La Matematica dell'Informatica
Note Del Corso

Ugo Dal Lago Cecilia Di Florio

1 Introduzione

Consideriamo la seguente funzione Python.

```
def A():
    x=1
    y=0
    while x<=100:
        y=y+x
        x=x+1
    return y
```

Pensiamo un attimo a cosa questa funzione sia stata progettata per fare. Come potremmo dimostrare che la nostra idea è quella corretta.

Consideriamo invece quest'altra funzione Python, che fa qualcosa di leggermente più interessante:

```
def B(x,y):
    while x!=y:
        if x<y:
            y=y-x
        else:
            x=x-y
    return x
```

Ancora, come potremmo convincerci che questo programma faccia quello che abbiamo in mente?

Infine, consdieriamo questa terza funzione.

```
def C(a):
    i=len(a)-1
    while i>0:
        j=0
        while j<i:
            if a[j]>a[j+1]:
                aux=a[j]
                a[j]=a[j+1]
                a[j+1]=aux
            j=j+1
        i=i-1
```

Anche qui, ci chiediamo cosa questa funzione possa fare.

2 Il Linguaggio IMP

L'informatica teorica mette a disposizione tutta una serie di strumenti e modelli adatti a dare una risposta alle domande che ci siamo appena fatti. In particolare, il concetto di programma può essere

distillato fino a ricavarne la vera essenza. Nel far ciò, in particolare, si astrae da moltissimi aspetti del linguaggio di programmazione, fino a dare una rappresentazione di programmi sostanzialmente indipendente dal linguaggio ma dipendente solo dal paradigma di programmazione. In questa sezione vedremo come ciò sia possibile, nel particolare caso del paradigma *imperativo*, sicuramente il più comune.

Partiamo dai seguenti *tre* insiemi, che assumiamo come dati e i cui elementi sono per noi entità atomiche senza una struttura interna:

- L'insieme $\mathbb{Z}$ degli interi. Generici elementi di questo insieme verranno indicati con n, m, etc.
- L'insieme $\mathbb{B}$ dei valori booleani True e False. Generici elementi di quest'altro insieme saranno invece indicati con α, β, .
- L'insieme $\mathbb{L}$ delle locazioni, che spesso nei linguaggi di programmazione vengono dette anche variabili o identificatori. Possono essere, ad esempio, delle stringhe. Elementi di questo insieme verranno indicati con X, Y, etc.

A partire da questi tre insiemi si possono definire altri tre insiemi di espressioni. Partiamo da quello delle espressioni aritmetiche, che pensiamo definite dalla seguente *grammatica*

$$t ::= n \mid X \mid t + u \mid t - u \mid t \times u$$

la quale dà luogo all'insieme $\mathbb{AEXP}$. Ma cosa vogliamo dire esattamente con la definizione che abbiamo dato? Vogliamo dire che gli elementi di $\mathbb{AEXP}$ sono *tutte e sole* le espressioni che possono essere dimostrate essere parte della grammatica attraverso una derivazione finita, ovvero tutte le espressioni t per le quali esista una *prova* del fatto che $t \in \mathbb{AEXP}$ ottenuta tramite le regole seguenti:

$$\frac{n \in \mathbb{Z}}{n \in \mathbb{AEXP}} \qquad \frac{X \in \mathbb{L}}{X \in \mathbb{AEXP}} \qquad \frac{t \in \mathbb{AEXP} \quad u \in \mathbb{AEXP}}{t + u \in \mathbb{AEXP}}$$

$$\frac{t \in \mathbb{AEXP} \quad u \in \mathbb{AEXP}}{t - u \in \mathbb{AEXP}} \qquad \frac{t \in \mathbb{AEXP} \quad u \in \mathbb{AEXP}}{t \times u \in \mathbb{AEXP}}$$

Esempio 1. *Mediante le regole appena introdotte possiamo verificare, ad esempio, che* $(5+X)\times 4 \in \mathbb{AEXP}$. *Infatti, dalle prime due regole, sappiamo che*

$$\frac{5 \in \mathbb{Z}}{5 \in \mathbb{AEXP}} \qquad \frac{X \in \mathbb{L}}{X \in \mathbb{AEXP}} \qquad \frac{4 \in \mathbb{Z}}{4 \in \mathbb{AEXP}}$$

Da cui, applicando la regola relativa alla somma, abbiamo che

$$\frac{5 \in \mathbb{AEXP} \quad X \in \mathbb{AEXP}}{5 + X \in \mathbb{AEXP}}$$

Da cui, applicando la regola relativa al prodotto, otteniamo quindi

$$\frac{5 + X \in \mathbb{AEXP} \quad 4 \in \mathbb{AEXP}}{(5 + X) \times 4 \in \mathbb{AEXP}}$$

Riassumendo quanto fatto, abbiamo

$$\frac{\dfrac{\dfrac{5 \in \mathbb{Z}}{5 \in \mathbb{AEXP}} \quad \dfrac{X \in \mathbb{L}}{X \in \mathbb{AEXP}}}{5 + X \in \mathbb{AEXP}} \quad \dfrac{4 \in \mathbb{Z}}{4 \in \mathbb{AEXP}}}{(5 + X) \times 4 \in \mathbb{AEXP}}$$

Prima di tutto, va detto che l'uso delle parentesi che faremo sarà abbastanza disinvolto, anche se nella grammatica di cui sopra le parentesi non compaiono. Osserviamo poi come espressioni aritmetiche distinte ma con lo stesso valore siano da considerarsi a tutti gli effetti diverse, almeno a questo punto. Ad esempio, $5 + (3 - 2)$ e 6 sono diverse. A tal proposito, possiamo utilizzare il simbolo $\equiv$ per indicare che due espressioni che appartengono allo stesso insieme sintattico sono

identiche dal punto di vista della sintassi, ovvero sono costruite *esattamente* nello stesso modo. Scriveremo quindi, ad esempio, $(3-2) \equiv 3-2$, mentre invece sarà $5+(3-2) \not\equiv 6$.

Passiamo ora a considerare ora l'insieme $\mathbb{BEXP}$ delle espressioni booleane, definite dalla seguente grammatica

$$b ::= \mathsf{True} \mid \mathsf{False} \mid t = u \mid t \leq u \mid \neg b \mid b \vee a \mid b \wedge a$$

Per poter intervenire più agilmente sulle espressioni booleane, considereremo due operatori binari $\mathsf{AND}, \mathsf{OR}, \mathsf{NEG} : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ e un operatore unario $\mathsf{NEG} : \mathbb{B} \to \mathbb{B}$, così definiti:

$$\begin{aligned}
\mathsf{AND}(\mathsf{True}, \mathsf{True}) &= \mathsf{True} \\
\mathsf{AND}(\mathsf{True}, \mathsf{False}) = \mathsf{AND}(\mathsf{False}, \mathsf{False}) &= \mathsf{AND}(\mathsf{False}, \mathsf{True}) = \mathsf{False} \\
\mathsf{OR}(\mathsf{False}, \mathsf{False}) &= \mathsf{False} \\
\mathsf{OR}(\mathsf{True}, \mathsf{False}) = \mathsf{OR}(\mathsf{False}, \mathsf{True}) &= \mathsf{OR}(\mathsf{True}, \mathsf{True}) = \mathsf{True} \\
\mathsf{NEG}(\mathsf{False}) &= \mathsf{True} \\
\mathsf{NEG}(\mathsf{True}) &= \mathsf{False}
\end{aligned}$$

Esempio 2. *Espressioni che esprimono un confronto tra numeri interi, e.g.* $5 \leq X$, $7 = X + Y$, *sono semplici espressioni booleane. Possiamo considerare anche espressoni leggermente più complesse, quali* $(11 \leq 7) \wedge ((3 + X) \leq (6 \times 9))$, $\neg(X = 2) \vee (Y - 3 \leq 0)$

Definiamo infine l'insieme $\mathbb{COM}$ dei comandi, a partire alla grammatica

$$c ::= \texttt{skip} \mid X := t \mid c; d \mid \texttt{if } b \texttt{ then } c \texttt{ else } d \mid \texttt{while } b \texttt{ do } c$$

Esempio 3. *In* **IMP** *possiamo definire, servendoci delle espressioni aritmetiche, delle espressioni booleane e dei comandi, alcuni semplici programmi. Ad esempio:*

- $X := 0; \texttt{while } (X \leq 100) \texttt{ do } (X := X + 1)$, *programma che scorre i primi 101 numeri interi non negativi.*
- $Y := 0; X := 0; \texttt{while } (X \leq 100) \texttt{ do } ((X := X + 1); (Y := Y + X))$, *programma che somma i primi 101 numeri interi non negativi.*
- $\texttt{while } \neg(X = Y) \texttt{ do } (\texttt{if } X \leq Y \texttt{ then } (Y := Y - X) \texttt{ else } (X := X - Y))$, *che corrisponde al programma B descritto nell'Introduzione.*

Consideriamo ora il primo esempio e verifichiamo che abbiamo effettivamente un elemento di $\mathbb{COM}$. *Abbiamo che:*

$$\frac{0 \in \mathbb{AEXP}}{X := 0 \in \mathbb{COM}} \qquad \frac{X \in \mathbb{AEXP} \quad 100 \in \mathbb{AEXP}}{X \leq 100 \in \mathbb{BEXP}}$$

Inoltre

$$\frac{\dfrac{X \in \mathbb{AEXP} \quad 1 \in \mathbb{AEXP}}{X + 1 \in \mathbb{AEXP}}}{X := X + 1 \in \mathbb{COM}}$$

Da cui abbiamo che

$$\frac{X \leq 100 \in \mathbb{BEXP} \quad X := X + 1 \in \mathbb{COM}}{\texttt{while } (X \leq 100) \texttt{ do } (X := X + 1) \in \mathbb{COM}}$$

e infine

$$\frac{X := 0 \in \mathbb{COM} \quad \texttt{while } (X \leq 100) \texttt{ do } X := X + 1 \in \mathbb{COM}}{X := 0; \texttt{while } (X \leq 100) \texttt{ do } (X := X + 1) \in \mathbb{COM}}$$

Ricapitolando,

$$\frac{\dfrac{0 \in \mathbb{AEXP}}{X := 0 \in \mathbb{COM}} \qquad \dfrac{\dfrac{X \in \mathbb{AEXP} \quad 100 \in \mathbb{AEXP}}{X \leq 100 \in \mathbb{BEXP}} \qquad \dfrac{\dfrac{X \in \mathbb{AEXP} \quad 1 \in \mathbb{AEXP}}{X + 1 \in \mathbb{AEXP}}}{X := X + 1 \in \mathbb{COM}}}{\texttt{while } (X \leq 100) \texttt{ do } (X := X + 1) \in \mathbb{COM}}}{X := 0; \texttt{while } (X \leq 100) \texttt{ do } (X := X + 1) \in \mathbb{COM}}$$

3 La Valutazione delle Espressioni e dei Comandi

Finora, i programmi per come li abbiamo definiti hanno una natura meramente statica. Occorre ora capire come sia possibile, con lo stesso grado di precisione matematica, catturare il *comportamento* di tali programmi, in modo da poter poi dimostrare alcune proprietà relative ad esso.

Si tratta in realtà di non fare altro che formalizzare un modello che, in senso intuitivo, utilizziamo comunemente tutti, ma che non viene reso esplicito e a cui non si dà di solito lo status di una costruzione matematica. In tale modello, la valutazione di un comando c dipende in modo determinante dallo *stato corrente* ovvero dal valore immagazzinato in ciascuna delle variabili *prima* dell'esecuzione del comando. Come formalizzare tale concetto? Basta stabilire che uno *stato* è nient'altro che una funzione $\sigma : \mathbb{L} \to \mathbb{Z}$, ovvero una funzione che associ ad *ogni* variabile il valore intero che essa contiene.

