Fondamenti dei linguaggi di programmazione

Aniello Murano
Università degli Studi di Napoli
"Federico II"

Semantica Operazionale del linguaggio imperativo IMP

Introduzione

- Il linguaggio IMP è detto **imperativo** perché l'esecuzione di un programma comporta l'esecuzione esplicito di comandi che modificano lo stato
- IMP è **descritto da regole** che specificano come valutare le sue espressioni e come eseguire i suoi comandi.
- Tali regole forniscono una **semantica operazionale** di IMP

Sintassi di IMP

vars		Set		definition
m, n	$\in$	N		integers *(note: no need for syntax)*
t	$\in$	T	::=	$\{$ true, false $\}$
X, Y	$\in$	Loc	::=	$\{ x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n, \ldots \}$
a	$\in$	Aexp	::=	$n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$
b	$\in$	Bexp	::=	**true** $\mid$ **false** $\mid a_0 = a_1 \mid a_0 \leq a_1 \mid$ $\neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$
c	$\in$	Com	::=	**skip** $\mid X := a \mid c_0; c_1 \mid$ **if** b **then** c_0 **else** $c_1 \mid$ **while** b **do** c

La forma degli elementi di **Aexp**, **Bexp** e **Com** viene specificata tramite regole di formazione, espresse in una variante della BNF(Bakus-Naur Form)

Grammatica BNF (Bakus-Naur Form)

$$a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$$

- Rules for constructing a language
  - "::=" means "can be"
  - "|" means "or"
  - "a" is a *metavariable*
- Language = set of expressions
  - Set of arithmetic expressions: **Aexp**
- But BNF is a shorthand...

Regole di costruzione

- Short hand for rule

  if $a_0 \in$ Aexp *and* $a_1 \in$ Aexp *then* $a_0 + a_1 \in$ Aexp
- Describe as *inference rule*

$$\frac{a_0 \in \text{Aexp} \quad a_1 \in \text{Aexp}}{a_0 + a_1 \in \text{Aexp}} \qquad \frac{\text{premise}_1 \quad \ldots \text{premise}_n}{\text{conclusion}}$$

– sometimes read better from the bottom up

Due livelli

- *Rule Template*

$$\frac{a_0 \in \text{Aexp} \quad a_1 \in \text{Aexp}}{a_0 + a_1 \in \text{Aexp}}$$

- *Rule Instance*

$$\frac{y \in \text{Aexp} \quad 5 \in \text{Aexp}}{y + 5 \in \text{Aexp}}$$

- n, a_0, a_1, X are *metavariables*
  - the are used to define the language
  - not part of the language being defined

Regole per generare espressioni

- Axioms

$n \in \text{Aexp}$ $\qquad$ $X \in \text{Aexp}$

- Inference Rules

$$\frac{a_0 \in \text{Aexp} \quad a_1 \in \text{Aexp}}{a_0 + a_1 \in \text{Aexp}} \qquad \frac{a_0 \in \text{Aexp} \quad a_1 \in \text{Aexp}}{a_0 - a_1 \in \text{Aexp}} \qquad \frac{a_0 \in \text{Aexp} \quad a_1 \in \text{Aexp}}{a_0 \times a_1 \in \text{Aexp}}$$

$$a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$$

Tutto questo è sufficiente?

- Axioms + inference rules *generate* set Aexp
  - smallest set closed under application of rules
- Set is infinite
  - Is it well-defined, not victim to some paradox?
    - "Set of all sets that contain themselves"
  - Reasonable assumption for now
    - More details next class
- Same works for Booleans and commands...

