Programmi deterministici

In un **programma deterministico** o **sequenziale**, in ogni istante c'è al più una prossima istruzione da eseguire. Da ogni **stato iniziale** si genera un'**unica** sequenza di stati durante l'esecuzione.

Studieremo un linguaggio per programmi deterministici:

- **sintassi**;
- **semantica operazionale**;
- **semantica assiomatica**: sistemi di **regole di Hoare** per dimostrare **correttezza parziale** e **correttezza totale** di programmi;
- **correttezza** della semantica assiomatica rispetto alla semantica operazionale;
- **completezza** della semantica assiomatica rispetto alla semantica operazionale.

SINTASSI

di un linguaggio di programmazione **imperativo tipato**

Sintassi dei programmi

$$S \ ::= \ \mathit{skip} \mid u := t \mid S_1; S_2 \mid \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \mid \textbf{while } B \textbf{ do } S_1 \textbf{ od} \ ,$$

dove u è una variabile semplice o sottoscritta, $t \in \mathit{Expr}$, $B \in \mathit{boolExpr}$.

Consideriamo solo programmi "ben tipati", cioè **sintatticamente corretti** rispetto ai tipi.

Tipi

- *Tipi di base*:
  - **int**
  - **bool**
  - **char**

- *Tipi di ordine superiore* (funzioni): per ogni $n \geq 1$,
  - $T_1 \times \ldots \times T_n \rightarrow T$, dove $T_1, \ldots, T_n, T$ sono tipi di base.
    $T_1, \ldots, T_n$ sono i **tipi argomento**.
    T è il **tipo valore**.

Variabili

- **Variabili semplici**: di tipo base.

- **Variabili array**: di tipo superiore.

  E.g. un array di tipo **int** $\rightarrow T$, è un array i cui elementi hanno tipo T.

L'insieme delle variabili è chiamato *Var*.

N.B. Gli array sono infiniti a priori. Se a è un array di tipo **int** $\rightarrow T$, la **sezione** $a[k : l]$ denota la restrizione di a all'intervallo $\{i \mid k \leq i \leq l\}$.

Costanti

- Costanti di tipo base:
  - **costanti intere**: $0, -1, 1, 2, -2, \ldots$
  - **costanti booleane**: **true**, **false**.

- Costanti di tipo superiore:
  - **simboli di relazione**: $T_1 \times \ldots \times T_n \rightarrow$ **bool**;
  - **simboli di funzione**: $T_1 \times \ldots \times T_n \rightarrow$ **int**.

  n è l'arietà della costante.

L'insieme delle costanti è chiamato *Const*.

Alcune costanti di ordine superiore:

- $|\ |$: **int** $\rightarrow$ **int**;
- $+, -, \cdot, min, max, div, mod$: **int** $\times$ **int** $\rightarrow$ **int**;
- $=_{int}, <, divides$: **int** $\times$ **int** $\rightarrow$ **boolean**;
- $\neg$: **bool** $\rightarrow$ **bool**;
- $=_{bool}, \vee, \wedge, \rightarrow, \leftrightarrow$: **bool** $\rightarrow$ **bool**.

Espressioni (di tipo base)

Le **espressioni** sono definite per **induzione**:

- una variabile semplice di tipo T è un'espressione di tipo T;
- una costante di tipo base è un'espressione di tipo T;
- se $s_1, \ldots, s_n$ sono espressioni di tipo $T_1, \ldots, T_n$ e *op* è una costante di tipo $T_1 \times \ldots \times T_n \rightarrow T$, allora $op(s_1, \ldots, s_n)$ è una espressione di tipo T;
- se $s_1, \ldots, s_n$ sono espressioni di tipo $T_1, \ldots, T_n$ e a è un array di tipo $T_1 \times \ldots \times T_n \rightarrow T$, allora $a[s_1, \ldots, s_n]$ è una espressione di tipo T (**variabile sottoscritta**);
- se B è una espressione booleana e s_1, s_2 sono espressioni di tipo T, allora $\mathbf{if}\ B\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2\ \mathbf{fi}$ è un'espressione di tipo T.

Variabili sottoscritte

Le **variabili sottoscritte** sono espressioni del tipo $a[s_1, \ldots, s_n]$, dove a è un array.