3.1 Valutazione delle Espressioni Aritmetiche

Prima di parlare di comandi, al solito, concentriamoci sulle espressioni aritmetiche. La valuazione di una tale espressione t dipende quindi da uno stato σ e darà luogo ad un valore intero n. In tal caso scriveremo

$$\langle t, \sigma \rangle \longrightarrow n$$

È a questo punto chiaro che, ad esempio, $\langle (3+1) \times 2, \sigma \rangle \longrightarrow 8$ e che $\langle 5 + X, \sigma \rangle \longrightarrow 5 + \sigma(X)$. Ma come *formalizzare* in modo preciso tutto questo. Anche qui, ci vengono in aiuto delle regole, le quali però non ci permettono di concludere l'appartenenza di certe espressioni a certi insiemi, ma ci permettono di concludere giudizi nella forma $\langle t, \sigma \rangle \longrightarrow n$. Le regole sono le seguenti:

$$\langle n, \sigma \rangle \longrightarrow n \qquad \langle X, \sigma \rangle \longrightarrow \sigma(X)$$

$$\frac{\langle t, \sigma \rangle \longrightarrow n \quad \langle u, \sigma \rangle \longrightarrow m}{\langle t + u, \sigma \rangle \longrightarrow n + m} \qquad \frac{\langle t, \sigma \rangle \longrightarrow n \quad \langle u, \sigma \rangle \longrightarrow m}{\langle t - u, \sigma \rangle \longrightarrow n - m} \qquad \frac{\langle t, \sigma \rangle \longrightarrow n \quad \langle u, \sigma \rangle \longrightarrow m}{\langle t \times u, \sigma \rangle \longrightarrow n \times m}$$

Esempio 4. *Possiamo ora verificare che* $\langle (3+1) \times 2, \sigma \rangle \longrightarrow 8$. *Infatti dalla regola di valutazione per i numeri interi sappiamo che*

$$\langle 3, \sigma \rangle \longrightarrow 3 \qquad \langle 1, \sigma \rangle \longrightarrow 1 \qquad \langle 2, \sigma \rangle \longrightarrow 2$$

Dalla regola per la valutazione della somma abbiamo inoltre

$$\frac{\langle 3, \sigma \rangle \longrightarrow 3 \quad \langle 1, \sigma \rangle \longrightarrow 1}{\langle 3 + 1, \sigma \rangle \longrightarrow 4}$$

Infine dalla regola per la valutazione del prodotto abbiamo che

$$\frac{\langle 3 + 1, \sigma \rangle \longrightarrow 4 \quad \langle 2, \sigma \rangle \longrightarrow 2}{\langle (3 + 1) \times 2, \sigma \rangle \longrightarrow 8}$$

Riassumendo,

$$\frac{\dfrac{\langle 3, \sigma \rangle \longrightarrow 3 \quad \langle 1, \sigma \rangle \longrightarrow 1}{\langle 3 + 1, \sigma \rangle \longrightarrow 4} \qquad \langle 2, \sigma \rangle \longrightarrow 2}{\langle (3 + 1) \times 2, \sigma \rangle \longrightarrow 8}$$

3.2 Valutazioni delle Espressioni Booleane

In modo molto simile a quanto abbiamo fatto per le espressioni aritmetiche, possiamo dare delle regole di valutazione per le espressioni booleane, che in analogia a quelle per le espressioni aritmentiche, deriveranno dei giudizi nella forma $\langle b, \sigma\rangle \longrightarrow w$.

$$\langle \mathsf{True}, \sigma\rangle \longrightarrow \mathsf{True} \qquad \langle \mathsf{False}, \sigma\rangle \longrightarrow \mathsf{False}$$

$$\frac{\langle t, \sigma\rangle \longrightarrow n \quad \langle u, \sigma\rangle \longrightarrow m \quad n = m}{\langle b = a, \sigma\rangle \longrightarrow \mathsf{True}} \qquad \frac{\langle t, \sigma\rangle \longrightarrow n \quad \langle u, \sigma\rangle \longrightarrow m \quad n \neq m}{\langle b = a, \sigma\rangle \longrightarrow \mathsf{False}}$$

$$\frac{\langle t, \sigma\rangle \longrightarrow n \quad \langle u, \sigma\rangle \longrightarrow m \quad n \leq m}{\langle b \leq a, \sigma\rangle \longrightarrow \mathsf{True}} \qquad \frac{\langle t, \sigma\rangle \longrightarrow n \quad \langle u, \sigma\rangle \longrightarrow m \quad n \not\leq m}{\langle b \leq a, \sigma\rangle \longrightarrow \mathsf{False}}$$

$$\frac{\langle b, \sigma\rangle \longrightarrow w}{\langle \neg b, \sigma\rangle \longrightarrow \mathsf{NEG}(w)} \qquad \frac{\langle b, \sigma\rangle \longrightarrow w_0 \quad \langle a, \sigma\rangle \longrightarrow w_1}{\langle b \wedge a, \sigma\rangle \longrightarrow \mathsf{AND}(w_0, w_1)} \qquad \frac{\langle b, \sigma\rangle \longrightarrow w_0 \quad \langle a, \sigma\rangle \longrightarrow w_1}{\langle b \vee a, \sigma\rangle \longrightarrow \mathsf{OR}(w_0, w_1)}$$

3.3 Valutazione dei Comandi

Infine, siamo finalmente in grado di dare un significato ai programmi. Differentemente dalle espressioni, i programmi *modificano* lo stato e quindi il tipo di giudizi che andremo a costruire hanno una forma differente, ossia $\langle c, \sigma\rangle \longrightarrow \theta$. Nel far ciò, dobbiamo essere in grado di scrivere lo stato ottenuto modificando il contenuto di una singola locazione di memoria in un certo stato di partenza: se σ è uno stato, allora scriveremo $\sigma[n/X]$ per lo stato che si comporta come σ *tranne* nella locazione X, in cui invece restituisce n:

$$\sigma[n/X](Y) = \begin{cases} n & \text{se } X = Y \\ \sigma(Y) & \text{altrimenti} \end{cases}$$

Le regole sono le seguenti:

$$\langle \texttt{skip}, \sigma\rangle \longrightarrow \sigma$$

$$\frac{\langle t, \sigma\rangle \longrightarrow n}{\langle X := t, \sigma\rangle \longrightarrow \sigma[n/X]} \qquad \frac{\langle c, \sigma\rangle \longrightarrow \theta \quad \langle d, \theta\rangle \longrightarrow \xi}{\langle c; d, \sigma\rangle \longrightarrow \xi} \qquad \frac{\langle b, \sigma\rangle \longrightarrow \mathsf{True} \quad \langle c, \sigma\rangle \longrightarrow \theta}{\langle \texttt{if } b \texttt{ then } c \texttt{ else } d, \sigma\rangle \longrightarrow \theta}$$

$$\frac{\langle b, \sigma\rangle \longrightarrow \mathsf{False} \quad \langle d, \sigma\rangle \longrightarrow \theta}{\langle \texttt{if } b \texttt{ then } c \texttt{ else } d, \sigma\rangle \longrightarrow \theta} \qquad \frac{\langle b, \sigma\rangle \longrightarrow \mathsf{False} \quad \langle d, \sigma\rangle \longrightarrow \theta}{\langle \texttt{if } b \texttt{ then } c \texttt{ else } d, \sigma\rangle \longrightarrow \theta} \qquad \frac{\langle b, \sigma\rangle \longrightarrow \mathsf{False}}{\langle \texttt{while } b \texttt{ do } c, \sigma\rangle \longrightarrow \sigma}$$

$$\frac{\langle b, \sigma\rangle \longrightarrow \mathsf{True} \quad \langle c, \sigma\rangle \longrightarrow \xi \quad \langle \texttt{while } b \texttt{ do } c, \xi\rangle \longrightarrow \theta}{\langle \texttt{while } b \texttt{ do } c, \sigma\rangle \longrightarrow \theta}$$

3.4 Una Prova Di Correttezza dell'Algoritmo di Euclide

Possiamo scrivere il programma d'esempio come un elemento $c_{EUCLIDE}$ di $\mathbb{COM}$:

$$\texttt{while } \neg(X = Y) \texttt{ do } (\texttt{if } X \leq Y \texttt{ then } (Y := Y - X) \texttt{ else } (X := X - Y))$$

Prima di tutto, come possiamo *formulare* la correttezza di $c_{EUCLIDE}$. Vorremmo dire che per ogni σ tale che $\sigma(X), \sigma(Y) > 0$ vale che

$$\langle c_{EUCLIDE}, \sigma\rangle \longrightarrow \theta$$

dove $\theta(X)$ è il massimo comun divisore di $\sigma(X)$ e $\sigma(Y)$, che indichiamo con $\gcd(\sigma(X), \sigma(Y))$. Cerchiamo prima di tutto di analizzare cosa succede quando si esegue il corpo del ciclo while di cui si compone $c_{EUCLIDE}$, che chiamiamo $d_{EUCLIDE}$, e che il comando seguente

$$\texttt{if } X \leq Y \texttt{ then } (Y := Y - X) \texttt{ else } (X := X - Y)$$

Lemma 1. *Per ogni σ tale che $\sigma(X), \sigma(Y) > 0$ vale che $\langle d_{EUCLIDE}, \sigma\rangle \longrightarrow \theta$, dove* $\gcd(\sigma(X), \sigma(Y)) = \gcd(\theta(X), \theta(Y))$.

Dimostrazione. Ci serviremo di una proprietà di invarianza del gcd: si può dimostrare che se $n, m \in \mathbb{Z}, 1 \leq n \leq m$, allora $\gcd(n, m) = \gcd(n, m - n)$. Infatti, sia $k := \gcd(n, m)$ allora $n = kN$ e $m = kM$ con $N, M \in \mathbb{Z}$. k sarà sicuramente un divisore comune di n e $m - n$, possiamo assumere che sia *il più grande*? Supponiamo per assurdo che esista $h \in \mathbb{Z}, h > k$ tale che $n = hT$ e $m - n = hU$ con $T, U \in \mathbb{Z}$, allora $m = n + (m - n) = hT + hU = h(T + U)$. Quindi $h > k$ è un divisore comune di m e n. Assurdo. Possiamo concludere quindi $\gcd(n, m) = \gcd(n, m - n)$.

Ritorniamo ora alla nostra dimostrazione e distinguiamo due casi:

- Supponiamo che sia $\sigma(X) \leq \sigma(Y)$. Allora dalle regole di valutazione per le espressioni booleane segue che $\langle X \leq Y, \sigma\rangle \longrightarrow \mathsf{True}$. D'altronde, dalle regole sulla sottrazione e sull'assegnazione possiamo giungere a derivare

$$\frac{\langle Y - X, \sigma\rangle \longrightarrow \sigma(Y) - \sigma(X)}{\langle Y := Y - X, \sigma\rangle \longrightarrow \sigma[\sigma(Y) - \sigma(X)/Y]}$$

  Da quanto detto, avvalendoci delle regole per il comando condizionale, otteniamo

$$\frac{\langle X \leq Y, \sigma\rangle \longrightarrow \mathsf{True} \quad \langle Y := Y - X, \sigma\rangle \longrightarrow \sigma[\sigma(Y) - \sigma(X)/Y]}{\langle \texttt{if } X \leq Y \texttt{ then } Y := Y - X \texttt{ else } X := X - Y, \sigma\rangle \longrightarrow \sigma[\sigma(Y) - \sigma(X)/Y]}$$

  In questo caso, lo stato finale è raggiunto sostituendo, nello stato iniziale, a $\sigma(Y)$ la differenza $\sigma(Y) - \sigma(X)$. Ma allora, grazie alla proprietà che abbiamo dimostrato sul gcd, possiamo dire che $\gcd(\sigma(Y) - \sigma(X), \sigma(X)) = \gcd(\sigma(Y), \sigma(X))$.
- Analogamente si procede se $\sigma(Y) \leq \sigma(X)$ giungendo a

$$\langle \texttt{if } X \leq Y \texttt{ then } Y := Y - X \texttt{ else } X := X - Y, \sigma\rangle \longrightarrow \sigma[\sigma(X) - \sigma(Y)/Y]$$

  In questo caso invece, lo stato finale è raggiunto sostituendo, nello stato iniziale, a $\sigma(X)$ la differenza $\sigma(X) - \sigma(Y)$. Ancora una volta, avvalendoci della proprietà sul gcd, abbiamo che che $\gcd(\sigma(X) - \sigma(Y), \sigma(Y)) = \gcd(\sigma(X), \sigma(Y))$.

