \,\overline{\cup}\, \{\langle s.target, v\rangle\}}}$$

ex) $\sigma = \{\cdots, \langle x, 3\rangle, \cdots\}$

$x = 5;$

$\sigma' = \sigma \,\overline{\cup}\, \{\langle x, 5\rangle\} = \{\cdots, \langle x, 5\rangle, \cdots\}$

$$\dfrac{\sigma(5) \Rightarrow 5}{\sigma(\underbrace{x=5;}_{\text{assign st.}}) \Rightarrow \underbrace{\sigma \,\overline{\cup}\, \{\langle x, 5\rangle\}}_{= \{\ldots, \langle x, 5\rangle, \cdots\}}}$$

— Sequence of statements — ($S1 ; S2$)

$$\boxed{\frac{\sigma(S_1) \Rightarrow \sigma_1 , \quad \sigma_1(S_2) \Rightarrow \sigma_2}{\sigma(S1 ; S_2) \Rightarrow \sigma_2}}$$

ref
$\sigma(S_1) \Rightarrow \sigma_1$
executing S_1 in state σ
yields state σ_1

— conditionals — ($S =$ if ($\underline{S.test}$) $\underline{S.thenpart}$ else $\underline{S.elsepart}$)

2 exec. rules

1. True case:

$$\frac{\sigma(S.test) \Rightarrow \text{True}, \quad \sigma(S.thenpart) \Rightarrow \sigma_1}{\sigma(\text{if } (S.test)\ S.thenpart \text{ else } S.elsepart) \Rightarrow \sigma_1}$$

2. False case:

$$\frac{\sigma(S.test) \Rightarrow \text{False}, \quad \sigma(S.elsepart) \Rightarrow \sigma_1}{\sigma(\text{if } (S.test)\ S.thenpart \text{ else } S.elsepart) \Rightarrow \sigma_1}$$

ex) $\sigma = \{ \cdots, \langle X, 6 \rangle, \cdots \}$

```
if (X < 0)
    X = X - 1;
else
    X = X + 1;
```

new state σ_1

2nd rule

$$\frac{\sigma(X < 0) \Rightarrow \text{False}, \quad \sigma(X = X+1;) \Rightarrow \overbrace{\{ \cdots, \langle X, 7 \rangle \cdots \}}^{\sigma_1}}{\sigma(\text{if } (X<0)\ X = X-1; \text{ else } X = X+1) \Rightarrow \underbrace{\{ \cdots, \langle X, 7 \rangle, \cdots \}}_{\sigma_1}}$$

— loops — (S = while (s.test) s.body)

2 exec. rules

1. True case

recursively define

1st iteration; recursion (loops); final result state from loop

$$\frac{\sigma(s.test) \Rightarrow True, \quad \sigma(s.body) \Rightarrow \sigma_1, \quad \sigma_1(\text{while } (s.test)\ s.body) \Rightarrow \sigma_2}{\sigma(\text{while } (s.test)\ s.body) \Rightarrow \sigma_2}$$

2. False case

terminal case

$$\frac{\sigma(s.test) \Rightarrow False}{\sigma(\text{while } (s.test)\ s.body) \Rightarrow \sigma}$$

same state (no change)

ex) Assume $\sigma = \{\ldots, \langle x, 7\rangle, \ldots\}$

```
while (x<100)
   x = 2*x;
```

— apply rule 1 recursively.

$$\frac{\sigma(x<100) \Rightarrow true, \quad \sigma(x = 2*x;) \Rightarrow \overbrace{\{\ldots, \langle x, 14\rangle, \ldots\}}^{\sigma_1}, \quad \sigma_1(\text{while } (x<100)\ x=2*x;) \Rightarrow \sigma_2}{\sigma(\text{while } (x<100)\ x=2*x;) \Rightarrow \sigma_2}$$

⋮

finally,

$\sigma_2 = \{\ldots, \langle x, 112\rangle, \ldots\}$

value x:

7, 14, 28, 56, 112

σ, σ_1, σ_1', σ_1'', σ_2

Then,

$$\frac{\sigma(x<100) \Rightarrow False}{\sigma(\text{while } (x<100)\ x=2*x;) \Rightarrow \sigma_2}$$

$\{\ldots, \langle x, 112\rangle, \ldots\}$.

= Axiomatic semantics — uses proof rules

provides tools for proving the correctness of the program.

- running a program with all possible input values → nearly impossible task.
- ⇒ better way: predict the program's behavior.

- Assertion (predicate) — describes the state of a program at any point during execution.
- precondition / postcondition for each statement in the program.

(what must be True before statement for the postcondition is True.)

