Semantica Assiomatica

La semantica assiomatica fu definita in unione con lo sviluppo di un metodo che migliorasse la correttezza dei programmi. Tali prove di correttezza, quando possono essere costruite, mostrano che un programma performa la computazione descritta dalle sue specifiche. In una prova, ogni dichiarazione di un programma viene sia preceduta che seguita da una espressione logica che specifica i vincoli sulle variabili del programma. Queste, piuttosto che l' intero stato di una macchina astratta (come con la semantica operazionale), sono utilizzate per specificare il significato della dichiarazione. La notazione utilizzata per descrivere i vincoli, senza dubbio il linguaggio della semantica assiomatica, è il calcolo dei predicati. Sebbene semplici espressioni booleane siano spesso adeguate per esprimere vincoli, a volte non lo sono.

Asserzioni

La semantica assiomatica è basata sulla logica matematica. Le espressioni logiche sono chiamate predicati o asserzioni. Un asserzione che precede immediatamente la dichiarazione di un programma, descrive i vincoli sulle variabili del programma in quel determinato punto del programma stesso. Un asserzione che segue immediatamente una dichiarazione descrive i nuovi vincoli su quelle variabili (e possibilmente altre) dopo l' USO delle dichiarazioni. Queste asserzioni sono chiamate rispettivamente precondizioni e postcondizioni delle dichiarazioni. Sviluppare una descrizione assiomatica o una prova di un dato programma richiede che ogni dichiarazione nel programma abbia sia una precondizione che una postcondizione.
Nelle sezioni successive, esamineremo le asserzioni dal punto di vista che le precondizioni per le dichiarazioni siano computate da postcondizioni date sebbene sia possibile considerare queste nel senso opposto. Assumeremo che tutte le variabili siano di tipo intero. Come semplice esempio, si consideri la seguente dichiarazione di assegnamento e postcondizione:

sum = 2 * x + 1 { sum > 1 }

Le asserzioni di precondizione e postcondizione sono presentate in BRACES per distinguerli dalle dichiarazioni del programma. Una possibile precondizione per questa dichiarazione è { x > 10 }.

Precondizioni Deboli

La precondizione più debole è la precondizione meno restrittiva che garantirà la validità delle postcondizioni associate. Per esempio, nelle precedenti dichiarazioni e postcondizioni, { x > 10}, { x > 50 }, e { x > 1000 } sono tutte valide precondizioni. La più debole di tutte le precondizioni in questo caso è { x > 0 }.
Se la precondizione più debole può essere computata dalle postcondizioni date per ogni dichiarazione di un linguaggio, allora le prove di correttezza possono essere costruite per programmi in quel linguaggio. La prova è iniziata utilizzando i risultati desiderati dell' esecuzione del programma come la postcondizione dell' ultima dichiarazione del programma. Questa postcondizione, insieme con l' ultima dichiarazione, è usata per computare la precondizione più debole per l' ultima dichiarazione. Questa precondizione è quindi utilizzata come postcondizione per la penultima dichiarazione. Questo processo continua fichè non si raggiunge l' inizio del programma. A quel punto le precondizioni della prima dichiarazione SPECIFICANO le condizioni sotto cui il programma computerà i risultati desiderati.

Per alcune dichiarazioni di programma, la computazione di una precondizione debole da dichiarazioni e una postcondizione è semplice e può essere specificata da un assioma. Nella maggior parte dei casi, tuttavia, la condizione debole può essere computata solo da una regola di inferenza. Un' assioma è una dichiarazione logica che si assume essere vera; una regola di inferenza è un metodo per inferire la veridicità di una asserzione sulla base dei valori di altre asserzioni. Per usare la semantica assiomatica con un dato linguaggio di programmazione, che sia per prove di correttezza o per specifiche di semantica formali, deve essere disponibile o un assioma o una regola di inferenza per ogni tipo di dichiarazione nel linguaggio. Nelle seguenti sottosezioni, presenteremo un assioma per dichiarazioni di assegnamento e regole di inferenza per sequenze di dichiarazioni, dichiarazioni di selezione e LOGICAL PRETEST LOOPS. Si noti che assumiamo che ne l' aritmetica ne le espressioni booleane abbiamo doppi effetti (side effects).

