Appunti di

Interpretazione Astratta

dal corso di Analisi di Programmi

Michael Lodi

"Fedelmente" tratti dalle lezioni e dagli appunti [6]

del Professor Cosimo Laneve

Versione 0.4 - 24 Maggio 2011

Sommario. Questi sono gli appunti di una parte del corso “Analisi di Programmi”, tenuto dal Professor Cosimo Laneve durante l’anno accademico 2010/2011. Non hanno la pretesa di essere completi ed è possibile (probabile) che contengano qualche errore. Non sono in alcun modo sostitutivi del materiale originale o degli appunti delle lezioni, ma sono da intendere come supporto allo studio.

Questioni burocratiche (quindi|comunque) importanti

0.1. Licenza

0.2. L'autore

Michael Lodi è attualmente studente al primo anno di Laurea Magistrale in Informatica (curr. A - Scienze Informatiche), Università di Bologna. È un appassionato di Informatica Teorica e di Linguaggi di Programmazione.

Quello che vi chiede è: di essere *citato come autore* di queste note ed eventuali derivati (nel rispetto cioè della **licenza**); di comunicargli qualsiasi *errore, imprecisione, commento, critica, insulto* a ***lodi@cs.unibo.it***

0.3. Notazione

In generale, gli operatori indicati con i consueti simboli $(+, *, /, =)$ sono operatori sintattici, mentre i corrispondenti operatori semantici saranno indicati con la seguente notazione $(\dot{+}, \dot{=}, \dot{\leq}, \dot{cond}...)$ mentre con la seguente notazione saranno indicati gli operatori su interi visti nella prima parte del corso: $(\tilde{+}, \tilde{-}, ...)$. Notare la differente notazione scelta rispetto a [1].

Le funzioni semantiche verranno indicate nel modo seguente: $\mathcal{S}[\![]\!]$, $\mathcal{A}[\![]\!]$, $\mathcal{B}[\![]\!]$...

Le ipotesi induttive nelle dimostrazioni saranno indicate con IH (Inductive Hypothesys).

In generale, si userà la lettera d per indicare gli elementi appartenenti al dominio astratto A. Per indicare che un dominio, una funzione, un elemento riguarda la semantica astratta, useremo la notazione ^{A} o talvolta un pedice (es. $_A$).

La non definizione (parzialità) di una funzione è indicata in due modi:

- omettendo di definire la funzione su un insieme di valori: in tal caso la funzione sarà definita solo sull'insieme di valori su cui... è stata definita! e indefinita altrimenti;
- utilizzando il "metavalore" undef.

L'ordine della composizione funzionale, per scelta arbitraria, indica l'ordine inverso di applicazione: $(g \circ f)(x) \stackrel{def}{=} g(f(x))$.

0.4. Note di rilascio

- Versione 0.1 - 1 Aprile 2010. Prima versione, non riletta dall'autore e con alcune parti mancanti. Rilasciata per avere un maggior numero di "revisori".
- Versione 0.2 - 19 Aprile 2010. Aggiunto il capitolo 4 sul Java Bytecode Verifier senza modificare i capitoli precedenti.
- Versone 0.3 - 16 Maggio 2010. Apportate le seguenti correzioni
  - P. 10 - Corretta l'osservazione sull'espressione aritmetica che contiene solo + e *
  - P. 12 - Aggiunta osservazione
  - P. 17 - Spiegazione della presenza della costante 5
  - P. 22-23 - Corrette definizioni della funzione semantica sui booleani, astratta e concreta
  - Fine p.24 / prop. 3.19 pag 25 - Importante correzione su quali funzioni siano l'identità.
  - P. 31, teorema 3.29 - Correzioni riguardanti la funzione di concretizzazione parziale
  - Corretto il Fatto 4.18 p. 44
  - Aggiunte esplicative di concetti poco chiari
  - Numerose correzioni di refusi e di lessico, non rilevanti ai fini della comprensione
- Versione 0.4 - 24 Maggio 2010.
  - Corretti dominio e codominio in definizione 2.2, che erano invertiti nelle funzioni di astrazione e concretizzazione
  - Chiarita la dimostrazione del teorema 2.4
  - Chiariti i dubbi legati alle proposizioni 3.1, 3.20 e a quella di pagina 30
  - Aggiunta "freccia parziale" nel teorema 3.23 (minimo impatto grafico, grande differenza semantica) e chiarita la dimostrazione del caso "assegnamento".
  - Chiarita la dimostrazione del teorema 3.29
  - Chiariti con il Professore molti dubbi sparsi (che erano indicati con ?)

0.5. Ringraziamenti

Si ringraziano Pietro Ansaloni, Domiziana Suprani e Matteo Romanelli, per i numerosi errori segnalatimi e Marco Poletti, perché esiste, perché i suoi appunti hanno ispirato numerose correzioni e ampliamenti e le sue (giustamente) pignole osservazioni stanno migliorando ancora questo lavoro.

CAPITOLO 1

Iniziamo con la pratica

1.1. L'idea di interpretazione astratta

L'*interpretazione astratta* è una tecnica utilizzata da 30 anni (è stata formalizzata nel 1977 da Patrick e Radhia Cousot) per trattare in modo sistematico astrazioni e approssimazioni. È nata per descrivere analisi statiche di programmi imperativi e provarne la correttezza (ed è stata poi estesa con successo a varie classi di linguaggi di programmazione). È una tecnica generale per correlare semantiche a diversi livelli di astrazione.