Sono chiamate **variabili** (anche se non sono propriamente variabili), in quanto è possibile assegnare ad esse un valore (tramite il **comando di assegnamento**). Inoltre ad esse si applica la **sostituzione**.

Semantica delle espressioni: dominio semantico

Dominio semantico:

$$\mathcal{D} = \bigcup_{T \text{ is a type}} \mathcal{D}_T \ ,$$

dove $\mathcal{D}_T$ è definito induttivamente:

- $\mathcal{D}_{int} = \{\ldots, -1, 0, 1, \ldots\}$
- $\mathcal{D}_{bool} = \{\mathbf{true}, \mathbf{false}\}$
- $\mathcal{D}_{T_1 \times \ldots \times T_n \to T} = [\mathcal{D}_{T_1} \times \ldots \times \mathcal{D}_{T_n} \to \mathcal{D}_T]$, i.e. l'insieme delle funzioni da $\mathcal{D}_{T_1} \times \ldots \times \mathcal{D}_{T_n}$ in $\mathcal{D}_T$.

Semantica delle costanti

La **funzione di interpretazione**

$$\mathcal{I} : Const \rightarrow \mathcal{D}_T$$

è definita come segue:

- se c è una costante di tipo base, allora $\mathcal{I}(c) = c$;
- le costanti di tipo superiore sono interpretate da funzioni:
  - $\mathcal{I}(|\ |)$ è la funzione valore assoluto;
  - $\mathcal{I}(\neg)$ è la funzione negazione di valori booleani;
  - ...

Semantica delle espressioni: gli stati

Per dare la semantica delle espressioni e dei programmi è necessario considerare la nozione di **stato** σ.

Uno **stato** può essere

- **proprio**: una funzione che assegna a ciascuna variabile di tipo T un valore in $\mathcal{D}_T$, i.e. $\sigma : Var \to \mathcal{D}$;

- **di errore**:
  - $\perp$, che rappresenta la **divergenza** di un programma,
  - **fail**, che denota **errore** in una esecuzione di un programma,
  - $\triangle$, che rappresenta **deadlock** in una esecuzione di un programma.

L'insieme degli stati propri è chiamato Σ.

Notazione sugli stati

Sia $Z \subseteq Var$ un insieme di variabili.

- $\sigma[Z]$ denota la **restrizione** dello stato σ alle variabili in Z.
- Per stati di errore, definiamo $\bot[Z] = \bot$, $\mathbf{fail}[Z] = \mathbf{fail}$, $\Delta[Z] = \Delta$.
- $X, Y \subseteq \Sigma$ **concordano modulo** Z, $X = Y \ \mathbf{mod}\ Z$, se

$$\{\sigma[Var \setminus Z] \mid \sigma \in X\} = \{\sigma[Var \setminus Z] \mid \sigma \in Y\} \ .$$

Semantica delle espressioni: funzione di interpretazione

La semantica di una espressione s di tipo T è una funzione

$$\mathcal{S}[\![s]\!] : \Sigma \to \mathcal{D}_T$$

definita per **induzione** sulla struttura di s:

- se s è una variabile, $\mathcal{S}[\![s]\!](\sigma) = \sigma(s)$;
- se s è una costante di tipo base, $\mathcal{S}[\![s]\!](\sigma) = \mathcal{I}(s)$;
- se $s \equiv op(s_1, \ldots, s_n)$ dove *op* è una costante di ordine superiore, $\mathcal{S}[\![s]\!](\sigma) = \mathcal{I}(op)(\mathcal{S}[\![s_1]\!](\sigma), \ldots, \mathcal{S}[\![s_n]\!](\sigma))$;
- se $s \equiv a[s_1, \ldots, s_n]$, $\mathcal{S}[\![s]\!](\sigma) = \sigma(a)(\mathcal{S}[\![s_1]\!](\sigma), \ldots, \mathcal{S}[\![s_n]\!](\sigma))$;

. . . funzione di interpretazione

- se $s \equiv \textbf{if } B \textbf{ then } s_1 \textbf{ else } s_2 \textbf{ fi}$,

$$\mathcal{S}[\![s]\!](\sigma) = \begin{cases} \mathcal{S}[\![s_1]\!](\sigma) & \text{if } \mathcal{S}[\![B]\!](\sigma) = \textbf{true} \\ \mathcal{S}[\![s_2]\!](\sigma) & \text{if } \mathcal{S}[\![B]\!](\sigma) = \textbf{false} \end{cases}$$

- se $s \equiv (s_1)$, $\mathcal{S}[\![s]\!](\sigma) = \mathcal{S}[\![s_1]\!](\sigma)$.