□

Proposition 1. *Per ogni σ tale che $\sigma(X), \sigma(Y) > 0$ vale che*

$$\langle c_{EUCLIDE}, \sigma\rangle \longrightarrow \theta$$

dove $\theta(X)$ è il massimo comun divisore di $\sigma(X)$ e $\sigma(Y)$

Dimostrazione. Procediamo per induzione (forte) su $\sigma(X) + \sigma(Y)$:

- Passo base: $\sigma(X) + \sigma(Y) = 2$. Ciò accade se $\sigma(X) = \sigma(Y) = 1$. In tal caso $\sigma(X) = \sigma(Y) = \gcd(\sigma(X), \sigma(Y))$ e dalle regole di valutazione delle espressioni booleane si ha che

$$\frac{\dfrac{\langle X, \sigma\rangle \longrightarrow \sigma(X) \quad \langle X, \sigma\rangle \longrightarrow \sigma(Y) \quad \sigma(X) = \sigma(Y)}{\langle X = Y, \sigma\rangle \longrightarrow \mathsf{True}}}{\langle \neg(X = Y), \sigma\rangle \longrightarrow \mathsf{NEG}(\mathsf{True}) = \mathsf{False}}$$

  Da questa derivazione abbiamo, considerando le regole di valutazione su `while do` , che

$$\frac{\langle \neg(X = Y), \sigma\rangle \longrightarrow \mathsf{False}}{\langle c_{EUCLIDE}, \sigma\rangle \longrightarrow \sigma}$$

  con $\sigma(X) = \sigma(Y) = \gcd(\sigma(X), \sigma(Y))$

- Passo induttivo: $\sigma(X) + \sigma(Y) \leq n$, $n \in \mathbb{Z}, n > 2$. Supponiamo che in tal caso valga $\langle c_{EUCLIDE}, \sigma\rangle \longrightarrow \theta$, dove $\theta(X) = \gcd(\sigma(X), \sigma(Y))$.
  Verifichiamo ora che la tesi valga nel caso in cui $\sigma(X) + \sigma(Y) = n + 1$. Se $\sigma(X) = \sigma(Y)$, allora, come prima, possiamo ottenere $\langle \neg(X = Y), \sigma\rangle \longrightarrow \mathsf{False}$ e quindi $\langle c_{EUCLIDE}, \sigma\rangle \longrightarrow \sigma$ e $\sigma(X) = \sigma(Y) = gcd(\sigma(X), \sigma(Y))$. Altrimenti, se $\sigma(X) \neq \sigma(Y)$, innanzittutto abbiamo, dalle regole di valutazione delle espressioni booleane, che

$$\dfrac{\dfrac{\langle X, \sigma\rangle \longrightarrow \sigma(X) \quad \langle X, \sigma\rangle \longrightarrow \sigma(Y) \quad \sigma(X) \neq \sigma(Y)}{\langle X = Y, \sigma\rangle \longrightarrow \mathsf{False}}}{\langle \neg(X = Y), \sigma\rangle \longrightarrow \mathsf{NEG(False)} = \mathsf{True}}$$

  Dal lemma precedente, sappiamo poi che $\langle d_{EUCLIDE}, \sigma\rangle \longrightarrow \xi$, dove $\gcd(\sigma(X), \sigma(Y)) = \gcd(\xi(X), \xi(Y))$. A questo punto, al passo successivo avremo

$$\xi(X) + \xi(Y) = \begin{cases} \sigma(X) + \sigma(Y) - \sigma(X) \leq n, & \text{se } \sigma(X) < \sigma(Y) \\ \sigma(X) + \sigma(Y) - \sigma(X) \leq n, & \text{se } \sigma(X) < \sigma(Y) \end{cases}$$

  In ogni caso possiamo ricondurci al passo precedente dell'induzione, ottenendo $\langle c_{EUCLIDE}, \xi\rangle \longrightarrow \theta$, dove $\theta(X) = \theta(Y) = \gcd(\xi(X), \xi(Y)) = \gcd(\sigma(X), \sigma(Y))$. In definitiva, considerando le regole di valutazione su `while do` abbiamo

$$\dfrac{\langle \neg(X = Y), \sigma\rangle \longrightarrow \mathsf{True} \quad \langle d_{EUCLIDE}, \sigma\rangle \longrightarrow \xi \quad \langle c_{EUCLIDE}, \xi\rangle \longrightarrow \theta}{\langle c_{EUCLIDE}, \xi\rangle \longrightarrow \theta}$$

  con $\theta(X) = \gcd(\sigma(X), \sigma(Y))$.

□

3.5 Dimostrare La Correttezza di Una Trasformazione tra Programmi

Una caratteristica spesso indagata dei programmi è la loro *equivalenza*. Ad esempio, una volta definita un'opportuna nozione di equivalenza tra programmi, si può determinare se una certa proprietà è valida per un programma, se la stessa proprietà è nota per un programma ad esso equivalente. Inoltre, a volte, l'equivalenza dei programmi è un utile strumento formale per la definizione di una semantica di un linguaggio.

Cerchiamo ora di dare un primo esempio di equivalenza tra programmi. Consideriamo l'esecuzione del programma $w \equiv \texttt{while}\ b\ \texttt{do}\ c$, con $b \in \mathbb{BEXP}, c \in \mathbb{COM}$, in uno stato σ. Intuitivamente, ci aspettiamo che, se b viene valutato True in σ, allora w eseguirà c e poi nuovamente w; in caso contrario, se b risulta False in σ, allora l'esecuzione di w termina immediatamente lasciando lo stato invariato. Possiamo tradurre questa nostra intuizione in una richiesta più formale. Ovvero, possiamo chiederci se

$$\langle w, \sigma\rangle \longrightarrow \theta \quad \textit{sse } \langle \texttt{if}\ b\ \texttt{then}\ c; w\ \texttt{else skip}, \sigma\rangle \longrightarrow \theta$$

comunque si considerino due stati σ, θ. La proposizione successiva fornisce una risposta positiva alla nostra domanda.

Proposition 2. *Sia w il programma* `while` b `do` c *con $b \in \mathbb{BEXP}, c \in \mathbb{COM}$. Allora*

$$\langle w, \sigma\rangle \longrightarrow \theta \quad \textit{sse } \langle \texttt{if}\ b\ \texttt{then}\ c; w\ \texttt{else skip}, \sigma\rangle \longrightarrow \theta$$

comunque si considerino due stati σ, θ.

Dimostrazione. Si tratta di provare una doppia implicazione.

⇒) Vogliamo provare che

$\langle w, \sigma\rangle \longrightarrow \theta$ implica $\langle \texttt{if}\ b\ \texttt{then}\ c; w\ \texttt{else skip}, \sigma\rangle \longrightarrow \theta$, per ogni stato σ, θ.

Supponiamo innanzitutto che $\langle w, \sigma\rangle \longrightarrow \theta$ per due stati σ, θ. Considerando le regole sui comandi capiamo che possiamo giungere a due forme di derivazione finale:

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma\rangle \longrightarrow \mathsf{False}}}{\langle w, \sigma\rangle \longrightarrow \theta}$$

oppure

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma\rangle \longrightarrow \mathsf{True}} \quad \dfrac{\vdots}{\langle c, \sigma\rangle \longrightarrow \xi} \quad \dfrac{\vdots}{\langle w, \xi\rangle \longrightarrow \theta}}{\langle w, \sigma\rangle \longrightarrow \theta}$$

Nel primo caso avremo a disposizione una derivazione di $\langle b, \sigma\rangle \longrightarrow \mathsf{False}$. Ma allora, possiamo servirci di questa derivazione e costruire una derivazione di $\langle \texttt{if}\ b\ \texttt{then}\ c; w\ \texttt{else skip}, \sigma\rangle \longrightarrow \sigma$. Più precisamente:

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma\rangle \longrightarrow \mathsf{False}} \quad \langle \texttt{skip}, \sigma\rangle \longrightarrow \sigma}{\langle \texttt{if}\ b\ \texttt{then}\ c; w\ \texttt{else skip}, \sigma\rangle \longrightarrow \sigma}$$

Nel secondo caso invece abbiamo derivazioni per $\langle b, \sigma\rangle \longrightarrow \mathsf{True}$, $\langle c, \sigma\rangle \longrightarrow \xi$ e $\langle w, \xi\rangle \longrightarrow \theta$. Possiamo innanzitutto servici della seconda e terza derivazione per ottenere una derivazione di $\langle c; w, \sigma\rangle \longrightarrow \theta$

$$\dfrac{\dfrac{\vdots}{\langle c, \sigma\rangle \longrightarrow \xi} \quad \dfrac{\vdots}{\langle w, \xi\rangle \longrightarrow \theta}}{\langle c; w, \sigma\rangle \longrightarrow \theta}$$

da cui a sua volta possiamo costruire

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma\rangle \longrightarrow \mathsf{True}} \quad \dfrac{\dfrac{\vdots}{\langle c, \sigma\rangle \longrightarrow \xi} \quad \dfrac{\vdots}{\langle w, \xi\rangle \longrightarrow \theta}}{\langle c; w, \sigma\rangle \longrightarrow \theta}}{\langle \texttt{if}\ b\ \texttt{then}\ c; w\ \texttt{else skip}, \sigma\rangle \longrightarrow \theta}$$

In ogni caso abbiamo provato che

$\langle w, \sigma\rangle \longrightarrow \theta$ implica $\langle \texttt{if}\ b\ \texttt{then}\ c; w\ \texttt{else skip}, \sigma\rangle \longrightarrow \theta$, per ogni stato σ, θ.

$\Leftarrow$) Proviamo ora il viceversa, ovvero che

$\langle \texttt{if}\ b\ \texttt{then}\ c; w\ \texttt{else skip}, \sigma\rangle \longrightarrow \theta$ implica $\langle w, \sigma\rangle \longrightarrow \theta$, per ogni stato σ, θ.

Supponiamo quindi che $\langle \texttt{if}\ b\ \texttt{then}\ c; w\ \texttt{else skip}, \sigma\rangle \longrightarrow \theta$ per gli stati σ, θ. Allora dovremmo avere una derivazione finale che può essere della forma

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma\rangle \longrightarrow \mathsf{False}} \quad \langle \texttt{skip}, \sigma\rangle \longrightarrow \sigma}{\langle \texttt{if}\ b\ \texttt{then}\ c; w\ \texttt{else skip}, \sigma\rangle \longrightarrow \sigma}$$

oppure

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma\rangle \longrightarrow \mathsf{True}} \quad \dfrac{\vdots}{\langle c; w, \sigma\rangle \longrightarrow \theta}}{\langle \texttt{if}\ b\ \texttt{then}\ c; w\ \texttt{else skip}, \sigma\rangle \longrightarrow \theta}$$

dove, nel primo caso, abbiamo che $\sigma = \theta$ dal momento che l'unica derivazione associata a `skip` è della forma

$$\frac{}{\langle \texttt{skip}, \sigma \rangle \longrightarrow \sigma}$$

Ora, nel primo caso dalla derivazione di $\langle b, \sigma \rangle \longrightarrow \mathsf{False}$ otteniamo immediatamente la derivazione

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma \rangle \longrightarrow \mathsf{False}}}{\langle w, \sigma \rangle \longrightarrow \theta}$$

Consideriamo adesso il secondo caso. La derivazione di $\langle c; w, \sigma \rangle \longrightarrow \theta$ deve essere della forma

$$\dfrac{\dfrac{\vdots}{\langle c, \sigma \rangle \longrightarrow \xi} \quad \dfrac{\vdots}{\langle w, \xi \rangle \longrightarrow \theta}}{\langle c; w, \sigma \rangle \longrightarrow \theta}$$

All'interno di questa derivazione abbiamo quindi derivazioni di $\langle c, \sigma \rangle \longrightarrow \xi$ e $\langle w, \xi \rangle \longrightarrow \theta$. Ora, servendoci di queste derivazioni e di quella su $\langle b, \sigma \rangle \longrightarrow \mathsf{True}$, possiamo ottenere la derivazione

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma \rangle \longrightarrow \mathsf{True}} \quad \dfrac{\vdots}{\langle c, \sigma \rangle \longrightarrow \xi} \quad \dfrac{\vdots}{\langle w, \xi \rangle \longrightarrow \theta}}{\langle w, \sigma \rangle \longrightarrow \theta}$$

In ogni caso abbiamo provato che

$\langle \texttt{if}\ b\ \texttt{then}\ c; w\ \texttt{else}\ \texttt{skip}, \sigma \rangle \longrightarrow \theta$ implica che $\langle w, \sigma \rangle \longrightarrow \theta$, per ogni stato σ, θ.