Si parte da una ***semantica concreta***, ovvero una *funzione* che assegna significati precisi ai comandi di un programma in un dominio di computazione fissato. Poi si individua un *dominio astratto*, che modella alcune *proprietà* delle computazioni concrete, tralasciando la rimanente informazione.
Si definisce infine una ***semantica astratta***, che permette di "eseguire astrattamente" il programma sul dominio astratto per calcolare la proprietà che il dominio astratto modella. Tipicamente, il calcolo della semantica astratta è un calcolo di punto fisso.

Bisogna inoltre dimostrare che la semantica astratta è una approssimazione corretta della semantica concreta.

1.2. Un primo semplice esempio

Iniziamo con un semplice esempio. Consideriamo un semplicissimo linguaggio delle espressioni `AExp` che definisce prodotti di interi

$$a ::= \mathsf{n} \,|\, a * a$$

e la cui semantica può essere espressa come una funzione

$$\mathcal{A} : \texttt{AExp} \to \wp(\mathbb{Z})$$

così definita

$$\begin{aligned} \mathcal{A}[\![n]\!] &= \{n\} \\ \mathcal{A}[\![a_1 * a_2]\!] &= \mathcal{A}[\![a_1]\!] \dot{*} \mathcal{A}[\![a_2]\!] \end{aligned}$$

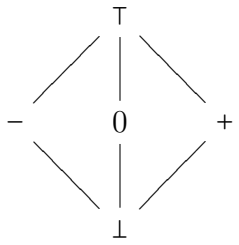
in cui $\dot{*}$ è il prodotto "semantico" concreto, analogo a quello visto nella prima parte del corso ($\tilde{*}$), ridefinito però su insiemi di interi:

$$A \dot{*} B = \{a \tilde{*} b \,|\, a \in A \ e \ b \in B\}$$

Vogliamo però ora considerare un'astrazione della semantica concreta (*semantica astratta*) che calcola solo il segno delle espressioni.

Nella prima parte del corso il nostro dominio era $\mathbb{Z}$ con ordinamento piatto, mentre ora è $\wp(\mathbb{Z})$. In particolare, come ordinamento utilizzeremo l'inclusione insiemistica; avremo quindi il reticolo completo $(\wp(\mathbb{Z}), \subseteq)$.

Ora consideriamo un dominio astratto con il suo ordinamento: $(A, \leq)$. È un reticolo completo definito nel modo seguente:



Consideriamo quindi una funzione di semantica astratta

$$\mathcal{A}^A : \texttt{AExp} \to A$$

definita come

$$\begin{aligned} \mathcal{A}^A[\![n]\!] &= \begin{cases} - & se\, n < 0 \\ 0 & se\, n = 0 \\ + & se\, n > 0 \end{cases} \\ \mathcal{A}^A[\![a_1 * a_2]\!] &= \mathcal{A}^A[\![a_1]\!] *^A \mathcal{A}^A[\![a_2]\!] \end{aligned}$$

in cui l'operatore astratto $*^A$ è definito per casi sugli elementi di A:

$*^{\mathbf{A}}$	$\bot$	**-**	**0**	$+$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
-	$\bot$	$+$	0	-	$\top$
0	$\bot$	0	0	0	0
$+$	$\bot$	-	0	$+$	$\top$
$\top$	$\bot$	$\top$	0	$\top$	$\top$

Vogliamo ora dimostrare che questa astrazione è corretta, ovvero che **prevede correttamente** il segno delle espressioni. Avremo quindi che, per ogni $a \in \texttt{AExp}$:

- se $\mathcal{A}[\![a]\!] = \{n\}$ e $n < 0 \Rightarrow \mathcal{A}^A[\![a]\!] = -$
- se $\mathcal{A}[\![a]\!] = \{n\}$ e $n = 0 \Rightarrow \mathcal{A}^A[\![a]\!] = 0$
- se $\mathcal{A}[\![a]\!] = \{n\}$ e $n > 0 \Rightarrow \mathcal{A}^A[\![a]\!] = +$

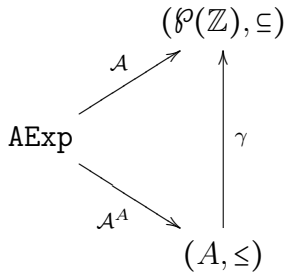
In alternativa, possiamo introdurre una funzione che associa ad ogni valore astratto l'insieme dei valori concreti che esso rappresenta. Avremo dunque

$$\gamma : A \to \wp(\mathbb{Z})$$

definita nel modo seguente

$$\begin{aligned}
\gamma(\bot) &= \varnothing \\
\gamma(-) &= \mathbb{Z}^- \\
\gamma(0) &= \{0\} \\
\gamma(+) &= \mathbb{Z}^+ \\
\gamma(\top) &= \mathbb{Z}
\end{aligned}$$

Possiamo riassumere con questo diagramma la relazione di concretizzazione tra i domini concreto e astratto:



Alla luce di quanto visto, la correttezza può essere riscritta più formalmente nel modo seguente.

Teorema 1.1 (Correttezza)**.** *Per ogni* $a \in$ `AExp` *vale che*

$$\mathcal{A}[\![a]\!] \subseteq \gamma(\mathcal{A}^A[\![a]\!])$$

Dimostrazione. Per induzione sulla struttura sintattica di a.