- partial proof for each statement ⇒ whole program proof.

ex) $\{P\}\ S\ \{Q\}$

(statement S is partially correct with respect to precondition P and postcondition Q.)

ex) $\{j=1\}\ j=j-1\ \{j=\emptyset\}$

- partial correctness proof rules

- Composition rule
- Conditional rule
- While rule
- rule of consequence
- Assignment Axiom

↓

— Composition rule

$$\frac{\{P\}\ S_1\ \{Q\},\quad \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1 ; S_2\ \{R\}}$$

— premise

— conclusion

ex)

$$\frac{\overbrace{\{j=1\}}^{P}\ j=j-1\ \overbrace{\{j=\phi\}}^{Q},\quad \overbrace{\{j=\phi\}}^{Q}\ j=j+2\ \overbrace{\{j=2\}}^{R}}{\underbrace{\{j=1\}}_{P}\ j=j-1 ; j=j+2\ \underbrace{\{j=2\}}_{R}}$$

— Conditional rule

$$\frac{\{P\wedge E\}\ S_1\ \{Q\},\quad \{P\wedge \neg E\}\ S_2\ \{Q\}}{\{P\}\ \text{if } E \text{ then } S_1 \text{ else } S_2\ \{Q\}}$$

we assume ∃ some P and E such that

Then, this is true

ex)

$P\wedge E : (j>\phi)$

$$\{\underbrace{(j\ge\phi)}_{P}\wedge\underbrace{(j>\phi)}_{E}\}\ \underbrace{j=j+1}_{S_1}\ \{\underbrace{(j>1)}_{Q}\}$$

$$\{\underbrace{(j\ge\phi)}_{P}\wedge\underbrace{(j\le\phi)}_{\neg E}\}\ \underbrace{j=j+2}_{S_2}\ \{\underbrace{(j>1)}_{Q}\}$$

$P\wedge\neg E : (j=\phi)$

$$\underbrace{\{j\ge\phi\}}_{P}\ \text{if}\ \underbrace{(j>\phi)}_{E}\ \text{then}\ \underbrace{j=j+1}_{S_1}\ \text{else}\ \underbrace{j=j+2}_{S_2}\ \underbrace{\{j>1\}}_{Q}$$

test with

$j=\phi \rightarrow 2$ ok

$j=1 \rightarrow 2$ ok

$j=2 \rightarrow 3$ ok

$j=3 \rightarrow 4$ ok

⋮

— while rule

$$\frac{\{p \wedge E\}\ S\ \{p\}}{\{p\}\ \text{while } E \text{ do } S\ \{p \wedge \neg E\}}$$

— if we assume this is True for S,

Then, we can conclude ~~that~~ ~~this is true~~ for while E do S.

ex) Consider code segment and complete all conditions.
(assertions)

⋮

$X = 5;$

$\{p\}$ — $X \geq \phi$

while $(X > \phi)$ do   ($= E$)

$\{p \wedge E\}$ — $(X \geq \phi) \wedge (X > 0) = (X > \phi)$

$X = X - 1;$ (S)

$\{p\}$ — $X \geq \phi$

$\{p \wedge \neg E\}$ — $(X \geq \phi) \wedge (X \leq \phi) = (X = \phi)$

premise

Conclusion

⇒ find a case (p) in which $\{p \wedge E\}\ S\ \{p\}$ holds since we assumed it (premise)

one possible p: $\{X \geq \phi\}$

$\Rightarrow\ p \wedge E:\ \{\underbrace{(X \geq \phi)}_{p} \wedge \underbrace{(X > \phi)}_{E}\} = (X > \phi)$

$p \wedge \neg E:\ \{\underbrace{(X \geq \phi)}_{p} \wedge \underbrace{(X \leq \phi)}_{\neg E}\} = (X = \phi)$

— Thus, if we assume

$\{(X \geq \phi) \wedge (X > \phi)\}\ X = X - 1\ \{X \geq \phi\}$

Then

$$\{X \geq \phi\}\ \text{while} (X > 0)\ \text{do}\ X = X - 1\ \underbrace{\{(X \geq \phi) \wedge (X \leq \phi)\}}_{=(X = \phi)}$$

test $X = 5$

loop
4
3
2
1
ϕ

$\underbrace{\{X = \phi\}}_{P}$ — ok.

Assigment Axiom — $X = E;$

$$\boxed{\frac{\text{True}}{\{Q[\underbrace{\text{target}}_{X} \backslash \underbrace{\text{source}}_{E}]\}\ S\ \{Q\}}}$$

— premise is always true (Axiom)

$\underbrace{X}_{\text{target}} = \underbrace{E}_{\text{source}};$ — value of E (source) before assignment becomes the value of X after the assignment.

$\exists$ only conclusion part

$$\boxed{\{Q[X \backslash E]\}\ X = E\ \{Q\}}$$

— if some condition Q for E (source), then same condition Q for X (target) after X=E.

eg) $\underbrace{\underbrace{\{i > \emptyset\}}_{Q[t|s]}}_{\text{precondition for } E \text{ (source)}}\ \underbrace{j}_{X} = \underbrace{i}_{E};\ \underbrace{\underbrace{\{j > \emptyset\}}_{Q}}_{\text{postcondition for } X \text{ (target)}}$

eg) $\underbrace{\{E > \emptyset\}}_{\text{precondition for source } (E)}\ X = E\ \underbrace{\{X > \emptyset\}}_{\text{postcondition for target } (X)}$

$>\emptyset$ $>\emptyset$

must be same

Rule of consequence

$p \supset p'$
$Q' \supset Q$

$$\left[\ \frac{P \text{ implies } P',\ \{P'\}\ S\ \{Q'\},\ Q' \text{ implies } Q}{\{P\}\ S\ \{Q\}}\right.$$

Axiomatic semantics