Dichiarazioni di assegnamento

Poniamo che x = E sia una dichiarazione di assegnamento generica e Q la sua postcondizione. Allora la sua precondizione, P, è definita dall' assioma

$P = Q_{[x \text{ --> } E]}$

che significa che P è calcolato come Q con tutte le istanze di x sostituite da E. Per esempio, se noi abbiamo la seguente dichiarazione di assegnamento e postcondizione

a = b / 2 – 1 { a < 10 }

la precondizione più debole è calcolata sostituendo b / 2 – 1 nell' asserzione { a < 10 } come di seguito:

b / 2 – 1 < 10
b < 22

Di conseguenza la precondizione più debole per la dichiarazione e la postcondizione data è { b < 22 }. Si ricordi che l' assioma di assegnamento è garantito essere vero solo in assenza di doppi effetti (side effects). Una dichiarazione di assegnamento ha un doppio effetto se cambia altre variabili oltre quelle al suo lato sinistro.

La notazione usuale per specificare la semantica assiomatica di una forma di dichiarazione data è

{P} S {Q}

Dove P è la precondizione, Q è la postcondizione, e S è la forma di dichiarazione. Nel caso di una dichiarazione di assegnamento, la notazione è

$\{Q_{[x\text{-->}E]}\}$ x=E {Q}

Come altro esempio di calcolo di precondizione per una dichiarazione di assegnamento, si consideri quanto di seguito:

x = 2 * y - 3 { x > 25 }

La precondizione è calcolata come di seguito:

2 * y - 3 > 25
y > 14

Così { y > 14 } è la precondizione più debole per questa dichiarazione di assegnamento e postcondizione.
Si noti che la comparsa del lato sinistro della dichiarazione di assegnamento nel suo lato destro non influenza il processo di calcolo della precondizione più debole. Per esempio, per

x = x + y - 3 { x > 10 }

la precondizione più debole è

x + y - 3 > 10
y > 13 – x

All' inizio della nostra discussione sulla semantica assiomatica, abbiamo stabilito che la semantica assiomatica fu sviluppata per provare la correttezza dei programmi. Alla luce di ciò, è naturale a questo punto chiedersi come l' assioma per le dichiarazioni MENT possa essere usato per provare qualsiasi cosa. Ecco come: una data dichiarazione di assegnamento con sia una precondizione che una postcondizione può essere considerata un teorema. Se l' assioma di assegnamento, quando applicato alla postcondizione e alla dichiarazione di assegnamento, produce la data precondizione,il teorema è provato. Per esempio, si consideri la seguente dichiarazione logica:

{ x > 3 } x = x - 3 { x > 0 }

Usando l' assioma di assegnamento su x = x - 3 { x > 0 }
produce { x > 3 }, che è la data precondizione. Quindi abbiamo provato la dichiarazione logica precedente.
Di seguito si consideri la dichiarazione logica

{ x > 5 } x = x - 3 { x > 0 }

In questo caso, la precondizione data, { x > 5 }, non è la stessa come l' asserzione prodotta dall' assioma. Tuttavia, è ovvio che { x > 5 } implichi { x > 3 }. Per utilizzare questo in una dimostrazione, abbiamo bisogno di una regola di inferenza, chiamata *la regola della conseguenza.*
La forma generale di una regola di inferenza è

$$\frac{S1,\ S2,\ ...,\ Sn}{S}$$

che stabilisce che se S1, S2,…Sn sono veri, allora la veridicità di S può essere inferita.

(AG) Per esempio, per la sostituzione, possiamo scrivere la regola seguente:

$$\frac{P = Q_{[x-->E]}}{\{P\}x = E\{Q\}}$$

La formula della regola di conseguenza è

$$\frac{\{P\}S\{Q\},\ P' \text{ --> } P, Q \text{ --> } Q'}{\{P'\}S\{Q'\}}$$

Il simbolo --> significa “implica”, ed S può essere qualsiasi dichiarazione di programma. ?? rule può essere stabilita come segue: se la dichiarazione logica {P} S {Q} è vera, l’ asserzione P implica l’ asserzione P’, e l’ asserzione Q implica l’ asserzione Q’, allora può essere inferito che {P’} S {Q’}. In altre parole la regola della conseguenza dice che una postcondizione può essere sempre indebolita e una precondizione può essere sempre rafforzata. Ciò risulta piuttosto utile nel provare i programmi. Per esempio permette il compimento della prova dell’ ultimo esempio di dichiarazione logica qui sopra.
Se permettiamo a P di essere {x > 3}, a Q e Q’ di essere {x > 0}, P’ essere {x > 5}, abbiamo

$$\frac{\{x > 3\}x = x - 3\{x > 0\},\ (x > 5) \text{ --> } (x > 3),\ (x > 0) \text{ --> } (x > 0)}{\{x > 5\}x = x - 3\{x > 0\}}$$

Questo completa la dimostrazione.