- Caso base: $a = n$. Avremo dunque che $\mathcal{A}[\![n]\!] = \{n\}$.
  - Se $n > 0$ allora $\gamma(\mathcal{A}^A[\![n]\!]) = \mathbb{Z}^+$ e dunque la relazione di contenimento è rispettata.
  - Se $n = 0$ allora $\gamma(\mathcal{A}^A[\![n]\!]) = \{0\}$ e dunque la relazione di contenimento è un'uguaglianza.
  - Se $n < 0$ allora $\gamma(\mathcal{A}^A[\![n]\!]) = \mathbb{Z}^-$ e dunque la relazione di contenimento è rispettata.
- Caso induttivo: $a = a' * a''$. Dunque sarà

$$\begin{aligned}
\mathcal{A}[\![a' * a'']\!] &= \mathcal{A}[\![a']\!] \dot{*} \mathcal{A}[\![a'']\!] \\
\text{per monotonia di } \dot{*} \text{e per IH} &\subseteq \gamma(\mathcal{A}^A[\![a']\!]) \dot{*} \gamma(\mathcal{A}^A[\![a'']\!]) \\
\text{per } \textbf{correttezza locale} \text{ di } *^A &\subseteq \gamma(\mathcal{A}^A[\![a']\!] *^A \mathcal{A}^A[\![a'']\!]) \\
&= \gamma(\mathcal{A}^A[\![a' * a'']\!])
\end{aligned}$$

□

Osservazione. Se consideriamo $(\wp(\mathbb{Z}), \subseteq)$ è facile dimostrare che $\dot{*}$ è monotono. Dobbiamo invece dimostrare la correttezza locale di $*^A$

Definizione 1.2. Se $\dot{op}$ è un'operazione n-aria e op^A è la sua astratta, la correttezza locale di op^A equivale a dimostrare che

$$\dot{op}(\gamma(d_1), ..., \gamma(d_n)) \subseteq \gamma(op^A(d_1, ..., d_n))$$

In questo specifico caso dovremo quindi dimostrare che, per ogni $d, d' \in A$ vale che $\gamma(d)\dot{*}\gamma(d') \subseteq \gamma(d *^A d')$.

Dimostrazione. Per casi (visto che A è finito) sugli elementi.

- Se $a = -$ e $b = -$, allora $\gamma(a *^A b) = \gamma(+) = \mathbb{Z}^+$ e d'altra parte $\gamma(a)\dot{*}\gamma(b) = \mathbb{Z}^-\dot{*}\mathbb{Z}^- = \mathbb{Z}^+$
- Se $a = -$ e $b = +$, allora $\gamma(a *^A b) = \gamma(-) = \mathbb{Z}^-$ e d'altra parte $\gamma(a)\dot{*}\gamma(b) = \mathbb{Z}^-\dot{*}\mathbb{Z}^+ = \mathbb{Z}^-$
- e così via... □

1.3. Estensione: le operazioni aritmetiche.

Estendiamo ora il linguaggio `AExp` con gli altri operatori aritmetici, di modo da ricondurci a quello visto nella prima parte del corso [1]. Avremo quindi:

$$a ::= \mathsf{n} \,|\, a * a \,|\, a + a \,|\, a - a \,|\, a/a$$

e la semantica delle espressioni aritmetiche a può essere espressa come una funzione

$$\mathcal{A} : \texttt{AExp} \to \wp(\mathbb{Z})$$

così definita

$$\begin{aligned} \mathcal{A}[\![n]\!] &= \{n\} \\ \mathcal{A}[\![a_1\, op\, a_2]\!] &= \mathcal{A}[\![a_1]\!]\, \dot{op}\, \mathcal{A}[\![a_2]\!] \end{aligned}$$

Dobbiamo ora ridefinire $\dot{op}$ su insiemi di interi. Siano $A, B \in \wp(\mathbb{Z})$:

$$A\, \dot{op}\, B \stackrel{def}{=} \{a\, \tilde{op}\, b \,|\, a \in A \text{ e } b \in B\}$$

Esempio. Possiamo modellare in modo pulito la non terminazione (utilizzando funzioni totali, senza fare ricorso alla parzialità). Infatti:

$\varnothing\, \dot{op}\, Y = \varnothing$

$X/\{0\} = \varnothing$

...

Proposizione 1.3. *Per ogni $a \in \texttt{AExp}$, $\mathcal{A}[\![a]\!]$ è un singoletto o l'insieme vuoto.*

DIMOSTRAZIONE. Per induzione strutturale su a. È lasciata allo studente per esercizio. □

La *semantica astratta*, allo stesso modo, è definita come in precedenza, estendendola agli altri operatori. La funzione $\mathcal{A}^A$ viene così ridefinita come

$$\begin{aligned}\mathcal{A}^A[\![n]\!] &= \begin{cases} - & se\, n < 0 \\ 0 & se\, n = 0 \\ + & se\, n > 0 \end{cases} \\ \mathcal{A}^A[\![a_1\, op\, a_2]\!] &= \mathcal{A}^A[\![a_1]\!]\, op^A\, \mathcal{A}^A[\![a_2]\!]\end{aligned}$$

Dobbiamo ora definire gli operatori astratti, come abbiamo fatto per $*^A$. A differenza di somma e moltiplicazione, differenza e divisione non sono commutative. Nelle tabelle avremo allora sulla prima colonna il primo argomento, mentre sulla prima riga il secondo argomento.