Albero di derivazione

- Show how an expression is derived (a form of proof)

$$\dfrac{\dfrac{x \in \text{Aexp} \qquad \dfrac{y \in \text{Aexp} \quad 3 \in \text{Aexp}}{3 * y \in \text{Aexp}}}{x \leq (3 * y) \in \text{Bexp}} \qquad \dfrac{\dfrac{x \in \text{Aexp} \qquad y \in \text{Aexp}}{x \times y \in \text{Aexp}}}{x := x \times y \in \text{Com}}}{\textbf{while } x \leq (3 + y) \textbf{ do } x := x \times y \in \text{Com}}$$

a	$\in$	Aexp	::=	$n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$
b	$\in$	Bexp	::=	**true** | **false** | $a_0 = a_1 \mid a_0 \leq a_1 \mid$ $\neg b_0 \mid b_0 \wedge b_1 \mid b_0 \vee b_1$
c	$\in$	Com	::=	**skip** | $X := a \mid c_0; c_1 \mid$ **if** b **then** c_0 **else** $c_1 \mid$ **while** b **do** c

Semantica operazionale

- How do we define the behavior of a program?

```
while x ≠ y do
  if x < y then
    y := y – x
  else
    x := x – y
```

- Informally
  - If we know the value of all the variables (locations) we can evaluate expressions (Aexp and Bexp)
  - Commands cause changes to the variables (:=) or affect the flow of control
- Let's formalize this
  - First we need a way to represent values of locations

Stati

- State is a mapping of locations to values
  - $\sigma : \Sigma = Loc \rightarrow N$
  - $\sigma(X)$ is value of location X in state σ
  - We will consider *finite* states
    - function defined by a *graph*: a set of pairs
- Example
  - Loc = { x, y, z, ... }
  - σ = { (x, 3), (y, 99) }
- Then...
  - $\sigma(x) = 3$
  - $\sigma(y) = 99$
  - $\sigma(z) =$ *undefined*

Valutazione delle espressioni

- Aexp si valuta in interi, rispetto ad un dato stato
- Con **<a, σ>** denotiamo una espressione aritmetica **a** che deve essere valutata nello stato σ
  - La coppia **<a, σ>** è una ***configurazione***
- Per dire che l'espressione a valutata nello stato σ si ridue a n usiamo

$$\langle a, \sigma \rangle \rightarrow n$$

  - Il simbolo " → " è una ***relazione di transizione***
- Specifica il comportamento di una ***macchina astratta***:
  - Quando forniamo in input alla macchina una coppia espressione (**a**) stato (**σ**), la macchina da in output il valore n
  - Questo può essere pensato come una ***transizione*** da una configurazione a un valore finale.

Terminologia

- The symbol $\rightsquigarrow$ is a *relation*

  $\rightsquigarrow \subseteq \text{Aexp} \times \Sigma \times \text{N}$

  - Remember "x R y" means $(x, y) \in R$
  - choice of symbol $\rightsquigarrow$ is arbitrary
- Relationship between syntax and semantics
  - Aexp is *syntactic domain*, N is *semantic domain*
- Next: specific cases that *define* this relation
  - A case of each rule in that constructs Aexp

$$a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$$

Semantica operazionale di Aexp (1)

- Numbers
  $\langle n, \sigma \rangle \rightsquigarrow n$
- Examples
  $\langle 1, \emptyset \rangle \rightsquigarrow 1$
  $\langle 99, \sigma \rangle \rightsquigarrow 99$ where $\sigma = \{ (x, 3), (y, 99) \}$
  $\langle 99, \sigma \rangle \rightsquigarrow 99$ for *all* σ
  $\rightsquigarrow$ contains a triples of this form for every store σ in Σ
- Remember: $\rightsquigarrow$ is just a set of triples:
  - $\langle 1, \emptyset, 1 \rangle \in \rightsquigarrow$

Semantica operazionale di Aexp (2)

- Locations
  $\langle X, \sigma \rangle \rightsquigarrow \sigma(X)$
- Examples
  $\langle x, \{ (x, 3) \} \rangle \rightsquigarrow 3$
  $\langle y, \{ (x, 3), (y, 99) \} \rangle \rightsquigarrow 99$
  $\langle z, \{ \ldots, (z, n), \ldots \} \rangle \rightsquigarrow n$
- What about
  $\langle x, \emptyset \rangle \rightsquigarrow ???$
  $\rightsquigarrow$ does not contain any triples of the form $\langle X, \emptyset, ??? \rangle$
  Only valid programs are defined by transitions to integers