Sequenze

La precondizione più debole per una sequenza di dichiarazioni non può essere descritta da un assioma, poiché la precondizione dipende dal particolare tipo di dichiarazione nella sequenza. In questo caso, la precondizione può essere descritta solo con una regola di inferenza. Poniamo S1 ed S2 essere dichiarazioni adiacenti del programma. Se S1 ed S2 hanno le seguenti pre e postcondizioni **MA È S1 O SI?!?**

{P1} S1 {P2} {P2} S2 {P3} **MA È P1 O PI!?!**

la regola di inferenza per questo tipo di sequenza di due dichiarazioni è

$$\frac{\{PI\}S1\{P2\},\ \{P2\}S2\{P3\}}{\{P1\}S1;\ S2\{P3\}}$$

Quindi, per l’ esempio precedente, {PI} S1; S2 {P3} descrive la semantica assiomatica della sequenza S1; S2. Se S1 ed S2 sono le dichiarazioni di assegnamento

x1 = E1

e

x2 = E2

quindi abbiamo

$\{ P3_{[x2 \text{-->} E2]} \}$ x2 = E2 { P3 }

$\{ (P3_{[x2 \text{-->} E2]})_{[x1 \text{-->} E1]} \}$ x1 = E1 $\{ P3_{[x2 \text{-->} E2]} \}$

Quindi, la precondizione più debole per la sequenza xl = El; x2 = E2 con postcondizione P3 è $\{ (P3_{[x2 \text{-->} E2]})_{[x1 \text{-->} E1]} \}$.
Per esempio, si consideri le seguente sequenza e postcondizione:

y = 3 * x + 1;
x = y + 3; { x < 10 }

La precondizione per l' ultima dichiarazione di assegnamento è

y < 7

Questo viene quindi usato come postcondizione per il primo. La precondizione per la prima dichiarazione di assegnamento può essere ora calcolata:

3 * x + 1 < 7
x < 2

Selezione

Consideriamo successivamente la regola di inferenza per la dichiarazione di selezione. Consideriamo solo selezioni che includono anche clausole. La regola di inferenza è

$$\frac{\{B \texttt{ and } P\}S1\{Q\},\ \{(\texttt{not}B) \texttt{ and } P\}S2\{Q\}}{\{P\}\texttt{if } B \texttt{ then } S1 \texttt{ else } S2\{Q\}}$$

Questa regola indica che le dichiarazioni di selezione devono essere provate per entrambi i loro casi. La prima dichiarazione logica sopra la linea è la clausola "then"; la seconda è la clausola "else".
Si consideri il seguente esempio di computazione utilizzando la regola di selezione di inferenza. La dichiarazione di selezione di esempio è

if (x > 0) y = y - 1
else y = y + 1

Si supponga che la postcondizione per questa dichiarazione di selezione sia {y > 0}. Possiamo utilizzare l' assioma per l' assegnamento sulla clausola "then"

y = y - 1 {y > 0}

Questo produce {y - 1 > 0} o {y > 1}. Ora applichiamo lo stesso assioma alla clausola "else"

y = y + 1 {y > 0}

Questo produce la precondizione {y + 1 > 0} o {y > -l}. Poiché {y > 1}-->{y > -1}, la regola di conseguenza ci permette di usare {y > 1} per la precondizione della dichiarazione di selezione.

Cicli While

Altro costrutto essenziale in un linguaggio di programmazione imperativo è il ciclo While. Computare le precondizioni deboli per un ciclo While è molto più complesso che per una normale sequenza in quanto non si può sempre conoscere a priori il numero di iterazioni che il ciclo

eseguirà. Nel caso si conoscano invece, il ciclo While può essere trattato come una normale sequenza.
Possiamo computare le precondizioni dei cicli usando il metodo dell'induzione. Primo passo di questo metodo è la ricerca di una *ipotesi induttiva* che, nel nostro caso, è la ricerca di una asserzione chiamata **Invariante di ciclo:** tale asserzione è cruciale per la ricerca delle precondizioni deboli.