$+^{\mathbf{A}}$	$\bot$	**-**	**0**	$+$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
-	$\bot$	-	-	$\top$	$\top$
0	$\bot$	-	0	$+$	$\top$
$+$	$\bot$	$\top$	$+$	$+$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$

$-^{\mathbf{A}}$	$\bot$	**-**	**0**	$+$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
-	$\bot$	$\top$	-	-	$\top$
0	$\bot$	$+$	0	-	$\top$
$+$	$\bot$	$+$	$+$	$\top$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$

$/^{\mathbf{A}}$	$\bot$	**-**	**0**	$+$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
-	$\bot$	$\top$	$\bot$	$\top$	$\top$
0	$\bot$	0	$\bot$	0	0
$+$	$\bot$	$\top$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$

Osservazione. Osserviamo alcuni fatti.

Dobbiamo stare attenti a preservare la correttezza. Per esempio dovremo perdere di precisione sulla divisione tra positivi e negativi (e tutte le combinazioni), perché potrebbe restituirci 0. Ad esempio $\{1\}\dot{/}\{2\} = \{0\}$ che non sarebbe corretto se $+/^A+ = +$.

Sia a contenente solo + e $*$. Allora

$$\mathcal{A}[\![a]\!] = \{n\} \text{ AND } \begin{cases} n \in \mathbb{Z}^+ \Rightarrow \mathcal{A}^A[\![a]\!] = + \\ n = 0 \Rightarrow \mathcal{A}^A[\![a]\!] = 0 \\ n \in \mathbb{Z}^- \Rightarrow \mathcal{A}^A[\![a]\!] = - \end{cases}$$

Questo non vale con gli altri operatori. [E non vale neanche con la somma, se ci sono costanti negative].

Esempio. In alcuni casi è possibile una perdita di informazione, in altri no.

$$\mathcal{A}[\![(1+2)-3]\!] = \{0\} \quad \text{mentre} \quad \mathcal{A}^A[\![(1+2)-3]\!] = (+ +^A +) +^A - = + +^A - = \top$$
$$\mathcal{A}[\![(5*4)+6]\!] = \{26\} \quad \text{e infatti} \quad \mathcal{A}^A[\![(5*4)+6]\!] = (+ *^A +) +^A + = + +^A + = +$$

Osservazione. La correttezza della semantica, la correttezza locale degli operatori e la loro monotonia si dimostrano generalizzando le dimostrazioni già viste in precedenza.

CAPITOLO 2

Formalizzazione

In questo capitolo daremo definizioni più formali di *interpretazione astratta*, servendoci di alcuni strumenti avanzati di algebra: le *connessioni di Galois* e le *inserzioni di Galois*.

2.1. Il dominio astratto e le funzioni γ e α

Il dominio astratto che abbiamo considerato $(A, \leq)$ è un *insieme parzialmente ordinato* in cui l'ordinamento parziale rappresenta la nozione di approssimazione/precisione ("più piccolo = più preciso").

$$\begin{array}{c}(\wp(\mathbb{Z}), \subseteq) \\ \alpha \downarrow \quad \uparrow \gamma \\ (A, \leq)\end{array}$$

In particolare notiamo che

- $(A, \leq)$ è un reticolo completo
- la funzione di concretizzazione γ è monotòna, ovvero vale

$$d \leq d' \Rightarrow \gamma(d) \leq_c \gamma(d')$$

ovvero tutti gli oggetti più grandi di d saranno mappati in elementi più grandi.

Possiamo definire una sorta di funzione inversa, e la chiameremo *funzione di astrazione* α che mappa un insieme X di valori concreti nel *più preciso valore astratto* che rappresenta X. Vogliamo cioè considerare la "migliore approssimazione di X", ovvero l'elemento più piccolo (= più preciso). Come fare? Considerando il *massimo dei minoranti.*

Costruiamo dunque α in funzione di γ. La funzione dovrà avere tipo

$$\alpha : \wp(\mathbb{Z}) \to A$$

e possiamo definirla, in modo automatico, come

$$\alpha(X) = \bigsqcap \{d \mid X \subseteq \gamma(d)\}$$

Intuitivamente, considero *le approssimazioni di X secondo γ e prendo la migliore (glb).*

Esempio. Vediamo di capire il motivo di questa definizione.

$\alpha(\{1,2\}) = \bigsqcap\{+,\top\} = +$ e noi vogliamo proprio $+$, cioè il *glb*.

$\alpha(2\mathbb{Z}) = \bigsqcap\{\top\} = \top$

$\alpha(\{0\} \cup \mathbb{Z}^+) = \top$

Possiamo però anche definire α direttamente, nel modo seguente:

$$\alpha(X) = \begin{cases} \bot & se\, X = \varnothing \\ - & se\, X \subseteq \mathbb{Z}^- \\ 0 & se\, X = \{0\} \\ + & se\, X \subseteq \mathbb{Z}^+ \\ \top & altrimenti \end{cases}$$

Ovviamente, è possibile fare anche il contrario, cioè definire γ in termini di α.

$$\begin{array}{c} (\wp(\mathbb{Z}), \subseteq) \\ \alpha \big(\quad \big| \, \gamma \\ (A, \leq) \end{array}$$

Dunque la definizione di **correttezza** diventerebbe

$$\alpha(\mathcal{A}[\![a]\!]) \leq \mathcal{A}^A[\![a]\!]$$

e possiamo definire γ in modo "automatico" come

$$\gamma(d) = \bigsqcup\{X \mid \alpha(X) \leq d\}$$

Osservazione. Da questo segue che, dato $d \in A$, se $X \subseteq \gamma(d)$ allora $\alpha(X) \leq d$.