Useremo l'abbreviazione $\sigma(s)$ per $\mathcal{S}[\![s]\!](\sigma)$.

Aggiornamento (update) di stati

Sia σ uno **stato proprio**, u una variabile semplice o sottoscritta di tipo T e d un elemento di $\mathcal{D}_T$.

- Se u è una **variabile semplice**, lo stato $\sigma[u := d]$ coincide con σ, eccetto che sulla variabile u, dove vale d, i.e.:

$$(\sigma[u := d])(v) = \begin{cases} d & \text{if } u \equiv v \\ \sigma(v) & \text{altrimenti} \end{cases}$$

- se u è una **variabile sottoscritta**, $a[t_1, \ldots, t_n]$, lo stato $\sigma[u := d]$ coincide con σ, eccetto che sulla variabile u per cui il valore $\sigma(a)(\sigma(t_1), \ldots, \sigma(t_n))$ è cambiato con d, i.e.:

$$(\sigma[u := d])(v) = \sigma(v) \quad \text{if } a \neq v$$

$$(\sigma[u := d])(a)(d_1, \ldots, d_n) = \begin{cases} d & \text{if } d_i = \sigma(t_i) \text{ for all } i \in \{1, \ldots, n \\ \sigma(a)(d_1, \ldots, d_n) & \text{altrimenti .} \end{cases}$$

Aggiornamento di stati di errore

Sia u una variabile semplice o sottoscritta di tipo T e d un elemento di tipo T.

Definiamo:

- $\perp[u := d] = \perp$
- $\Delta[u := d] = \Delta$
- $\mathbf{fail}[u := d] = \mathbf{fail}$

Semantica dei programmi deterministici

La **semantica** di un programma deterministico S è una funzione $\mathcal{M}[\![S]\!]$ dall'insieme degli stati propri nell'insieme degli stati.

Come definiamo $\mathcal{M}[\![S]\!]$?

- **Semantica denotazionale**: si definisce un **dominio semantico** degli stati e poi si definisce $\mathcal{M}[\![S]\!]$ per **induzione** sulla struttura di S (usando tecniche di **punto fisso** per i loop e la ricorsione, [Scott-Strachey71, Stoy77]).

  Svantaggio: è molto complicata per programmi non-deterministici, paralleli e distribuiti.

- **Semantica operazionale** [Hennessy-Plotkin79, Plotkin81]: $\mathcal{M}[\![S]\!]$ è definita in termini di una **relazione di transizione** $\rightarrow$ tra **configurazioni** di una **macchina astratta**. A seconda della definizione di configurazione, $\rightarrow$ modella l'**esecuzione** a vari livelli di astrazione.

Transizioni

Configurazioni: $\langle S, \sigma \rangle$, dove S è un programma (eventualmente il **programma vuoto** E) e σ uno stato.

Transizioni: $\rightarrow$ è una relazione tra configurazioni t.c. lo stato iniziale σ è proprio

$$\langle S, \sigma \rangle \rightarrow \langle R, \tau \rangle$$

Intuitivamente: un passo di esecuzione di S a partire dallo stato σ porta in uno stato τ; R è il programma che rimane da eseguire.

Formalmente: la relazione di transizione $\rightarrow$ è definita per induzione sulla struttura dei programmi tramite un **sistema formale**, chiamato **sistema di transizione**.

Che cosa è un sistema formale di dimostrazione?

Sistema formale di prova

Un **sistema di prova** o **calcolo** su un insieme Φ di formule è un insieme finito di **schemi di assiomi** e di **regole di derivazione**.

Schema di assioma (o **assioma**) $\mathcal{A}$: è un sottoinsieme decidibile di Φ,

$$\mathcal{A}: \quad \phi \qquad \text{dove ``...''},$$

dove la condizione "..." è una condizione decidibile sulla formula ϕ.