Semantica operazionale di Aexp (3)

- Addition

$$\frac{\langle a_1, \sigma\rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma\rangle \rightsquigarrow n_2}{\langle a_1 + a_2, \sigma\rangle \rightsquigarrow n} \quad \textit{where } n \textit{ is the sum of } n_1 \textit{ and } n_2$$

- Example

$\langle 99+x\ ,\ \{(x,3)\}\rangle \rightsquigarrow 102$

- Because

$\langle 99\ ,\ \{(x,3)\}\rangle \rightsquigarrow 99$

$\langle x\ ,\ \{(x,3)\}\rangle \rightsquigarrow 3$

Semantica operazionale di Aexp (4)

$$\langle n, \sigma\rangle \rightsquigarrow n \quad \text{[Const]}$$

$$\langle X, \sigma\rangle \rightsquigarrow \sigma(X) \quad \text{[Loc]}$$

$$\frac{\langle a_1, \sigma\rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma\rangle \rightsquigarrow n_2}{\langle a_1 + a_2, \sigma\rangle \rightsquigarrow n} \quad \text{[Sum]}$$

where n *is the sum of* n_1 *and* n_2

$$\frac{\langle a_1, \sigma\rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma\rangle \rightsquigarrow n_2}{\langle a_1 - a_2, \sigma\rangle \rightsquigarrow n} \quad \text{[Sub]}$$

where n *is the result of subtracting* n_2 *from* n_1

$$\frac{\langle a_1, \sigma\rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma\rangle \rightsquigarrow n_2}{\langle a_1 \times a_2, \sigma\rangle \rightsquigarrow n} \quad \text{[Prod]}$$

where n *is the product of* n_1 *and* n_2

Interpretazione delle regole

- Ogni regola di valutazione ha una premessa (scritta sopra la linea) e una conclusione (scritta sotto la linea)
- Siccome le regole specificano il significato delle espressioni in modo operazionale, si dice che esse definiscono una ***semantica operazionale*** di tali espressioni
- Alcune regole non hanno premesse. Queste regole, vengono anche chiamate ***assiomi*** come la regola seguente

$$\frac{}{\langle n,\sigma \rangle \rightarrow n}$$

Albero di derivazione

- Sia $a \equiv (Init + 5) + (7 + 9)$ nello stato σ_0
- Init una locazione tale che $\sigma_0(init)=0$

$$\dfrac{\dfrac{\dfrac{}{\langle Init,\sigma_0\rangle \rightarrow 0} \quad \dfrac{}{\langle 5,\sigma_0\rangle \rightarrow 5}}{\langle Init + 5,\sigma_0\rangle \rightarrow 5} \quad \dfrac{\dfrac{}{\langle 7,\sigma_0\rangle \rightarrow 7} \quad \dfrac{}{\langle 9,\sigma_0\rangle \rightarrow 9}}{\langle 7 + 9,\sigma_0\rangle \rightarrow 16}}{\langle (Init + 5) + (7 + 9),\sigma_0\rangle \rightarrow 21}$$

- Tale struttura viene detta ***albero di derivazione***
- La conclusione della derivazione si chiama **derivata**
- Si noti come le regole forniscono anche un ***algoritmo*** per la valutazione di espressioni aritmetiche basato sulla ricerca di un albero di derivazione.

Equivalenza in Aexp

- We say that $a_1 \sim a_2$ if and only if a_1 and a_2 evaluate to the same value in all states

$$a_1 \sim a_2 \Leftrightarrow \forall n \in N. \forall \sigma \in \Sigma. \langle a_1, \sigma \rangle \rightsquigarrow n \Leftrightarrow \langle a_2, \sigma \rangle \rightsquigarrow n$$

Intermissione

- Summary
  - We have defined the valuate of arithmetic expressions
  - Defining a *transition relation* that relates abstract syntax (in context) to values
- Next: Boolean expressions

$$b ::= \mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$$

Semantica operazionale di Bexp(1)