Esempio. $\gamma(+) = \bigsqcup\{\varnothing, \{1\}, \{2\}, \{1,2\}, ..., \mathbb{Z}^+\} = \mathbb{Z}^+$

Vediamo con un semplice diagramma le relazioni tra γ e α:

$$\begin{array}{ccc} \gamma(d) & \longleftarrow & d \\ \cup\mathrm{I} & & \vee\mathrm{I} \\ X & \longrightarrow & \alpha(X) \end{array}$$

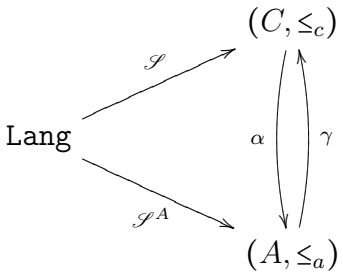
2.2. Interpretazione astratta: definizione formale

Definizione 2.1. In generale, possiamo definire un'interpretazione astratta come formata da

- Un dominio concreto $(C, \leq_c)$, che è un reticolo completo
- Un dominio astratto $(A, \leq_a)$, che è un reticolo completo
- Una funzione di concretizzazione γ monotòna
- Una funzione di astrazione α monotòna

tali che

- $((C, \leq_c), (A, \leq_a), \gamma, \alpha)$ formano un'*inserzione di Galois*
- le operazioni astratte astraggono correttamente su A una qualche semantica concreta (che indicheremo in generale con $\mathscr{S}$) su C, ovvero vale che $\forall S \in \texttt{Lang} : \mathscr{S}[\![S]\!] \leq_c \gamma(\mathscr{S}^A[\![S]\!])$



2.3. Connessioni e Inserzioni di Galois

Questi concetti sono dovuti al matematico Évariste Galois, dalla vita breve ma avventurosa [10].

Definizione 2.2. Si definisce **connessione di Galois (GC)** una quadrupla (C, A, γ, α) in cui

- $(C, \leq_c)$ e $(A, \leq_a)$ sono reticoli completi
- $\gamma : A \to C$, detta *funzione di concretizzazione* e $\alpha : C \to A$, detta *funzione di astrazione* sono **monotòne**
- $\forall c \in C,\ c \leq_c \gamma(\alpha(c))$
- $\forall d \in A,\ \alpha(\gamma(d)) \leq_a d$

Possiamo comprendere meglio questi ultimi due vincoli riguardando le relazioni tra α e γ:

$$\begin{array}{ccc} \gamma(d) & \longleftarrow & d \\ \vee^{c} & & \vee^{a} \\ c & \longrightarrow & \alpha(c) \end{array}$$

In particolare i vincoli nascono considerando separatamente due metà del diagramma:

- se $c = \gamma(d)$ allora avremo immediatamente che $\alpha(\gamma(d)) \leq_a d$

- se $d = \alpha(c)$ allora sarà $c \leq_c \gamma(\alpha(c))$

Definizione 2.3. Si definisce **inserzione di Galois (GI)** una connessione di Galois in cui vale una delle tre condizioni equivalenti

- a. α è suriettiva (l'immagine coincide col codominio, ovvero $\forall y \in Cod \exists x \in Dom \,|\, y = f(x)$)
- b. γ è iniettiva (elementi distinti del dominio hanno immagini distinte, ovvero $\forall x_1, x_2 \in Dom\ f(x_1) = f(x_2) \Rightarrow x_1 = x_2$)
- c. $\forall d \in A\ \alpha(\gamma(d)) = d$

Teorema 2.4. *a.* $\equiv$ *b.* $\equiv$ *c.*

Dimostrazione. Dimostreremo che

- $(c \Rightarrow b)$. Dobbiamo provare che γ è iniettiva, ovvero $\gamma(d) = \gamma(d') \Rightarrow d = d'$. Ipotizziamo dunque che $\gamma(d) = \gamma(d')$ e applichiamo α. Avremo $\alpha(\gamma(d)) = \alpha(\gamma(d'))$. Per ipotesi (c) sappiamo che $\forall d \in A\ d = \alpha(\gamma(d))$ e dunque segue immediatamente che $d = d'$.
- $(b \Rightarrow a)$.
  - Dimostriamo che $(b \Rightarrow c)$, ovvero che, supposta γ iniettiva, allora $\alpha(\gamma(d)) = d$. Per def. di GC, $\alpha(\gamma(d)) \leq_a d$ e per monotonia $\gamma(\alpha(\gamma(d))) \leq_c \gamma(d)$. Sempre per def. di GC $c = \gamma(d) \leq_c \gamma(\alpha(c))$. Ma allora $\gamma(\alpha(\gamma(d))) = \gamma(d)$ e, per (b), $\alpha(\gamma(d)) = d$.
  - Proviamo poi che $(c \Rightarrow a)$, ovvero sapendo che $\alpha(\gamma(d)) = d$ dobbiamo provare che $\forall d \in A\ \exists c \in C \,|\, d = \alpha(c)$. Per ipotesi, $\alpha(\gamma(d)) = d$ e cioè il c che cerchiamo sarà proprio $\gamma(d)$.
- $(a \Rightarrow c)$. Dalla GC sappiamo già che $\alpha(\gamma(d)) \leq_a d$. Dobbiamo allora provare solo che $d \leq_a \alpha(\gamma(d))$. Da (a - suriettività di α) sappiamo che $\forall d\, \exists c \mid \alpha(c) = d$. Dunque, applicando γ, avremo che $\gamma(\alpha(c)) = \gamma(d)$. Dalla GC sappiamo che $c \leq_c \gamma(\alpha(c))$ e dunque $c \leq_c \gamma(d)$. Poiché α è monotòna, $\alpha(c) \leq_a \alpha(\gamma(d))$ ovvero $d \leq_a \alpha(\gamma(d))$.