Regola di derivazione: è una relazione $k+1$-aria $\mathcal{R}$ sull'insieme Φ di formule,

$$\mathcal{R}: \qquad \frac{\phi_1, \ldots, \phi_k}{\phi} \qquad \text{dove ``...''},$$

dove la condizione "..." è una condizione decidibile sulle formule $\phi_1, \ldots, \phi_k, \phi$.

Dalle **premesse** $\phi_1, \ldots, \phi_k$, si deduce la formula ϕ (**conclusione**), se vale la condizione di applicabilità "...".

Derivazioni

Una **derivazione** (**prova**, **dimostrazione**) di un formula ϕ in un sistema formale $\mathcal{P}$ è una sequenza **finita**

$$\phi_1$$

$$\vdots$$

$$\phi_n$$

di formule tali che:

- $\phi = \phi_n$

- ogni formula ϕ_i è un assioma o è ottenuta mediante applicazione di una regola $\mathcal{R}$. I.e. esistono formule $\phi_{i_1}, \ldots, \phi_{i_k}$ con $i_1, \ldots, i_k < i$ t.c. $(\phi_{i_1}, \ldots, \phi_{i_k}, \phi_i) \in \mathcal{R}$.
  In particolare, ϕ_1 è un assioma.

L'ultima formula ϕ di una derivazione è detta **teorema** in $\mathcal{P}$, e si scrive $\vdash_P \phi$ (**giudizio**, i.e. formula derivata o asserita vera).

Semantica operazionale: sistema di transizione

(i) $\langle skip, \sigma \rangle \rightarrow \langle E, \sigma \rangle$

(ii) $\langle u := t, \sigma \rangle \rightarrow \langle E, \sigma[u := \sigma(t)] \rangle$

(iii) $$\frac{\langle S_1, \sigma \rangle \rightarrow \langle S_2, \tau \rangle}{\langle S_1; S, \sigma \rangle \rightarrow \langle S_2; S, \tau \rangle}$$

(iv) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma \rangle \rightarrow \langle S_1, \sigma \rangle$ dove $\sigma(B) = \textbf{true}$

(v) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma \rangle \rightarrow \langle S_2, \sigma \rangle$ dove $\sigma(B) = \textbf{false}$

(vi) $\langle \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma \rangle \rightarrow \langle S; \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma \rangle$ dove $\sigma(B) = \textbf{true}$

(vii) $\langle \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma \rangle \rightarrow \langle E, \sigma \rangle$ dove $\sigma(B) = \textbf{false}$

Computazioni

- Una **sequenza di transizioni** di S a partire da σ è una sequenza finita o infinita di configurazioni $\langle S_i, \sigma_i \rangle$ $(i \geq 0)$ t.c.

$$\langle S, \sigma \rangle = \langle S_0, \sigma_0 \rangle \rightarrow \langle S_0, \sigma_0 \rangle \rightarrow \ldots \rightarrow \langle S_i, \sigma_i \rangle \rightarrow \ldots$$

- Una **computazione** di S a partire da σ è una sequenza di transizioni non estendibile.

- Una computazione di S a partire da σ **termina** in τ se è finita e la configurazione finale è $\langle E, \tau \rangle$.

- Una computazione di S a partire da σ **diverge** se è infinita.

- Sia $\rightarrow^*$ la **chiusura riflessiva e transitiva** di $\rightarrow$, i.e.:

$$\langle S, \sigma \rangle \rightarrow^* \langle R, \tau \rangle$$

se e solo se esistono $\langle S_1, \sigma_1 \rangle, \ldots, \langle S_n, \sigma_n \rangle$, $n \geq 0$, t.c.

$$\langle S, \sigma \rangle = \langle S_1, \sigma_1 \rangle \rightarrow \ldots \rightarrow \langle S_n, \sigma_n \rangle = \langle R, \tau \rangle \ .$$

Determinismo

Lemma.
Per ogni programma S e stato proprio σ, esiste un'**unica** computazione di S a partire da σ.