La regola di inferenza per le precondizioni di un ciclo While è la seguente:

$$\frac{\{I \; and \; B\} S \{I\}}{\{I\} \; \texttt{while} \; B \; \texttt{do} \; S \; \texttt{end} \; \{I \; and \; (not \; B)\}}$$

Dove I è l'Invariante.
La descrizione assiomatica di un ciclo While è la seguente:

{P} while B do S end{Q}

L'Invariante **deve** soddisfare alcune richieste per essere utile:

1. Le precondizioni deboli devono garantire la verità dell'Invariante;
2. L'Invariante deve garantire la verità delle postcondizioni;

Durante l'esecuzione del ciclo, la verità dell'Invariante non dovrà essere influenzata dalla valutazione delle condizioni di verità del ciclo e dalle asserzioni del corpo del ciclo.

Altro problema è la terminazione del ciclo. Se Q è la postcondizione che si mantiene all'uscita del ciclo, P è la precondizione che garantisce Q all'uscita del ciclo e che il ciclo termini.

La descrizione assiomatica di un ciclo While richiede la veridicità delle seguenti asserzioni:

- **P → I**
- **{I and B} S {I}**
- **{I and (notB)} → Q**
- **Il ciclo termina**

Per determinare l'Invariante possiamo procedere induttivamente computando la relazione in un certo numero di casi sperando che emerga un modello comune applicabile al caso generale.
Trattiamo il processo di produzione di condizioni deboli come una funzione wp. In generale

wp (statement, postcondition) = precondition

In caso di assegnamento abbiamo

wp (x = E, Q) = $Q_{[x!E]}$

Per trovare I utilizziamo Q per computare P un certo numero di iterazioni del corpo del ciclo. Se il corpo contiene una singola asserzione di assegnamento, l'assioma per le asserzioni di assegnamento può essere usato per questi casi.

ESEMPIO 1 (DA VERIFICARE L'INSERIBILITÀ O MENO NEL RIASSUNTO, IO PROPENDO PER IL SÌ)

while y <> x **do** y = y + 1 **end** {y = x}

Iterazione 0, la precondizione debole è{y = x}

Iterazione 1, wp (y = y + 1, {y = x}) = {y + 1 = x} = {y = x – 1}

Iterazione 2, wp (y = y + 1, {y = x – 1}) = {y + 1 = x – 1} = {y = x – 2}

Iterazione 3, wp (y = y + 1, {y = x – 2}) = {y + 1 = x – 2} = {y = x – 3}

Si nota come {y < x} sia sufficiente per casi di 1 o più iterazioni. Combinando questa precondizione con quella dell'iterazione 0 otteniamo {y ≤ x}, che può essere usata per l'Invariante. Ora dobbiamo essere sicuri che il nostro Invariante soddisfi le 4 condizioni precedentemente enunciate.

In primis, poiché P = I, **P → I**.

In Secondo luogo deve essere vero che **{I and B} S {I}.**
Nel nostro esempio abbiamo che: {y ≤ x and y <> x} y = y + 1 {y ≤ x}. Applicando l'assioma di assegnamento a y = y + 1 {y ≤ x}, otteniamo {y + l ≤ x}, equivalente a {y < x}, implicato da {y ≤ x and y <> x).

Terzo, deve essere vero che **{I and (not B)} → Q.**
Abbiamo:

{(y ≤ x) and not (y <> x)} → {y = x}
{(y ≤ x) and (y = x)} → {y = x}
{y = x} → {y = x}

Quindi l'asserzione è ovviamente vera.

In ultimo dobbiamo verificare che il ciclo termini effettivamente o meno. La precondizione garantisce che inizialmente y < x. Il ciclo incrementa y fino a diventare uguale a x. Non ci riguarda di quanto y sia minore di x inizialmente poiché comunque diventerà uguale a x. A questo punto il ciclo terminerà. La nostra scelta soddisfa le 4 condizioni enunciate, **pertanto è adeguata**.

Il processo sopra usato però non sempre ci porta a trovare la precondizione debole.

ESEMPIO 2 (ANCHE QUESTO DA VERIFICARE, PROPENDO PER IL SÌ)

while s > l **do** s = s/2 **end** {s = 1}

Iterazione 0, la precondizione debole è {s = 1}

Iterazione 1, wp (s = s / 2, {s = 1}) = {s / 2 = l} = {s = 2}

Iterazione 2, wp (s = s / 2, {s = 2}) = {s / 2 = 2} = {s = 4}

Iterazione 3, wp (s = s / 2, {s = 4}) = {s / 2 = 4} = {s = 8}

Possiamo notare come l'Invariante sia s = una Potenza non negativa di 2.
Ancora una volta I rispetta le 4 richieste fatte in precedenza e può essere utilizzato come P. Tuttavia si vede come questa volta I non sia una condizione debole. Consideriamo il caso di {s > 1}, l'asserzione
{s > 1} **while** s > l **do** s = s/2 **end** {s = 1}

Può facilmente essere dimostrata e questa precondizione è significativamente più debole di quella computata precedentemente. Il ciclo e le precondizioni sono soddisfatte per qualunque valore positivo di s e non solo per le potenze di 2. Per la regola della conseguenza, usare una precondizione più forte di quella debole non invalida una prova.