□

Esercizi

Esercizio 1. *Si estenda il linguaggio* **IMP** *con il comando* `for`*. Intuitivamente questo comando dovrà essere del tipo :* `for`*([condizione]; [espressione finale]);comando. dove, la condizione deve essere valutata prima dell'esecuzione di ogni ciclo. Se non è verificata il comando nel corpo del ciclo non viene eseguito. L'espressione finale viene valutata al termine di ogni ciclo; tipicamente esprime un incremento su una variabile contatrice.*
Una volta definita una sintassi per il comando `for` *definire le relative regole di valutazione.*

Esercizio 2. *Si definisca un'estensione del linguaggio* **IMP***, in cui siano definiti gli array.*

Esercizio 3. *Sia* $w \equiv$ `while` True `do skip`*. Si spieghi (servendosi delle regole di valutazione) perchè, comunque si consideri* σ *non esiste* θ *tale che* $\langle w, \sigma \rangle \longrightarrow \theta$

Esercizio 4. *Sia* $w \equiv$ `if` $\mathsf{AND}(b, a)$ `then` c `else` d*. Si scriva un programma equivalente a* w *e si provi l'equivalenza servendosi delle regole di valutazione.*

4 L'Induzione Strutturale sul Sistema di Regole

Alla fine della Sezione precedente abbiamo dimostrato un programma corretto utilizzando la semantica operazionale e uno schema di induzione classico, ovvero quello sui numeri naturali. A volte, però, c'è bisogno di qualcosa di leggermente diverso e più potente. A tal scopo, consideriamo il problema di dimostrare non una proprietà di un singolo programma, ma una proprietà *della semantica operazionale in sé.* Immaginiamo, ad esempio, di voler dimostrare che tale semantica è deterministica, ovvero che per ogni c e per ogni σ se $\langle c, \sigma \rangle \longrightarrow \theta$ e $\langle c, \sigma \rangle \longrightarrow \xi$, allora $\theta = \xi$. In questo caso, come potremmo procedere?

4.1 Le Relazioni Ben Fondate e L'Induzione Strutturale

Riconsideriamo lo schema di induzione sui numeri naturali utilizzato nella dimostrazione della correttezza del programma $c_{EUCLIDE}$. L'idea alla base di tale schema è che, se se si vuole provare che una proprietà vale a partire da un certo intero non negativo (*base dell'induzione*), basta verificare che la proprietà valga per un intero non negativo n, se vale per i suoi predecessori (*passo induttivo*).

Allo stesso modo si procede per l'induzione strutturale: per provare che una proprietà è valida per tutte le espressioni di un determinato linguaggio è sufficiente provare che tale proprietà è valida per una generica espressione, se vale per le sue sottoespressioni.

Entrambe le induzioni si basano su un principio: la relazione tra le sottoespressioni, così come la relazione di predecessore tra i interi, non devono dare origine a *catene discendenti infinite*; in altre parole, il processo di suddivisione deve arrestarsi in corrispondenza di elementi *minimali*. Più formalmente, quel che si richiede, è che le relazioni siano *ben fondate*.

Definizione 1. *Una* relazione ben fondata *è una relazione binaria $\prec$ su un insieme A tale che non esistano catene discendenti $\cdots \prec r \prec \cdots \prec q \prec p$. Quando $q \prec p$ diciamo che q è un predecessore di p.*

Notiamo che una relazione ben fondata è necessariamente irriflessiva, ovvero per nessun p possiamo avere $p \prec p$ altrimenti si avrebbe la catena discendente infinita $\cdots \prec p \prec \cdots \prec p \prec p$. Scriveremo $\preceq$ per la chisura riflessiva di $\prec$, i.e.

$$q \preceq p \Leftrightarrow q = p \ \circ \ q \prec p$$

Una definizione alternativa di relazione ben fondata viene data ricorrendo agli elementi minimali.

Proposition 3. *Sia $q \prec p$ una relazione binaria su un insieme A. Allora la relazione $\prec$ è ben fondata se e solo se ogni sottoinsieme non vuoto Q di A ha elemento minimale s, ovvero tale che*

$$s \in Q \ \& \ \forall p \prec s.\ p \notin Q$$

L'insieme Q definito nella proposizione precedente si dice *ben fondato*. Più in generale possiamo dare la seguente definizione.

Definizione 2. *Un insieme A si dice ben fondato rispetto a una relazione $\prec$ se ammette elemento minimale.*

Possiamo ora definire il principio di induzione ben fondata, che afferma che, per provare che una proprietà vale per tutti gli elementi di un insieme bene fondato, è sufficiente provare che, se la proprietà vale per tutti i predecessori di un elemento arbitrario p, allora vale per p.

Definizione 3 (Principio di induzione ben fondata)**.** *Sia $\prec$ una relazione ben fondata su un insieme A. Sia P una proprietà. Allora $\forall p \in A.\ P(p)$ sse*

$$\forall p \in A.([\forall q \prec p\ P(q)] \Rightarrow P(p)).$$

4.2 L'Induzione sulle Derivazioni

A volte l'induzione strutturale è inadeguata per provare proprietà della semantica operazionale. In questi casi è utile procedere per *induzione sulla struttura delle derivazioni*. Come abbiamo visto, possiamo costruire le derivazioni servendoci di determinate *regole* che devono essere *istanziate*. Possiamo dire che l'*istanza di una regola* è ottenuta dalle regole sostituendo le metavariabili con termini o valori. L'istanza di una regola sarà quindi definita da un insieme di *premesse* (eventualmente vuoto, se siamo in presenza di un assioma) e da una *conclusione*. Le regole che ci interessano sono *finitarie* nel senso che le loro premesse sono *finite*. Di conseguenza anche le istanze di regole avranno un insieme finito di premesse.

Cerchiamo di formalizzare le derivazioni a partire dall'idea di *insieme di istanze di regole*.

Definizione 4. *Un* insieme di istanze regole R *consiste in elementi che sono coppie* (S/y) *dove* S *è un insieme* finito*, che definisce le* premesse *e* y *è un elemento, detto* conclusione*.*

Solitamente siamo abituati a vedere istanze di regole scritte in una delle seguenti forme

$$\frac{}{y} \qquad \frac{x_1, \cdots, x_n}{y}$$

In questi casi, gli insiemi delle premesse sono rispettivamente $X = \emptyset$ e $X = \{x_1, \cdots, x_n\}$.

Definizione 5. *Una* R-derivazione *di* y *è un'istanza di regola* $(\emptyset/y)$ *o una coppia* $(\{f_1, \cdots, f_n\}/y)$ *dove* $(\{x_1, \cdots, x_n/y\})$ *è una regola e* f_1 *è una* R*-derivazione di* x_1*,...,*f_n *è una* R*-derivazione di* x_n*.*

Scriviamo $f \Vdash_R y$ per indicare che f è una R-derivazione di y, ovvero:

- $(\emptyset/y) \Vdash_R y$ *se* $(\emptyset/y) \in R$
- $(\{f_1, .., f_n\} \Vdash_R y$ *se* $(\{x_1, \cdots, x_n\}/y)$ & $f_1 \Vdash_R x_1$ & $\cdots$ & $f_n \Vdash_R x_n$

Diciamo che una conclusione y è *derivata* da R se esiste una $R-$derivazione di y, i.e., $f \Vdash_R y$ per qualche derivazione f. Scriviamo $\Vdash_R y$ per indicare che y è derivata da R. In assenza di possibili ambiguità sulle regole scriviamo più brevemente $f \Vdash y$ e $\Vdash y$.

Nell'ambito della semantica operazionale, scriviamo solitamente $\langle c, \sigma\rangle \longrightarrow \theta$, sottointendendo che possiamo derivare $\langle c, \sigma\rangle \longrightarrow \theta$ dalle regole date sui comandi. Se volessimo evidenziare il ruolo giocato dalla derivazione potremmo scrivere $\Vdash \langle c, \sigma\rangle \longrightarrow \theta$.
Possiamo definire ora una relazione tra le derivazioni.

Definizione 6. *Siano* f, f' *derivazioni. Diciamo che* f' *è una* sottoderivazione immediata *di* f*, e scriveremo* $f' \preceq_1 f$ *se e solo se* f *è della forma* (f/y) *con* $f' \in f$*. Scriviamo* $\prec$ *per la chiusura transitiva di* $\prec_1$ *(ovvero la più piccola relazione transitiva che contenga* $\prec$*. Diciamo allora che* f' *è una* sottoderivazione propria *di* f *se e solo se* $f' \prec f$*.*

Dal momento che le derivazioni sono finite allora $\prec_1$ e $\prec$ sono ben fondate. Possiamo servirci di questa considerazione per provare che l'esecuzione dei comandi è determinstica.

4.3 La Prova del Determinismo

Proposition 4. *Per ogni* c *e per ogni* σ *se* $\langle c, \sigma\rangle \longrightarrow \theta$ *e* $\langle c, \sigma\rangle \longrightarrow \xi$*, allora* $\theta = \xi$

Dimostrazione. La prova procede per induzione sulle derivazioni per i comandi, o più precisamente per induzione ben fondata sulla relazione di sottoderivazione tra le derivazioni per l'esecuzione dei comandi. Vorremmo provare la proprietà seguente

$$P(f) \Leftrightarrow \forall c \in \mathbb{COM}, \sigma, \theta, \xi.\ f \Vdash \langle c, \sigma\rangle \longrightarrow \theta\ \&\ \langle c, \sigma\rangle \longrightarrow \xi \Rightarrow \theta = \xi$$

Per il principio di induzione ben fondata è sufficiente provare che $\forall f' \prec f.P(f')$ implica $P(f)$. Sia f una derivazione della semantica operazionale dei comandi. Assumiamo allora che $\forall f' \prec f.P(f')$. Supponiamo

$$f \Vdash \langle c, \sigma\rangle \longrightarrow \theta\ \ \&\ \ \Vdash \langle c, \sigma\rangle \longrightarrow \xi$$

Il che vuole dire che $f_1 \Vdash \langle c, \sigma\rangle \longrightarrow \xi$ per qualche f_1.

Ora consideriamo le possibili stutture di c e proviamo che $\theta = \xi$

$c \equiv$ `skip`. In questo caso

$$f = f_1 = \frac{}{\langle \texttt{skip}, \sigma\rangle \longrightarrow \sigma}$$

$c \equiv X := t$. In questo caso le derivazioni hanno una forma simile

$$f = \frac{\begin{array}{c}\vdots\\ \langle t, \sigma\rangle \longrightarrow n\end{array}}{\langle X := t, \sigma\rangle \longrightarrow \sigma[n/X]} \qquad f_1 = \frac{\begin{array}{c}\vdots\\ \langle t, \sigma\rangle \longrightarrow m\end{array}}{\langle X := t, \sigma\rangle \longrightarrow \sigma[m/X]}$$

dove $\theta = \sigma[n/X]$ e $\xi = \sigma[m/X]$. Dal momento che la valutazione di espressioni aritmetiche è deterministica $n = m$, quindi $\sigma = \theta$.