◆True and false

$\langle \mathbf{true}, \sigma \rangle \rightsquigarrow$ true

$\langle \mathbf{false}, \sigma \rangle \rightsquigarrow$ false

◆Note

- The **true** on the left is syntax, while true on the right is the element of the set of truth values T

Semantica operazionale di Bexp(2)

◆Comparisons

$$\frac{\langle a_1, \sigma \rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma \rangle \rightsquigarrow n_2}{\langle a_1 = a_2, \sigma \rangle \rightsquigarrow t}$$

where t is true *if* n_1 *is equal to* n_2 *and* false *otherwise*

$\langle a_1 \leq a_2, \sigma \rangle \rightsquigarrow t$ *where t is* true *if* n_1 *is less than or equal to* n_2 *and* false *otherwise*

– Shorthand: allow two conclusions for a set of premises

Semantica operazionale di Bexp(3)

◆Negation

$$\frac{\langle b, \sigma\rangle \rightsquigarrow \text{true}}{\langle \neg b, \sigma\rangle \rightsquigarrow \text{false}} \qquad \frac{\langle b, \sigma\rangle \rightsquigarrow \text{false}}{\langle \neg b, \sigma\rangle \rightsquigarrow \text{true}}$$

Semantica operazionale di Bexp(4)

◆And (Or is similar...)

$$\frac{\langle b_0, \sigma\rangle \rightsquigarrow \text{false}}{\langle b_0 \wedge b_1, \sigma\rangle \rightsquigarrow \text{false}} \qquad \frac{\langle b_0, \sigma\rangle \rightsquigarrow \text{true} \quad \langle b_1, \sigma\rangle \rightsquigarrow \text{true}}{\langle b_0 \wedge b_1, \sigma\rangle \rightsquigarrow \text{true}}$$

$$\frac{\langle b_0, \sigma\rangle \rightsquigarrow \text{true} \quad \langle b_1, \sigma\rangle \rightsquigarrow \text{false}}{\langle b_0 \wedge b_1, \sigma\rangle \rightsquigarrow \text{false}}$$

◆Note

- There are only three cases. Why?
- Any unconstrained variable can take any value
  - Example is b_1 in first inference rule

Intermissione

◆Summary
- We have defined the valuate of arithmetic and Boolean expressions
- Defining *transition relations* that relate abstract syntax and stores to values

◆There are three different relations

$\rightsquigarrow_{Aexp} \subseteq Aexp \times \Sigma \times N$

$\rightsquigarrow_{Bexp} \subseteq Bexp \times \Sigma \times T$

$\rightsquigarrow_{Com} \subseteq Com \times \Sigma \times \Sigma$

◆But we write them without subscripts, as $\rightsquigarrow$

– Distinguish them by context

Comandi

- Valutazione: Il ruolo dei programmi (e quindi dei comandi) è quello di essere eseguiti per **cambiare lo stato**.
- Quando si esegue un programma IMP, sia assume che lo stato (iniziale σ_0) associ valore 0 ad ogni locazione ("variabile"). **In pratica $\sigma_0(X)=0$**. Successivamente l'esecuzione può terminare in uno **stato finale** oppure **divergere**
- $\langle c, \sigma \rangle$ denota una ***configurazione di comando*** e denota la possibilità di eseguire il comando c a partire dallo stato σ
- La valutazione di un comando è formalmente definita da una funzione da un comando e uno stato ad un nuovo stato.

$$\langle c, \sigma \rangle \rightarrow \sigma'$$

$c ::= \mathbf{skip} \mid X := a \mid c_0; c_1 \mid$
$\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ c$

Semantica Operazionale di Com (1)

◆Skip

$\langle \mathbf{skip}, \sigma \rangle \rightsquigarrow \sigma$

◆Assignment

$$\frac{\langle a, \sigma \rangle \rightsquigarrow n}{\langle X := a, \sigma \rangle \rightsquigarrow \sigma'}$$

where σ' is σ updated to have n in location X

– Need a notation for updating state

Per esempio, < X:=5, σ > → σ', indica che lo stato σ' si ottiene dallo stato σ, aggiornandolo in modo che la locazione X contenga il valore 5