□

2.3.1. Inserzione di Galois: alcune proprietà. Sia $((C, \leq_c), (A, \leq_a), \gamma, \alpha)$ una GI.

$$\begin{array}{ccc} \gamma(d) & \longleftarrow & d \\ \wr\vee & & \wr\vee \\ c & \longrightarrow & \alpha(c) \end{array}$$

(1) Come già visto, α determina γ e viceversa:

$$\begin{aligned} \alpha(c) &= \bigsqcap \{d \in A \,|\, c \leq_c \gamma(d)\} \\ \gamma(d) &= \bigsqcup \{c \in C \,|\, \alpha(c) \leq_a d\} \end{aligned}$$

(2) α preserva i LUBs e γ preserva i GLBs.
In questo caso (se α preserva i LUBs), è possibile definire γ come sopra, ed essa formerà con α una GI. Stessa cosa vale per γ (se preserva i GLBs).

(3) $\alpha(\bot_c) = \bot_a$ e $\gamma(\top_a) = \top_c$

(4) Poiché α è suriettiva, più punti di C sono mappati su uno stesso punto di A (a meno che non sia biiettiva...). Dunque α induce una equivalenza ($\equiv_\alpha$), e in generale il dominio astratto A di una GI induce una partizione del dominio concreto C di modo che

$$c_1 \equiv_\alpha c_2 \stackrel{def}{=} \alpha(c_1) = \alpha(c_2)$$

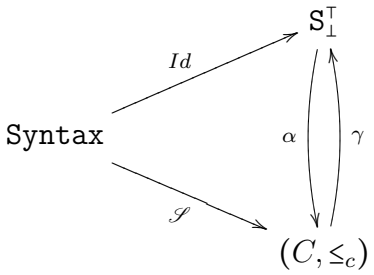
(5) Ogni classe di equivalenza $[c]_{\equiv_\alpha}$ha LUB, cioè ogni classe di equivalenza ha un *rappresentante canonico*

Esempio 2.5. Se consideriamo $C = \wp(\{-1, 0, 1\})$ e come dominio astratto il solito sui segni A allora avremo che

$$\begin{aligned} \alpha(\varnothing) &= \bot \\ \alpha(-1) &= - \\ \alpha(0) &= 0 \\ \alpha(1) &= + \\ \alpha(altrimenti) &= \top \end{aligned}$$

Nell'ultimo caso (cioè tutti gli insiemi di 2 o 3 elementi) vediamo l'equivalenza indotta. Li trattiamo come $\top$. Abbiamo partizionato C in 5 insiemi.

Nota 2.6. Qualunque semantica astratta è la definizione di una partizione. Ad esempio la semantica concreta è un'astrazione della sintassi. Con $\mathtt{S}^\top_\bot$ indichiamo la sintassi piatta.



Osservazione 2.7. La figura seguente riassume le proprietà enunciate in precedenza. TODO

Teorema 2.8 (Correttezza dell'interpretazione astratta)**.** *Teorema e dimostrazione: vedere pp. 19,20,21 di* [6].

CAPITOLO 3

Variabili, espressioni booleane, comandi

3.1. Astrarre sullo stato

Vogliamo ora aggiungere al nostro linguaggio il concetto di variabile modificabile. Sia $\texttt{Var}$ un insieme di variabili, con $x \in \texttt{Var}$ e dunque estendiamo il nostro insieme $\texttt{AExp}$.

$$a ::= \mathtt{n} \mid a \, op \, a \mid x$$

Stato concreto e semantica concreta. Supponiamo di avere un insieme di funzioni

$$\texttt{State} \stackrel{def}{=} \texttt{Var} \to \wp(\mathbb{Z})$$

e assumiamo che ad ogni variabile sia associato un insieme (*singoletto* oppure *vuoto*: lo stato cioè è sempre definito su qualsiasi variabile).

Proposizione 3.1. *Uno stato è detto "coerente" se associa ad ogni variabile l'insieme vuoto o un singoletto.*

A questo punto possiamo ridefinire la funzione semantica $\mathcal{A}$, che ora avrà tipo:

$$\mathcal{A} : \texttt{AExp} \to (\texttt{State} \to \wp(\mathbb{Z}))$$

e sarà così definita

$$\begin{aligned} \mathcal{A}[\![n]\!](s) &= \{n\} \\ \mathcal{A}[\![x]\!](s) &= s(x) \\ \mathcal{A}[\![a_1 \, op \, a_2]\!](s) &= \mathcal{A}[\![a_1]\!](s) \, \dot{op} \, \mathcal{A}[\![a_2]\!](s) \end{aligned}$$

La funzione $\mathcal{A}^A$ viene così ridefinita come

$$\begin{aligned} \mathcal{A}^A[\![n]\!](s^A) &= \alpha(\{n\}) \\ \mathcal{A}^A[\![x]\!](s^A) &= s^A(x) \\ \mathcal{A}^A[\![a_1 \, op \, a_2]\!](s^A) &= \mathcal{A}^A[\![a_1]\!](s^A) \, op^A \, \mathcal{A}^A[\![a_2]\!](s^A) \end{aligned}$$

Osservazione 3.2. Si tratta ora di una funzione totale, a differenza di quanto accadeva nella prima parte del corso, in quanto qui la "parzialità" è catturata dall'insieme vuoto.