$c \equiv c; d$. In questo caso le derivazioni saranno del tipo

$$f = \dfrac{\begin{array}{cc}\vdots & \vdots \\ \langle c, \sigma\rangle \longrightarrow \theta' & \langle d, \theta'\rangle \longrightarrow \theta\end{array}}{\langle c; d, \sigma\rangle \longrightarrow \theta} \qquad f_1 = \dfrac{\begin{array}{cc}\vdots & \vdots \\ \langle c, \sigma\rangle \longrightarrow \xi' & \langle d, \xi'\rangle \longrightarrow \xi\end{array}}{\langle c; d, \sigma\rangle \longrightarrow \xi}$$

Ora, siano f' la sottoderivazione

$$\dfrac{\vdots}{\langle c, \sigma\rangle \longrightarrow \theta'}$$

e f'' la sottoderivazione

$$\dfrac{\vdots}{\langle d, \theta'\rangle \longrightarrow \theta'}$$

in f. Allora $f' \prec f$ e $f'' \prec f$, quindi $P(f')$ e $P(f'')$. Segue allora che $\theta' = \xi'$ e $\theta = \xi$. Infatti dalla sottoderivazione di $\langle c, \sigma\rangle \longrightarrow \xi'$ in f_1 sappiamo che

$$f \Vdash \langle c, \sigma\rangle \longrightarrow \sigma' \;\&\; \Vdash \langle c, \sigma\rangle \longrightarrow \xi'$$

e quindi, da $P(f)$, segue $\sigma' = \xi'$. Analogamente procediamo per provare che $\theta = \xi$, servendoci di $P(f_1)$ e della derivazione di $\langle d, \xi'\rangle \longrightarrow \theta$ in f_1.

$c \equiv \texttt{if}\ b\ \texttt{then}\ c\ \texttt{else}\ d$. La valutazione dell' `if then else` dipende dalla valutazione di b. Questa valutazione è deterministica, può essere $\langle b, \sigma\rangle \longrightarrow \mathsf{True}$ oppure $\langle b, \sigma\rangle \longrightarrow \mathsf{False}$, ma non entrambe. Consideriamo il caso in cui $\langle b, \sigma\rangle \longrightarrow \mathsf{True}$. Abbiamo:

$$f = \dfrac{\begin{array}{cc}\vdots & \vdots \\ \langle b, \sigma\rangle \longrightarrow \mathsf{True} & \langle c, \sigma\rangle \longrightarrow \theta\end{array}}{\langle \texttt{if}\ b\ \texttt{then}\ c\ \texttt{else}\ d, \sigma\rangle \longrightarrow \theta} \qquad f_1 = \dfrac{\begin{array}{cc}\vdots & \vdots \\ \langle b, \sigma\rangle \longrightarrow \mathsf{True} & \langle c, \sigma\rangle \longrightarrow \xi\end{array}}{\langle \texttt{if}\ b\ \texttt{then}\ c\ \texttt{else}\ d, \sigma\rangle \longrightarrow \xi}$$

Sia f' la sottoderivazione di $\langle c, \sigma\rangle \longrightarrow \sigma$ in f. Allora $f' \prec f$. Quindi $P(f')$. Allora, come abbiamo fatto nel caso precedente, possiamo verificare che $\theta = \xi$. Se $\langle b, \sigma\rangle \longrightarrow \mathsf{False}$ si procede in maniera simile.

$c \equiv \texttt{while}\ b\ \texttt{do}\ c$. Ancora una volta, la valutazione del comando `while do` dipende dalla valutazione di b. Sappiamo che può essere $\langle b, \sigma\rangle \longrightarrow \mathsf{True}$ oppure $\langle b, \sigma\rangle \longrightarrow \mathsf{False}$. Se $\langle b, \sigma\rangle \longrightarrow \mathsf{False}$, abbiamo

$$f = \dfrac{\dfrac{\vdots}{\langle b, \sigma\rangle \longrightarrow \mathsf{False}}}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \longrightarrow \sigma} \qquad f_1 = \dfrac{\dfrac{\vdots}{\langle b, \sigma\rangle \longrightarrow \mathsf{False}}}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \longrightarrow \sigma}$$

Quindi ovviamente abbiamo $\theta = \xi$.

Se invece $\langle b, \sigma\rangle \longrightarrow \mathsf{True}$, abbiamo

$$f = \dfrac{\begin{array}{ccc}\vdots & \vdots & \vdots \\ \langle b, \sigma\rangle \longrightarrow \mathsf{True} & \langle c, \sigma\rangle \longrightarrow \theta' & \langle \texttt{while}\ b\ \texttt{do}\ c, \theta'\rangle \longrightarrow \theta\end{array}}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \longrightarrow \theta}$$

$$f_1 = \dfrac{\begin{array}{ccc}\vdots & \vdots & \vdots \\ \langle b, \sigma\rangle \longrightarrow \mathsf{True} & \langle c, \sigma\rangle \longrightarrow \xi' & \langle \texttt{while}\ b\ \texttt{do}\ c, \xi'\rangle \longrightarrow \xi\end{array}}{\langle \texttt{while}\ b\ \texttt{do}\ c, \xi\rangle \longrightarrow \xi}$$

Sia f' la sottoderivazione di $\langle c, \sigma\rangle \longrightarrow \theta'$ e f'' la sottoderivazione di $\langle \texttt{while}\ b\ \texttt{do}\ c, \theta'\rangle \longrightarrow \theta$ in f. Allora $f' \prec f$ e $f'' \prec f$, quindi $P(f')$ e $P(f'')$. Segue quindi, procedendo come negli altri casi, che $\theta' = \xi'$ e quindi $\theta = \xi$.

In definitiva abbiamo provato, per tutti i casi, che

$$f \Vdash \langle c, \sigma\rangle \longrightarrow \theta \ \ \& \ \ \Vdash \langle c, \sigma\rangle \longrightarrow \xi \text{ implica che } \theta = \xi$$

.

Per il principio dell'induzione ben fondata abbiamo quindi che $P(f)$ vale per tutte le derivazioni f per le esecuzioni dei comandi. Questo è equivalente a

$$P(f) \Leftrightarrow \forall c \in \mathbb{COM}, \sigma, \theta, \xi.\ f \Vdash \langle c, \sigma\rangle \longrightarrow \theta\ \&\ \langle c, \sigma\rangle \longrightarrow \xi \Rightarrow \theta = \xi$$

□

4.4 Esercizi

Esercizio 5. *Si provi per induzione strutturale, che la valutazione delle espressioni aritmetiche termina sempre, ovvero, per tutte le espressioni aritmetiche t e gli stati σ esiste un m tale che $\langle t, \sigma\rangle \longrightarrow m$*

Esercizio 6. *Usando il risultato dell'esercizio 5, si provi per induzione strutturale che la valutazione delle espressioni booleane termina sempre.*

Esercizio 7. *Si estenda il lingugaggio IMP, con un' ulteriore espressione aritmetica, in maniera tale che il determinismo della semantica operazionale non valga più.*

5 La Semantica Assiomatica

Consideriamo il problema di stabilire se un programma in **IMP** esegue il compito per cui è stato scritto. Ripensiamo al nostro programma di esempio, $c_{EUCLIDE}$. Chi ci garantisce che questo calcoli il massimo comun divisore tra due interi dati in input? Abbiamo visto che una possibile risposta può essere data grazie alle regole di valutazioneo della semantica operazionale. Una soluzione alternativa può essere ottenuta ricorrendo ad una semantica diversa, detta *assiomatica*.

Prima di procedere con la definizione di questa nuova semantica introduciamo alcune notazioni. Costruiremo un sistema di prove ottenuto a partire da *asserzioni* della forma

$$\{A\}c\{B\}$$

dove A e B sono asserzioni come quelle che abbiamo visto in $\mathbb{BEXP}$ e c è un comando. Con questa asserzione composta intendiamo: "per ogni stato σ che soddisfa A, se l'esecuzione c dallo stato σ termina nello stato θ, allora θ soddisfa B."

L'asserzione A è detta la *precondizione* e B la *postcondizione* dell'*asserzione di correttezza parziale* $\{A\}c\{B\}$. Parliamo di correttezza parziale perchè $\{A\}c\{B\}$ non ci fornisce informazioni sul comando c nel caso in cui questo non termini. Infine, per indicare che uno stato σ soddisfa un'asserzione A, o, equivalentemente, che l'asserzione A è vera nello stato σ, scriveremo

$$\sigma \vDash A$$

Nel caso in cui un'asserzione A sia vera per ogni stato σ scriveremo invece

$$\vDash A$$

.

Allora l'asserzione di correttezza parziale $\{A\}c\{B\}$ significherà:

$$\forall \sigma, \theta.(\sigma \vDash A \wedge \langle c, \sigma\rangle \longrightarrow \theta) \Rightarrow \theta \vDash B$$

5.1 Il Linguaggio delle Asserzioni $\mathbb{ASSN}$

Dal momento che vogliamo intervenire su espressioni booleane, dobbiamo includere $\mathbb{BEXP}$ nel linguaggio delle asserzioni. Inoltre sarà utile poter formulare asserzioni che contengano quantificatori "$\forall \cdots$" e "$\exists \cdots$". Per fare ciò dobbiamo estendere $\mathbb{BEXP}$ e $\mathbb{AEXP}$ con variabili intere a cui applicare i qunatificatori. Quindi innanzitutto estendiamo $\mathbb{AEXP}$ con variabili intere $i, j, k...$, prese da un insieme $\mathbb{INTVAR}$, ottenendo così l'insieme $\mathbb{AEXPV}$ definito dalla seguente grammatica

$$t ::= n \mid X \mid i \mid t + u \mid t - u \mid t \times u$$

Estendiamo quindi l'insieme delle espressioni booleane, in maniera che questo includa le nuove variabili aritmetiche e i quantificatori, ottendendo

$$A ::= \mathsf{True} \mid \mathsf{False} \mid t = u \mid t \leq u \mid \neg A \mid A \vee B \mid A \wedge B \mid A \Rightarrow B \mid \forall i.A \mid \exists i.A$$

Abbiamo così definito il linguaggio delle asserzioni booleane estese $\mathbb{ASSN}$.

5.2 Variabili Libere e Variabili Legate

Diciamo che l'occorrenza di una variabile intera i in un'asserzione è e*legata* se occorre nel campo di un quantificatore $\forall i$ o $\exists i.$. In caso contrario si dice *libera*. Ad esempio nell'asserzione $\exists i.j = i \times k$, l'occorrennza della variabile i è legata, mentre le occorrenze di j e k sono libere.

In maniera più formale possiamo definire l'insieme $FV(t)$ delle variabili libere di un'espressione aritmetica, procedendo per induzione strutturale

$$\begin{aligned} FV(n) = FV(X) &= \emptyset \\ FV(i) &= \{i\} \\ FV(t+u) = FV(t-u) = FV(t \times u) &= FV(t) \cup FV(u) \end{aligned}$$

per ogni $n \in \mathbb{Z}, i \in \mathbb{INTVAR}, t, u \in \mathbb{AEXP}$

Analogamente definiamo l'insieme delle variabili libere $FV(A)$

$$\begin{aligned} FV(\mathsf{True}) = FV(\mathsf{False}) &= \emptyset \\ FV(t = u) = FV(t \leq u) &= FV(t) \cup FV(u) \\ FV(A \wedge B) = FV(A \vee B) = FV(A \Rightarrow B) &= FV(A) \cup FV(B) \\ FV(\neg A) &= FV(A) \\ FV(\forall i.A) = FV(\exists i.A) &= FV(A) \setminus \{i\} \end{aligned}$$

per ogni $t, u \in \mathbb{AEXP}, i \in \mathbb{INTVAR}, A, B \in \mathbb{ASSN}$.

Quindi abbiamo dato una definizione formale di variabile libera. Una variabile che occorre non è libera in un'asserzione, si dirà quindi legata. Un'asserzione senza variabili libere si dice *chiusa*.

5.3 Sostituzione

Sia i una variabile intera e t un'espressione aritmetica senza variabili intere. Possiamo definire la sostituzione per le espressioni aritmetiche tramite la seguente induzione strutturale

$$\begin{aligned} n[t/i] \equiv n \quad X[t/i] &\equiv X \\ j[t/i] \equiv j \quad i[t/i] &\equiv t \\ (u+v)[t/i] &\equiv (t[t/i] + u[t/i]) \\ (v-u)[t/i] &\equiv (v[t/i] - u[t/i]) \\ (v \times u)[t/i] &\equiv (v[t/i] \times u[t/i]) \end{aligned}$$

dove $n \in \mathbb{Z}, X \in \mathbb{L}, i, j \in \mathbb{INTVAR}, i \not\equiv j$ e $u, v \in \mathbb{AEXPV}$.