Semantica Operazonale di Com (2)

◆σ' is σ updated to have n in location X

- $\sigma' = \sigma[n/X]$

◆Change a value of a function for one input

$$\sigma[m/X](Y) = \begin{cases} m & \text{if } Y = X \\ \sigma(X) & \text{otherwise} \end{cases}$$

◆Final rule for assignment:

$$\frac{\langle a, \sigma \rangle \rightsquigarrow n}{\langle X := a, \sigma \rangle \rightsquigarrow \sigma[n/X]}$$

Osservazione

- Does the assignment rule say anything useful?

$$\frac{\langle a, \sigma\rangle \rightsquigarrow n}{\langle X := a, \sigma\rangle \rightsquigarrow \sigma[n/X]}$$

- It tells us that
  - a is evaluated in the store before the assignment takes place
  - No side effects: the only thing changed in the store is the value of X after a is evaluated
  - oh, and this language does not allow Booleans to be stored in variables

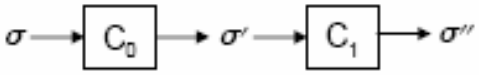
Con questa notazione si può scrivere
$\langle X:=5,\sigma\rangle = \sigma[5/X]$

Semantica Operazonale di Com (3)

- Sequence

$$\frac{\langle c_0, \sigma\rangle \rightsquigarrow \sigma' \qquad \langle c_1, \sigma'\rangle \rightsquigarrow \sigma''}{\langle c_0; c_1, \sigma\rangle \rightsquigarrow \sigma''}$$

- Order of evaluation is defined by the use of the store:



Semantica Operazonale di Com (4)

- Conditionals

$$\frac{\langle b, \sigma\rangle \rightsquigarrow \text{true} \qquad \langle c_0, \sigma\rangle \rightsquigarrow \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \rightsquigarrow \sigma'}$$

$$\frac{\langle b, \sigma\rangle \rightsquigarrow \text{false} \qquad \langle c_1, \sigma\rangle \rightsquigarrow \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \rightsquigarrow \sigma'}$$

- Outcome of Boolean determines which branch is executed

Semantica Operazonale di Com (5)

- While loop

$$\frac{\langle b, \sigma\rangle \rightsquigarrow \text{false}}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \rightsquigarrow \sigma}$$

$$\frac{\langle b, \sigma\rangle \rightsquigarrow \text{true} \qquad \langle c, \sigma\rangle \rightsquigarrow \sigma' \qquad \langle \textbf{while } b \textbf{ do } c, \sigma'\rangle \rightsquigarrow \sigma''}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \rightsquigarrow \sigma''}$$

- Defined in *terms of itself*
  - *Need to ensure that this makes sense*

Equivalenza in Com (1)

- We say that $c_1 \sim c_2$ if and only if c_1 and c_2 evaluate to the same state when started in the same state

$$c_1 \sim c_2 \Leftrightarrow \forall \sigma,\sigma' \in \Sigma.\ \langle c_1, \sigma \rangle \rightsquigarrow \sigma' \Leftrightarrow \langle c_2, \sigma \rangle \rightsquigarrow \sigma'$$

Esercizio

- Show $w \sim w'$ where
  - $w \equiv$ **while** b **do** c
  - $w' \equiv$ **if** b **then** c; w **else skip**
    - Note: $=$ is syntactic equivalence
- That is,

$$w \sim w' \Leftrightarrow \forall \sigma,\sigma' \in \Sigma.\ \langle w, \sigma \rangle \rightsquigarrow \sigma' \Leftrightarrow \langle w', \sigma \rangle \rightsquigarrow \sigma'$$

- Intuition of proof:
  - Given a derivation of $\langle w, \sigma \rangle \rightsquigarrow \sigma'$ we can construct a derivation of $\langle w', \sigma \rangle \rightsquigarrow \sigma'$ (and vice-versa)