Esempio 3.3. Vediamo il comportamento di questa funzione semantica in alcuni casi normali e alcuni anomali

Se $s : x \mapsto \{4\}$ allora $\mathcal{A}[\![x+1]\!](s) = \{4\}\dot{+}\{1\} = \{5\}$

Se $s : x \mapsto \{3, 4\}$ allora $\mathcal{A}[\![x+1]\!](s) = \{4, 5\}$

Se $s : x \mapsto 2\mathbb{Z}$ allora $\mathcal{A}[\![x+1]\!](s) = 2\mathbb{Z}\dot{+}\{1\}$

Lo stato astratto. Sia $(A, \leq)$ il dominio astratto dei segni già visto e sia

$$\texttt{State}^A \stackrel{def}{=} \texttt{Var} \to A$$

un insieme (infinito) di funzioni totali.

Osservazione 3.4. L'altezza di questo insieme è infinita. $\bot$ è la più piccola funzione, $\top$ è la più grande. Ad esempio possiamo definire un dominio ad altezza infinita come

$$\begin{aligned} s_0 &= \bot \\ s_1 &= s_0[x_1 \mapsto +] \\ s_2 &= s_1[x_2 \mapsto +] \\ &\vdots \\ s_n &= s_{n-1}[x_n \mapsto +] \end{aligned}$$

Dunque la ricerca dei punti fissi potrebbe non essere calcolabile in tempo finito.

Fortunatamente, nella nostra analisi lavoreremo sempre con un numero **finito** di variabili (il che è ovvio, visto che un programma ne ha solamente in numero finito). La funzione astratta *stato* in questo caso crescerà finitamente: con n variabili avrà al più 5^n elementi (poiché $|A| = 5$).

Sia inoltre $\leq_s^A$ la relazione d'ordine definita come segue:

$$s_1^A \leq_s^A s_2^A \stackrel{def}{=} \forall x \in \texttt{Var} : \; s_1^A(x) \leq s_2^A(x)$$

e sia corrispondentemente $\leq_s$ la relazione d'ordine definita come segue:

$$s \leq_s s' \stackrel{def}{=} \forall x \in \texttt{Var} : \; s(x) \subseteq s'(x)$$

Definiamo poi due funzioni di concretizzazione e astrazione.

Sia γ_s la funzione di tipo

$$\gamma_s : \texttt{State}^A \to \texttt{State}$$

definita come

$$\gamma_s \stackrel{def}{=} s^A \mapsto [x \mapsto \gamma(s^A(x))]$$

Sia α_s la funzione di tipo

$$\alpha_s : \texttt{State} \to \texttt{State}^A$$

definita come

$$\alpha_s \stackrel{def}{=} s \mapsto [x \mapsto \alpha(s(x))]$$

Osservazione 3.5. Le funzioni γ_s e α_s sono **monotòne**.

Esempio 3.6. Consideriamo alcuni stati astratti.

$$s_1^A: \quad \begin{cases} x \mapsto - \\ Var \setminus \{x\} \mapsto \bot \end{cases}$$

$$s_2^A: \quad \begin{cases} x \mapsto \top \\ Var \setminus \{x\} \mapsto \bot \end{cases}$$

$$s_3^A: \quad \begin{cases} x \mapsto - \\ y \mapsto + \\ Var \setminus \{x, y\} \mapsto \bot \end{cases}$$

Ci accorgiamo subito che $s_1^A \leq_s^A s_2^A$ e che $s_1^A \leq_s^A s_3^A$. Applichiamo ora la funzione di concretizzazione γ_s:

$$\gamma_s(s_1^A): \quad \begin{cases} x \mapsto \mathbb{Z}^- \\ Var \setminus \{x\} \mapsto \varnothing \end{cases}$$

$$\gamma_s(s_2^A): \quad \begin{cases} x \mapsto \mathbb{Z} \\ Var \setminus \{x\} \mapsto \varnothing \end{cases}$$

$$\gamma_s(s_3^A): \quad \begin{cases} x \mapsto \mathbb{Z}^- \\ y \mapsto \mathbb{Z}^+ \\ Var \setminus \{x, y\} \mapsto \varnothing \end{cases}$$

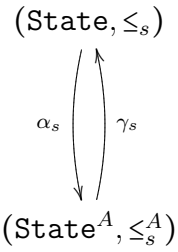
e notiamo quindi che γ_s è monotòna.