Definiamo ora la sostituzione di t ad i nelle asserzioni per induzione strutturale (ricordiamo che t non deve avere variabili libere)

$$\begin{array}{l}
\mathsf{True}[t/i] \equiv [t/i] \quad \mathsf{False}[t/i] \equiv [t/i] \\
(u = v)[t/i] \equiv (u[t/i] = v[t/i]) \quad (u \leq v)[t/i] \equiv (u[t/i]) \leq v[t/i]) \\
(A \wedge B)[t/i] \equiv (A[t/i] \wedge B[t/i]) \quad (A \vee B)[t/i] \equiv (A[t/i] \vee B[t/i]) \\
(A \Rightarrow B)[t/i] \equiv (A[t/i] \Rightarrow B[t/i]) \quad (\neg A)[t/i] \equiv \neg(A[t/i]) \\
(\forall j.A)[t/i] \equiv \forall j.(A[t/i]) \quad (\exists j.A)[t/i] \equiv \exists j.(A[t/i]) \\
(\forall i.A)[t/i] \equiv \forall i.A \quad (\exists i.A)[t/i] \equiv \exists i.A
\end{array}$$

dove $u, v \in \mathbb{AEXPV}, i, j \in \mathbb{INTVAR}, i \not\equiv j, A, B \in \mathbb{ASSN}$.

Sottolineiamo che abbiamo assunto che t non abbia variabili libere, in caso contrario, il processo di sostituzione si complica, dal momento che si dovrebbero rinominare le variabili legate.

5.4 Regole per la Correttezza Parziale

Definiamo ora le regole per la generazione di asserzioni parzialmente corrette. Queste regole sono definite in termini sintattici. Le regole sono dette *regole di Hoare* e il sistema di regole è detto *logica di Hoare*.

$$\frac{}{\{A\}\texttt{skip}\{A\}} \qquad \frac{}{\{B[t/X]\}X := t\{B\}}$$

$$\frac{\{A\}c\{C\} \quad \{C\}d\{B\}}{\{A\}c; d\{B\}} \qquad \frac{\{A \wedge b\}c\{B\} \quad \{A \wedge \neg b\}d\{B\}}{\{A\}\texttt{if } b \texttt{ then } c \texttt{ else } d\{B\}}$$

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\texttt{while } b \texttt{ do } c\{A \wedge \neg b\}} \qquad \frac{\vDash (A \Rightarrow A') \quad \{A'\}c\{B'\} \quad \vDash (B \Rightarrow B')}{\{A\}c\{B\}}$$

Dal momento che siamo in presenza di regole, possiamo parlare di derivazione per le regole di Hoare. In questo contesto le regole di Hoare sono pensate come un sistema di prove, le derivazioni sono dette *prove* e ogni conclusione di una derivazione è detta *teorema*. Scriveremo $\vdash \{A\}c\{B\}$ se $\{A\}c\{B\}$ è un teorema.

Prestiamo ora particolare attenzione alla regola per il costrutto `while`. L'asserzione A è detta *invariante* perchè la premessa, $A \wedge b$, ci dice che A è preservata dall'esecuzione di un intero corpo del *while* e che l'esecuzione del ciclo avviene solo a partire da stati che soddisfano b. A partire da uno stato che soddisfa A, o l'esecuzione del ciclo diverge, oppure si ha un numero finito di esecuzioni del corpo del ciclo, ognuna delle quali inizia in uno stato che soddisfa b. In quest'ultimo caso, lo stato finale soddisfa sia A, dal momento che questo è invariante, sia $\neg b$, uscendo quindi dal ciclo.

La regola sulla conseguenza è una regola particolare, dal momento che le premesse includono implicazioni valide. Ovvero, l'istanza di una regola di conseguenza ha, tra le sue premesse, una della forma $\vDash (A \Rightarrow A)$ e una della forma $\vDash (B \Rightarrow B')$. Quindi per potersi servire della regola è necessario provare le asserzioni $(A \Rightarrow A)$ e $(B \Rightarrow B')$.

Diamo ora una prova di correttezza del nostro programma di esempio $c_{EUCLIDE}$ servendoci delle regole appena definite. Ricordiamo che

$$c_{EUCLIDE} \equiv \texttt{while } \neg(X = Y) \texttt{ do } (\texttt{if } X \leq Y \texttt{ then } (Y := Y - X) \texttt{ else } (X := X - Y))$$

Quello che vorremmo provare è che, $c_{EUCLIDE}$ calcola effettivamente il gcd. Più precisamente proviamo la seguente proposizione.

Proposition 5.

$$\vdash \{X = n \wedge Y = m \wedge n \geq 1 \wedge m \geq 1\}c_{EUCLIDE}\{X = \gcd(n, m)\}$$

Dimostrazione. Essendo in presenza di un costrutto `while`, dalle regole per la correttezza parziale, sappiamo che dobbiamo definire un oppportuno invariante. Vorremmo definire I tale che

$$\{I\}c_{EUCLIDE}\{I \wedge (X \neq Y)\}$$

Sia allora

$$I \equiv \{(\gcd(X,Y) = \gcd(n,m)) \wedge (X \geq 1) \wedge (Y \geq 1)\}$$

Consideriamo ora i comandi di assegnazione all'interno del comando `if`. Dalle regole di assegnazione sappiamo che

$$\{I[Y - X/Y]\}Y := Y - X\{I\}$$

dove $I[Y - X/Y] \equiv ((\gcd(X, Y - X) = \gcd(n,m)) \wedge (X \geq 1) \wedge (Y - X \geq 1))$. Analogamente abbiamo

$$\{I[X - Y/X]\}X := X - Y\{I\}$$

dove $I[X - Y/] \equiv ((\gcd(Y, X - Y) = \gcd(n,m)) \wedge (X \geq 1) \wedge (X - Y \geq 1))$

D'altronde, da quanto provato in precedenza sul gcd abbiamo la seguente implicazione:

$$I \wedge (X \neq Y) \wedge (X < Y) = (\gcd(X,Y) = \gcd(n,m) \wedge (X \geq 1) \wedge (Y \geq 1)) \wedge (X \neq Y) \wedge (X < Y) \Rightarrow$$
$$\gcd(X, Y - X) = \gcd(n,m) \wedge (X \geq 1) \wedge (Y - X \geq 1) \equiv I[Y - X/X]$$

Quindi, dalle regole sulla conseguenza abbiamo

$$\{I \wedge (X \neq Y) \wedge (X < Y)\}Y := Y - X\{I\}$$

Allo stesso modo avremo

$$\{I \wedge (X \neq Y) \wedge (X \not< Y)\}X := X - Y\{I\}$$

Allora, dalle regole sull'`if`

$$\vdash \{I \wedge (X \neq Y)\}\texttt{if } X \leq Y \texttt{ then } (Y := Y - X) \texttt{ else } (X := X - Y)\{I\}$$

Ma allora I è invariante per il corpo c del ciclo. Quindi, dalle regole sul `while`

$$\vdash \{I\}c_{EUCLIDE}\{I \wedge (X \neq Y)\}$$

Ora, osserviamo che

$$\vDash (X = n) \wedge (Y = m) \wedge (n \geq 1) \wedge (m \geq 1) \Rightarrow I$$

e anche

$$\vDash I \wedge (X = Y) \Rightarrow X = Y = \gcd(X,Y) = \gcd(n,m)$$

Quindi dalla regola sulla conseguenza logica abbiamo che

$$\{X = n \wedge Y = m \wedge n \geq 1 \wedge m \geq 1\}c_{EUCLIDE}\{X = \gcd(n,m)\}$$

□

5.5 Esercizi

Esercizio 8. *Si scriva un'asserzione $A \in \mathbb{ASSN}$ con una variabile libera i, che esprima che i è un numero primo, ovvero tale che: $\sigma \vDash A$ sse i è un numero primo*

Esercizio 9. *Sia*

$$w \equiv X := 0; Y := 1; \texttt{while } Y \leq V \texttt{ do } X := X + T; Y := Y + 1$$

Usando le regole di Hoare si provi la correttezza di

$$\{1 \leq V\}w\{U = T \times V\}$$

Esercizio 10. *Sia*

$$w \equiv \texttt{while } \neg(Y = 0) \texttt{ do } Y := Y - 1; X := 2 \times X$$

Usando ole regole di Hoare si provi che

$$\{i = Y\}w\{Y = 2^i\}$$

6 La Semantica Denotazionale

Nelle prime sezioni, abbiamo definito il comportamento dei programmi in **IMP** secondo un metodo operazionale, definendo induttivamente relazioni di transizione per eprimere valutazioni ed esecuzioni. Questo tipo di semantica, basata essenzialmente sulla sintassi, rende difficile comparare due programmi scritti in diversi linguaggi di programmazione.

Riconsideriamo ora l'esempio di equivalenza tra comandi che abbiamo analizzato in precedenza (tra `while` b `do` c e `if` b `then` $c; w$ `else skip`). Generalizzando possiamo dire che due comandi c, d sono equivalenti sse

$$\forall \sigma, \theta.(\langle c, \sigma\rangle \longrightarrow \theta \quad sse \ \langle d, \sigma\rangle \longrightarrow \theta)$$

Ma allora notiamo che c e d sono equivalenti se e solo se

$$\{(\sigma, \theta) \mid \langle c, \sigma\rangle \longrightarrow \theta\} = \{(\sigma, \theta) \mid \langle d, \sigma\rangle \longrightarrow \theta\}$$

ovvero due comandi sono equivalenti se determinano la stessa funzione parziale sugli stati. Questa osservazione ci suggerisce che possiamo definire una semantica di **IMP** ad un livello *più astratto*, considerando la *denotazione* di un comando come una funzione parziale tra gli stati. Ovvero, possiamo definire una *semantica denotazionale*.

Per poter procedere in questa direzione abbiamo bisogno di definire delle funzioni semantiche. Chiameremo Σ l'insieme dei possibili stati. Definiremo

$$\mathcal{A} : \mathbb{AEXP} \to (\Sigma \to \mathbb{Z})$$

$$\mathcal{B} : \mathbb{BEXP} \to (\Sigma \to \mathbb{B})$$

$$\mathcal{C} : \mathbb{COM} \to (\Sigma \to \Sigma)$$

per induzione strutturale. Così ad esempio, per ogni comando c consideremo la funzione parziale $\mathcal{C}[\![c]\!]$. Tale funzione è una *denotazione* di c (diremo anche c *denota* $\mathcal{C}[\![c]\!]$). Che cosa intendiamo quando affermiamo che procederemo per induzione strutturale? Vuol dire che definiremo $\mathcal{C}[\![c]\!]$ a partire dalla denotazione $\mathcal{C}[\![c']\!]$, dei sottocomandi c' di c. In questo modo otteniamo una semantica *composizionale*, abbiamo in maniera modulare, dal momento che la denotazone di un comando è costruita a partire dalle denotazioni dei suoi sottocomandi.

6.1 Denotazioni di $\mathbb{AEXP}$

Innanzitutto definiamo per induzione strutturale la denotazione di un'espressione aritmetica, intendendola come una relazione tra stati e numeri:

$$\begin{aligned}
\mathcal{A}[\![n]\!] &= \{(\sigma, n) \mid \sigma \in \Sigma\} \\
\mathcal{A}[\![X]\!] &= \{(\sigma, \sigma(X)) \mid \sigma \in \Sigma\} \\
\mathcal{A}[\![t + u]\!] &= \{(\sigma, n + m) \mid (\sigma, n) \in \mathcal{A}[\![t]\!] \ \&\ (\sigma, m) \in \mathcal{A}[\![u]\!]\} \\
\mathcal{A}[\![t - u]\!] &= \{(\sigma, n - m) \mid (\sigma, n) \in \mathcal{A}[\![t]\!] \ \&\ (\sigma, m) \in \mathcal{A}[\![u]\!]\} \\
\mathcal{A}[\![t \times u]\!] &= \{(\sigma, n \times m) \mid (\sigma, n) \in \mathcal{A}[\![t]\!] \ \&\ (\sigma, m) \in \mathcal{A}[\![u]\!]\}
\end{aligned}$$

Possiamo dimostrare, per induzione strutturale, che ogni $\mathcal{A}[\![t]\!]$ è in effetti una funzione. Evidenziamo qui che i simboli $+, -, \times$ che compaiono a sinistra delle definizioni sono da considerarsi come simboli sintattici in **IMP**, mentre quelli che compaiono a destra rappresentano le usuali operazioni tra numeri.