È utile capire la natura degli Invarianti: un Invariante è una versione indebolita della postcondizione e funge da precondizione per il ciclo. Quindi deve essere debole abbastanza da essere vera a priori dell'esecuzione del ciclo ma, combinata con la condizione di uscita del loop, deve essere forte abbastanza per forzare la verità della postcondizione.
Provare la terminazione del ciclo è comunque molto difficile. Se questa è ben visibile la descrizione assiomatica del ciclo è chiamata **totale correttezza**, altrimenti è chiamata **parziale correttezza**.

ESEMPIO 3 (VERIFICARE ANCHE QUI SE INSERIRLO OPPURE NO)

Consideriamo il seguente pseudocodice di questo programma che calcola la funzione fattoriale:

{n ≥ 0}
count = n;
fact = 1;
while count <> 0 **do**
fact = fact * count;
count = count - 1;
end
{fact = n!}

Vogliamo verificarne la correttezza.
In questo caso la tecnica di ricerca dell'Invariante non funziona. Dobbiamo studiare il codice: si nota come la computazione del fattoriale di n parta dall'ultima moltiplicazione n*(n – 1). L'Invariante potrebbe essere

fact = (count + 1) * (count + 2) * ... * (n - 1) * n

ma dobbiamo anche considerare che count deve essere sempre non negative per cui I diventerà

I = (fact = (count + 1) * ... * n) AND (count ≥ 0)

Adesso procediamo a controllare che I rispetti le 4 condizioni definite.

I può essere usato al posto di P per cui P implica I
Per la seconda richiesta, **{ I and B} S { I},** vediamo che I and B è:

((fact = (count + 1) * ... * n) AND (count ≥ 0)) AND (count <> 0) che si può ridurre a:

(fact = (count + 1) * ... * n) AND (count > 0). Nel nostro caso dobbiamo computare la precondizione del corpo del ciclo, usando I per la postcondizione. Avremo

{P} count = count - 1 {I}

P diventa

{(fact = count * (count + 1) * ... * n) AND (count ≥ 1)}

Usando questa come postcondizione del primo assegnamento nel corpo del ciclo otteniamo

{P} fact = fact * count {(fact = count * (count + 1) * ... * n) AND (count ≥ 1)}

con P = {(fact = (count + 1) * ... * n) AND (count ≥ 1)}

I implica B, quindi la regola è dimostrata essere vera.
L'ultimo test da effettuare su I è **I and (not B) → Q.** Per il nostro esempio si ha

((fact = (count +1) * ... * n) AND (count ≥ 0)) AND (count = 0)) → fact = n!

Cio è vero per count = 0 (si nota come la prima parte sia la definizione di fattoriale). La nostra scelta quindi incontra le condizioni che un Invariante di ciclo deve rispettare.
Adesso possiamo usare P come postcondizione nel secondo assegnamento del programma

{P} fact = 1 {(fact = (count + 1) * ... * n) AND (count ≥ 0) }

con P pari a

(1 = (count + 1) * ... * n) AND (count ≥ 0))

Usando questa come postcondizione del primo assegnamento nel codice

{P} count = n {(1 = (count + 1) * ... * n) AND (count ≥ 0)) }

che produce P

{(n+l)*...*n = 1 AND (n ≥ 0)

L'operatore a sinistra dell' AND è vera (perchè 1 = l), mentre a destra abbiamo la precondizione dell'intero segmento di codice. Quindi il programma è corretto.

Valutazione

Per definire la semantica di un linguaggio di programmazione usando il metodo assiomatico dobbiamo definire una regola di inferenza per ogni asserzione del linguaggio. Per alcune asserzioni è difficile definire un regola di inferenza. La soluzione più semplice è quella di definire un linguaggio avendo in mente la semantica assiomatica per inserire solo quelle asserzioni definibili tramite le regole di inferenza descritte.
La semantica assiomatica è un potente strumento per definire la correttezza di un programma; inoltre fornisce un ambiente dove poter ragionare sul programma, sia in costruzione, sia terminato. Tuttavia la sua utilità nella descrizione del significato del linguaggio di programmazione ad altri programmatori e ai realizzatori di compilatori e molto limitata.