Assenza di deadlock

Lemma.
Se $S \not\equiv E$, allora, per ogni stato proprio σ, esiste una configurazione $\langle S_1, \tau \rangle$ t.c.

$$\langle S, \sigma \rangle \rightarrow \langle S_1, \tau \rangle \ .$$

Semantica operazionale (input/output)

- **Correttezza parziale**: $\mathcal{M}[\![S]\!] : \Sigma \to \mathcal{P}(\Sigma)$

$$\mathcal{M}[\![S]\!](\sigma) = \{\tau \mid \langle S, \sigma\rangle \to^* \langle E, \tau\rangle\}$$

- **Correttezza totale**: $\mathcal{M}_{tot}[\![S]\!] : \Sigma \to \mathcal{P}(\Sigma \cup \{\bot\})$

$$\mathcal{M}_{tot}[\![S]\!](\sigma) = \mathcal{M}[\![S]\!](\sigma) \cup \{\bot \mid S \text{ diverge da } \sigma\} \ .$$

Proprietà della semantica

Definizioni preliminari

- Sia $\Omega \equiv \textbf{while true do } \mathit{skip} \textbf{ od}$

- Definiamo:
  $(\textbf{while } B \textbf{ do } S \textbf{ od})^0 = \Omega$
  $(\textbf{while } B \textbf{ do } S \textbf{ od})^{k+1} = \textbf{if } B \textbf{ then } S;\ (\textbf{while } B \textbf{ do } S \textbf{ od})^k \textbf{ else } \mathit{skip} \textbf{ fi}$

- Denotiamo $\mathcal{M}$ o $\mathcal{M}_{tot}$ con $\mathcal{N}$.

  - Estendiamo $\mathcal{N}$ sullo stato di errore $\perp$:

$$\mathcal{M}[\![S]\!](\perp) = \emptyset \qquad \mathcal{M}_{tot}[\![S]\!](\perp) = \{\perp\}$$

  - Estendiamo $\mathcal{N}$ ad insiemi di stati: sia $X \subseteq \Sigma \cup \{\perp\}$,

$$\mathcal{N}[\![S]\!](X) = \bigcup_{\sigma \in X} \mathcal{N}[\![S]\!](\sigma) \ .$$

Input/Output Lemma

Lemma.

(i) $\mathcal{N}[\![S]\!]$ è **monotona** su insiemi di stati, i.e.

$$X \subseteq Y \subseteq \Sigma \cup \{\bot\} \;\Rightarrow\; \mathcal{N}[\![S]\!](X) \subseteq \mathcal{N}[\![S]\!](Y) \;.$$

(ii) $\mathcal{N}[\![S_1; S_2]\!](X) = \mathcal{N}[\![S_2]\!](\mathcal{N}[\![S_1]\!](X))$.

(iii) $\mathcal{N}[\![(S_1; S_2); S_3]\!](X) = \mathcal{N}[\![S_1; (S_2; S_3)]\!](X)$

(iv) $\mathcal{N}[\![\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}]\!](X) =$
$\mathcal{N}[\![S_1]\!](X \cap [\![B]\!]) \cup \mathcal{N}[\![S_2]\!](X \cap [\![\neg B]\!]) \cup \{\bot \mid \bot \in X \text{ and } \mathcal{N} = \mathcal{M}_{tot}\}$,
where $[\![B]\!] = \{\sigma \in \Sigma \mid \sigma(B) = \textbf{true}\}$.

(v) $\mathcal{M}[\![\textbf{while } B \textbf{ do } S \textbf{ od}]\!] = \cup_{k=0}^{\infty} \mathcal{M}[\![(\textbf{while } B \textbf{ do } S \textbf{ od})^k]\!]$.

Change and Access Lemma

Lemma.
Siano σ, τ stati propri.

(i)

$$\tau \in \mathcal{N}[\![S]\!](\sigma) \;\Rightarrow\; \tau[Var \setminus change(S)] = \sigma[Var \setminus change(S)] \;.$$

(ii)

$$\sigma[var(S)] = \tau[var(S)] \;\Rightarrow\; (\mathcal{N}[\![S]\!](\sigma) = \mathcal{N}[\![S]\!](\tau) \;\mathbf{mod}\; Var \setminus var(S)) \;.$$