6.2 Denotazioni di $\mathbb{BEXP}$

Definiamo per induzione strutturale la denotazione di un'espressione booelana, come una relazione tra stati e valori di verità.

$$\begin{aligned}
\mathcal{B}[\![\mathsf{True}]\!] &= \{(\sigma, \mathsf{True}) \mid \sigma \in \Sigma\} \\
\mathcal{B}[\![\mathsf{False}]\!] &= \{(\sigma, \mathsf{False}) \mid \sigma \in \Sigma\} \\
\mathcal{B}[\![t = u]\!] &= \{(\sigma, \mathsf{True}) \mid \sigma \in \Sigma \;\&\; \mathcal{A}[\![t]\!]\sigma = \mathcal{A}[\![u]\!]\sigma\} \cup \{(\sigma, \mathsf{False}) \mid \sigma \in \Sigma \;\&\; \mathcal{A}[\![t]\!]\sigma \neq \mathcal{A}[\![u]\!]\sigma\} \\
\mathcal{B}[\![t \leq u]\!] &= \{(\sigma, \mathsf{True}) \mid \sigma \in \Sigma \;\&\; \mathcal{A}[\![t]\!]\sigma \leq \mathcal{A}[\![u]\!]\sigma\} \cup \{(\sigma, \mathsf{False}) \mid \sigma \in \Sigma \;\&\; \mathcal{A}[\![t]\!]\sigma \not\leq \mathcal{A}[\![u]\!]\sigma\} \\
\mathcal{B}[\![\neg b]\!] &= \{(\sigma, \mathsf{NEG}(t) \mid \sigma \in \Sigma \;\&\; (\sigma, t) \in \mathcal{B}[\![b]\!]\} \\
\mathcal{B}[\![\neg b]\!] &= \{(\sigma, \mathsf{NEG}(t) \mid \sigma \in \Sigma \;\&\; (\sigma, t) \in \mathcal{B}[\![b]\!]\} \\
\mathcal{B}[\![b \wedge a]\!] &= \{(\sigma, \mathsf{AND}(t_1, t_2) \mid \sigma \in \Sigma \;\&\; (\sigma, t_1) \in \mathcal{B}[\![b]\!] \;\&\; (\sigma, t_2) \in \mathcal{B}[\![a]\!]\} \\
\mathcal{B}[\![b \vee a]\!] &= \{(\sigma, \mathsf{OR}(t_1, t_2) \mid \sigma \in \Sigma \;\&\; (\sigma, t_1) \in \mathcal{B}[\![b]\!] \;\&\; (\sigma, t_2) \in \mathcal{B}[\![a]\!]\}
\end{aligned}$$

Prima di procedere nella definzione delle denotazioni per i comandi, introduciamo la nozione di *minimo punto fisso di un operatore.*

6.3 Operatori e Minimo Punto Fisso

Consideriamo un insieme di istanze di regole R. Indicheremo con I_R, *l'insieme definito a da* R, che contiene tutti e solo gli elementi x per i quali esiste una derivazione. Ovvero

$$I_R = \{x \mid \vDash_R x\}$$

Possiamo ora introdurre un principio che permette di verificare che una proprietà è valida per tutti gli elementi di un insieme definito da determinate regole. Ovvero, definiamo il *principio di induzione sulle regole.* Tale principio è basato sull'idea che, se una proprietà è preservata nel passaggio dalle premesse alla conclusione di *tutte* le istanze di regole in una derivazione, allora la proprietà è valida per la conclusione della derivazione, quindi la proprietà è valida tutti gli elementi dell'insieme definito dalle regole. Più precisamente diamo la seguente definizione.

Definizione 7 (Principio generale dell'induzione sulle regole)**.** *Sia I_R l'insieme definito dalle istanze di regole . Sia P una proprietà. Allora $\forall x \in I_R.P(x)$ se e solo se per tutte le istanze di regole (S/y) in R tali che $S \subseteq I_R$ si ha che*

$$(\forall x \in S.P(x)) \Rightarrow P(y))$$

Definizione 8. *Diciamo che un insieme Q è* chiuso *rispetto alle istanze di regole R, o R-*chiuso*, se e solo se per ogni istanza di regola (S/y)*

$$S \subseteq Q \Rightarrow y \in Q$$

In altre parole, un insieme è chiuso rispetto alle istanze di regole se, per ogni istanza di regola, se le sue premesse sono nell'insieme considerato, allora lo è anche la conclusione. Si può provare che I_R è il *più piccolo* insieme chiuso rispetto ad R. Ovvero, vale la seguente proposizione.

Proposition 6. *Rispetto all'insieme di istanze di regole R si ha che*
- *I_R è R-chiuso*
- *se Q è un insieme R-chiuso allora $I_R \subseteq Q$*

Come possiamo costruire l'insieme I_R? Cerchiamo di dare una definizione alternativa di insieme definito a partire da R. Osserviamo innanzitutto che R definisce a sua volta un operatore $\widehat{R}$ sugli insiemi, tale che, se B è un insieme allora

$$\widehat{R}(B) = \{y \mid \exists S \subseteq B.(S/y) \in R\}.$$

Allora, possiamo dare una definizione equivalente di insieme R-chiuso.

Proposition 7. *Un insieme B è R-chiuso se e solo se $\widehat{R}(B) \subseteq B$*

Notiamo poi che $\widehat{R}$ è *monotono* rispetto alla relazione di inclusione, ovvero

$$A \subseteq B \Rightarrow \widehat{R}(A) \subseteq \widehat{R}(B).$$

Ora, se applichiamo ripetutamente $\widehat{R}$ all'insieme vuoto $\emptyset$ otteniamo una successione di insiemi

$$A_0 = \widehat{R}^0(\emptyset) = \emptyset$$

$$A_1 = \widehat{R}^1(\emptyset) = \widehat{R}(\emptyset)$$

$$A_2 = \widehat{R}^2(\emptyset) = \widehat{R}(\widehat{R}(\emptyset))$$

$$\vdots$$

$$A_n = \widehat{R}^n(\emptyset)$$

$$\vdots$$

Osserviamo che l'insieme A_1 è costituito da tutte le conclusioni delle istanze di assiomi, mentre in generale A_{n+1} è costituito da tutti e soli gli elementi che seguono direttamente dalle istanze di regole con premesse in A_n.

Ora, ovviamente $\emptyset \subseteq \widehat{R}(\emptyset)$, quindi $A_0 \subseteq A_1$. Dalla monotonia di $\widehat{R}$ segue allora che $A_1 = \widehat{R}(A_0) \subseteq \widehat{R}(A_1)) = A_2$. Quindi $A_1 \subseteq A_2$. Allo stesso modo si verifica che $A_2 \subseteq A_3$ e così via. Otteniamo allora una catena crescente di inclusioni

$$A_0 \subseteq A_1 \subseteq \cdots \subseteq A_n \subseteq \cdots$$

Consideriamo adesso $A = \bigcup_{n\in\mathbb{N}} A_n$. Vale la proposizione seguente.

Proposition 8. *1) A è R-chiuso.*
2) $\widehat{R}(A) = A$
3) A è il più piccolo insieme R-chiuso.

Da 1) e da 3) segue allora che $A = I_R$. D'altro canto 2) ci dice che I_R è un punto fisso di $\widehat{R}$. Inoltre da 3) sappiamo che I_R è il *minimo* punto fisso di $\widehat{R}$, ovvero

$$\widehat{R}(B) = B \Rightarrow I_R \subseteq B$$

dal momento che, per la Prosizione 7, se B è punto fisso allora è R-chiuso. Quindi dalla minimalità di I, segue che $I_R \subseteq B$. Ricapitolando, I_R è il minimo punto fisso di $\widehat{R}$, che denoteremo $fix(\widehat{R})$, ottenuto come

$$fix(\widehat{R}) = \bigcup_{n\in\mathbb{N}} \widehat{R}^n(\emptyset).$$

6.4 Denotazioni di $\mathbb{COM}$

Innanzitutto definiamo le denotazioni come relazioni tra stati. Avremo

$\mathcal{C}[\![\texttt{skip}]\!] = \{(\sigma, \sigma) \mid \sigma \in \Sigma\}$
$\mathcal{C}[\![X := t]\!] = \{(\sigma, \sigma[n/X]) \mid \sigma \in \Sigma \;\&\; n = \mathcal{A}[\![t]\!]\sigma\}$
$\mathcal{C}[\![c; d]\!] = \mathcal{C}[\![d]\!] \circ \mathcal{C}[\![c]\!]$
$\mathcal{C}[\![\texttt{if } b \texttt{ then } c \texttt{ else } d]\!] = \{(\sigma, \theta) \mid \mathcal{B}[\![b]\!]\sigma = \mathsf{True} \;\&\; (\sigma, \theta) \in \mathcal{C}[\![c]\!]\} \cup \{(\sigma, \theta) \mid \mathcal{B}[\![b]\!]\sigma = \mathsf{False} \;\&\; (\sigma, \theta) \in \mathcal{C}[\![d]\!]\}$

dove $\circ$ rappresenta la composizione di relazioni.

Il problema sorge quando vogliamo definre una denotazione per `while do`. Sia infatti, $w \equiv$ `while` b `do`c. Abbiamo provato che w è equivalente a `if` b `then` $c; w$ `else`. Quindi ci aspetteremmo che $\mathcal{C}[\![w]\!]$ coincida con la funzione parziale $\mathcal{C}[\![$`if` b `then` c; `else skip`$]\!]$. Dovremmo avere allora

$$\begin{aligned}\mathcal{C}[\![w]\!] &= \{(\sigma,\theta) \mid \mathcal{B}[\![b]\!]\sigma = \mathsf{True} \;\&\; (\sigma,\theta) \in \mathcal{C}[\![c; w]\!]\} \cup \{(\sigma,\sigma) | \mathcal{B}[\![b]\!]\sigma = \mathsf{False}\} \\ &= \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!]\sigma = \mathsf{True} \;\&\; (\sigma,\theta) \in \mathcal{C}[\![w]\!] \circ \mathcal{C}[\![c]\!]\} \cup \{(\sigma,\theta) \mid \mathcal{B}[\![b]\!]\sigma = \mathsf{False}\}\end{aligned}$$

Se scriviamo φ per $\mathcal{C}[\![w]\!]$, β per $\mathcal{B}[\![b]\!]$ e γ per $\mathcal{C}[\![c]\!]$, otteniamo allora

$$\varphi = \{(\sigma,\theta) \mid \beta(\sigma) = \mathsf{True} \;\&\; (\sigma,\theta) \in \varphi \circ \gamma\} \cup \{(\sigma,\sigma) \mid \beta(\sigma) = \mathsf{False}\}$$

Notiamo però che la definzione di questa funzione parziale è una definizione circolare: utilizza φ da entrambe le parti dell'uguaglianza. Abbiamo bisogno di qualche tecnica che permetta di risolvere un'equazione *ricorsiva* di questa forma. Possiamo riscrivere l'equazione in questo modo

$$\begin{aligned}\Gamma(\varphi) &= \{(\sigma,\theta) \mid \beta(\sigma) = \mathsf{True} \;\&\; (\sigma,\theta) \in \varphi \circ \gamma\} \cup \{(\sigma,\sigma) \mid \beta(\sigma) = \mathsf{False}\} \\ &= \{(\sigma,\theta) \mid \exists\xi.\beta(\sigma) = \mathsf{True} \;\&\; (\sigma,\xi) \in \gamma \;\&\; (\xi,\theta) \in \varphi\} \cup \{(\sigma,\sigma) \mid \beta(\sigma) = \mathsf{False}\}\end{aligned}$$

Otteniamo quindi una funzione che data φ restituisce $\Gamma(\varphi)$. Quello che vorremmo trovare allora è φ tale che

$$\varphi = \Gamma(\varphi),$$

ovvero vorremmo individuare φ *punto fisso* di Γ.

Possiamo servirci allora delle nozione di punto fisso introdotta nelle sezione precedente. Per procedere in questa direzione abbiamo quindi biosgno di definire un opportuno insieme di istanze di regole. Sia

$$R = \{(\{(\xi,\theta)\}/(\sigma,\theta)) \mid \beta(\sigma) = \mathsf{True} \;\&\; (\sigma,\xi) \in \gamma)\} \cup \{(\emptyset/(\sigma,\sigma)) \mid \beta(\sigma) = \mathsf{False}\}$$

Si verifica facilmente che l'operatore $\widehat{R}$ coincide con Γ. D'altronde abbiamo provato che $\widehat{R}$ ammette un minimo punto fisso $\varphi = fix(\widehat{R})$; quindi φ sarà punto fisso di Γ.

Siamo ora in grado di definire una semantica denotazionale per il `while do`. Abbiamo

$$[\![\texttt{while}\ b\ \texttt{do}\ c]\!] = fix(\Gamma)$$

dove

$$\Gamma(\varphi) = \{(\sigma,\theta) \mid \mathcal{B}[\![b]\!]\sigma = \mathsf{True} \;\&\; (\sigma,\theta) \in \varphi \circ \mathcal{C}[\![c]\!]\} \cup \{(\sigma,\sigma) \mid \beta(\sigma) = \mathsf{False}\}$$

Abbiamo motivato la definizione della denotazione del comando `while do`, basandoci sull'equivalenza a livello operazionale tra $w \equiv$ `while` b `do` c e `if` b `then` $c; w$ `else`. Proviamo allora che questa equivalenza vale effittivamente nell'ambito della semantica denotazionale. Ovvero proviamo la seguente proposizione.

Proposition 9. $\mathcal{C}[\![w]\!] = \mathcal{C}[\![$`if` b `then` c; `else skip`$]\!]$

Dimostrazione.

$$\begin{aligned}\mathcal{C}[\![w]\!] &= \Gamma(\mathcal{C}[\![w]\!]) \\ &= \{(\sigma,\theta) \mid \mathcal{B}[\![b]\!]\sigma = \mathsf{True} \;\&\; (\sigma,\theta) \in \mathcal{C}[\![w]\!] \circ \mathcal{C}[\![c]\!]\} \cup \{(\sigma,\sigma) \mid \beta(\sigma) = \mathsf{False}\} \\ &= \{(\sigma,\theta) \mid \mathcal{B}[\![b]\!]\sigma = \mathsf{True} \;\&\; (\sigma,\theta) \in \mathcal{C}[\![c; w]\!] \cup \{(\sigma,\sigma) \mid \beta(\sigma) = \mathsf{False} \;\&\; (\sigma,\theta) \in \mathcal{C}[\![\texttt{skip}]\!]\} \\ &= \mathcal{C}[\![\texttt{if}\ b\ \texttt{then}\ c;\ \texttt{else skip}]\!]\end{aligned}$$

dove nel terzo passaggio ci siamo basati sulle definizioni delle denotazioni per la sequenza di comandi e per lo `skip`. Nell'ultimo passaggio invece si applica la definizione di denotazione per il comando `if then else`. □

6.5 Ordini Parziali Completi e Funzioni Continue

Per definire una denotazione per il costrutto `while`, abbiamo introdotto la nozione di minimo punto fisso di un operatore su insiemi. In realtà, per poter lavorare con definizioni ricorsive, è spesso utile servirsi delle nozioni più astratte di *ordine parziale completo* e di *funzione continua*.

Se ripensiamo alla definizione di minimo punto fisso data in precedenza, ci rendiamo conto che l'idea di minimalità era legata più che altro alla relazione di inclusione tra insiemi. Possiamo invece introdurre la nozione più generale di *ordine parziale*.

Definizione 9. *Definiamo* ordine parziale *un insieme P sul quale è definita una relazione binaria $\sqsubseteq$ che sia*

- *riflessiva:* $\forall p \in P.\ p \sqsubseteq p$
- *transitiva:* $\forall p, q, r \in P.\ p \sqsubseteq q\ \&\ q \sqsubseteq r \Rightarrow p \sqsubseteq r$
- *antisimmetrica:* $\forall p, q \in P.\ p \sqsubseteq q\ \&\ q \sqsubseteq p \Rightarrow p = q$

Torniamo nuovamente alla costruzione del minimo punto fisso illustrata in precedenza. A partire dalla catena $A_0 \subseteq A_1 \subseteq \cdots \subseteq A_n \subseteq \cdots$, con $A_0 = \emptyset$, definivamo l'unione $A = \bigcup_{n\in\mathbb{N}} A_n$. Possiamo generalizzare l'unione di insiemi ordinati per inclusione servendoci della nozione di *estremo superiore*.

Definizione 10. *Sia $(P, \sqsubseteq)$ un ordine parziale e $X \subseteq P$, diciamo che p è un* maggiorante *di X se e solo se*

$$\forall q \in X.\ q \sqsubseteq p$$

Diciamo che p è estremo superiore di X se e solo se

- *p è un maggiorante di X;*
- *per ogni maggiorante q di X, $p \sqsubseteq q$.*

Scriveremo $\bigsqcup X$ per indicare l'estremo superiore di X. Scriveremo $d_1 \sqcup \cdots \sqcup d_n$ per indicare $\bigsqcup\{d_1, \cdots d_m\}$

In particolar modo siamo interessati agli ordini parziali *completi*.

Definizione 11. *Sia $(D, \sqsubseteq_D)$ un ordine parziale. Una $\mathbb{N}$-catena di un ordine parziale è una catena crescente $d_0 \sqsubseteq_D d_1 \sqsubseteq_D \cdots \sqsubseteq_D d_n \sqsubseteq_D \cdots$ di elementi dell'ordine parziale.*

L'ordine parziale $(D, \sqsubseteq_D)$ è un ordine parziale completo *(abbreviato cpo), se ha un estremo superiore per ogni $\mathbb{N}$-catena $d_0 \sqsubseteq_D d_1 \sqsubseteq_D \cdots \sqsubseteq_D d_n \sqsubseteq \cdots$, i.e. ogni catena crescente $\{d_n \mid n \in \mathbb{N}\}$ di elementi in D ha un estremo superiore $\bigsqcup\{d_n \mid n \in \mathbb{N}\}$, indicato anche come $\bigsqcup_{n\in\mathbb{N}} d_n$. Diciamo che $(D, \sqsubseteq_D)$ è un cpo con bottom se ha un elemento minimo $\bot_D$.*

Ritorniamo a considerare l'operatore $\widehat{R}(B)$. Sappiamo che tale operatore è monotono, ovvero

$$A \subseteq B \Rightarrow \widehat{R}(A) \subseteq \widehat{R}(B).$$

Dalla monotonia, dal momento che ogni $A_n \subseteq \bigcup_{n\in\mathbb{N}} A_n$, abbiamo che

$$\bigcup_{n\in\mathbb{N}} \widehat{R}(A_n) \subseteq \widehat{R}(\bigcup_{n\in\mathbb{N}} A_n)$$

L'inclusione inversa vale perchè le istanze di regole sono finitarie. Quindi in definitiva abbiamo

$$\bigcup_{n\in\mathbb{N}} \widehat{R}(A_n) = \widehat{R}(\bigcup_{n\in\mathbb{N}} A_n)$$

ovvero $\widehat{R}$ è *continuo*. Come introduciamo questa proprietà nel nostra trattazione degli ordini parziali? Ci serviremo della nozione di *funzione continua* tra cpo.

Definizione 12. *Una funzione $f : D \to E$ tra due cpo D e E è* monotona *sse*

$$\forall d, d' \in D.\ d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d').$$

f è detta poi continua *se è monotona e se per ogni catena $d_0 \sqsubseteq_D d_1 \sqsubseteq_D \cdots \sqsubseteq_D d_n \sqsubseteq \cdots$, abbiamo*

$$\bigsqcup_{n\in\mathbb{N}} f(d_n) = f(\bigsqcup_{n\in\mathbb{N}} d_n).$$

Possiamo ora introdurre il concetto di minimo punto fisso.

Definizione 13. *Sia $f : D \to D$ una funzione continua su un cpo D. Un* punto fisso *di f è un elemento d di D tale che $f(d) = d$. Un pre-punto fisso di f è un elemento d di D tale che $f(d) \sqsubseteq d$.*

La seguente proposizione definisce un metodo esplicito di costruzione del punto punto fisso di una funzione continua su un cpo.

Proposition 10. *Sia $f : D \to D$ una funzione continua su un cpo D con bottom. Sia*

$$fix(f) = \bigsqcup_{n\in\mathbb{N}} f^n(\bot).$$

Allora $fix(f)$ è un punto fisso di f e il più piccolo punto pre-fisso di f, ovvero:
- *$f(fix(f)) = fix(f)$*
- *se $f(d) \sqsubseteq d$ allora $fix(f) \sqsubseteq d$.*

Di conseguenza $fix(f)$ è il minimo punto fisso di f.

6.6 Esercizi

Esercizio 11. *Si determini, servendosi della semantica denotazionale, un programma equivalente a*

$$w \equiv \texttt{if}\ b\ \texttt{then}\ c; d\ \texttt{else}\ l; d$$

E un programma equivalente a

$$\texttt{while}\ b\ \texttt{do}\ c; \texttt{if}\ b\ \texttt{then}\ c\ \texttt{else}\ d$$

Esercizio 12. *Consideriamo una una variazione del linguaggio IMP, i cui comandi sono descritti dalla seguente grammatica*

$$c ::= X := t \mid c; d \mid \texttt{if}\ b\ \texttt{then}\ c\ \texttt{else}\ d \mid \texttt{repeat}\ c\ \texttt{until}\ b$$

In cui la semantica intuitiva di `repeat` c `until` b *sarà: "ripeti il comando c finchè è vera b". Adattando quanto fatto per il costrutto* `while` *si definisca :*
- *una semantica operazionale per i comandi*
- *una semantica denotazionale per i comandi*

Esercizio 13. *Definire una semantica denotazionale per il comando* `for` *definito nell'esercizio 1.*

7 La Corrispondenza tra i Tre Mondi

Abbiamo definito tre tipi di semantica per i programmi di **IMP**. Ci chiediamo ora, quale relazione esista tra queste semantiche. Vorremmo che fossero tra loro *equivalenti*.

Se consideriamo ad esempio la semantica operazionale e denotazionale, ci aspettiamo che se per un comando c si ha che $\langle c, \sigma\rangle \longrightarrow \theta$ allora $(\sigma, \theta) \in \mathcal{C}[\![c]\!]$ e viceversa. Più in generale vale il seguente risultato.

Proposition 11.

- *Per ogni* $t \in \mathbb{AEXP}$
$$\mathcal{A}[\![t]\!] = \{(\sigma, n) \mid \langle t, \sigma\rangle \longrightarrow n\}$$
- *Per ogni* $b \in \mathbb{BEXP}$
$$\mathcal{B}[\![b]\!] = \{(\sigma, w) \mid \langle b, \sigma\rangle \longrightarrow w\}$$
- *Per ogni* $c \in \mathbb{COM}$
$$\mathcal{C}[\![c]\!] = \{(\sigma, \theta) \mid \langle c, \sigma\rangle \longrightarrow \theta\}$$

Per quanto riguarda la semantica assiomatica ci interessa, invece, il seguente risultato di *soundness* che ci garantisce essenzialmente che tutto ciò scriviamo nella semantica assiomatica è corretto.

Proposition 12. *Sia* $\{A\}c\{B\}$ *un'asserzione di correttezza parziale. Allora*

$$\textit{Se } \vdash \{A\}c\{B\} \textit{ allora } \vDash \{A\}c\{B\}$$

Esercizio 14. *Si provino i primi due punti della Proposizione 11*