Consideriamo ora uno stato concreto s e calcoliamone la sua astrazione:

$$s: \quad \begin{cases} x \mapsto \{-1, 1\} \\ y \mapsto \{-2\} \\ Var \setminus \{x, y\} \mapsto \varnothing \end{cases}$$

$$\alpha_s(s): \quad \begin{cases} x \mapsto \top \\ y \mapsto - \\ Var \setminus \{x, y\} \mapsto \bot \end{cases}$$

Teorema 3.7. *La quadrupla* $((\texttt{State}, \leq_s), (\texttt{State}^A, \leq_s^A), \gamma_s, \alpha_s)$ *definisce un'inserzione di Galois.*



Esercizio. Definirla e dimostrarla.

3.2. La nuova Interpretazione

Consideriamo ora i domini $(\mathtt{State} \to \wp(\mathbb{Z}), \leq_{\to})$ e $(\mathtt{State}^A \to A, \leq^A_{\to})$ che sono insiemi di **funzioni monotòne**.

Definiamo l'ordinamento $\leq_{\to}$ come segue

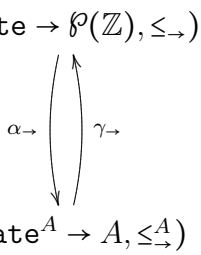
$$f \leq_{\to} g \stackrel{def}{=} \forall s\ f(s) \subseteq g(s)$$

e l'ordinamento $\leq^A_{\to}$ come

$$f^A \leq^A_{\to} g^A \stackrel{def}{=} \forall s^A\ f^A(s^A) \leq g^A(s^A)$$

Esempio 3.8. Esempi di ordinamento. [TODO: DAL QUADERNO]

Esiste una *inserzione di Galois* definita come segue



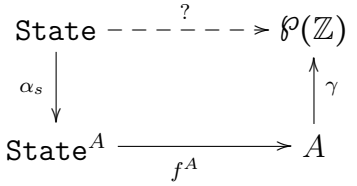
in cui

$$\gamma_{\to} : f^A \mapsto \bigsqcup\{f \,|\, \alpha_{\to}(f) \leq^A_{\to} f^A\}$$

e

$$\alpha_{\to} : f \mapsto \bigsqcap\{f^A \,|\, f \leq_{\to} \gamma_{\to}(f^A)\}$$

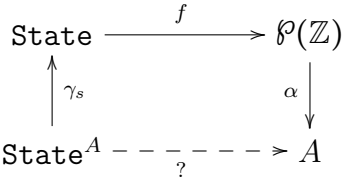
Scritta così è poco comprensibile. Proviamo a dare definizioni *dirette* di $\alpha_{\to}$ e $\gamma_{\to}$. Facciamoci aiutare da semplici diagrammi.



Applicando nell'ordine avremo la definizione di $\gamma_\to$:

$$\gamma_\to : f^A \mapsto [s \mapsto \gamma(f^A(\alpha_s(s)))]$$

Allo stesso modo:



Applicando nell'ordine avremo la definizione di $\alpha_\to$:

$$\alpha_\to : f \mapsto [s^A \mapsto \alpha(f(\gamma_s(s^A)))]$$

Osservazione 3.9. Condizione necessaria affinché $\gamma_\to$ e $\alpha_\to$ siano **monotòne** è che sia f che f^A lo siano.

Esercizio 3.10. Dimostrare l'equivalenza delle definizioni automatiche di $\gamma_\to$ e $\alpha_\to$ con quelle dirette date ora.

Esempio 3.11. Esempi di uso di funzioni ed assurdo se non fosse monotona. [TODO: DAL QUADERNO]

Teorema 3.12 (Correttezza di $\mathcal{A}^A$ rispetto ad $\mathcal{A}$)**.** *Per ogni* $a \in \texttt{AExp}$

$$\mathcal{A}[\![a]\!] \leq_\to \gamma_\to(\mathcal{A}^A[\![a]\!])$$

Dimostrazione. Per induzione sulla struttura di a. Semplifichiamo l'enunciato:

$$\begin{array}{rcl} & & \mathcal{A}[\![a]\!] \leq_\to \gamma_\to(\mathcal{A}^A[\![a]\!]) \\ \text{per definizione di } \leq_\to & \iff & \forall s \in \texttt{State} : \mathcal{A}[\![a]\!](s) \subseteq \gamma_\to(\mathcal{A}^A[\![a]\!](s)) \\ \text{per definizione di } \gamma_\to & \iff & \forall s \in \texttt{State} : \mathcal{A}[\![a]\!](s) \subseteq \gamma(\mathcal{A}^A[\![a]\!]\alpha_s(s)) \end{array}$$

Caso base $a = n$**.** Avremo $\mathcal{A}[\![n]\!](s) = \{n\}$ mentre $\gamma(\mathcal{A}^A[\![n]\!]\alpha_s(s)) = \gamma(\alpha(\{n\}))$ e dunque, per definizione di GC, $\{n\} \subseteq \gamma(\alpha(\{n\}))$.

Caso base $a = x$**.** Avremo

$$\begin{array}{rcl} \mathcal{A}[\![x]\!](s) & = & s(x) \\ \text{perché è una GC} & \subseteq & \gamma(\alpha(s(x))) \\ \text{per def. di } \alpha_s & = & \gamma(\alpha_s(s)(x)) \\ \text{per def. di } \mathcal{A}^A & = & \gamma(\mathcal{A}^A[\![x]\!](\alpha_s(s))) \end{array}$$

Caso induttivo $a = a' \, op \, a''$**.** Avremo

$$\begin{array}{rcl} \mathcal{A}[\![a' \, op \, a'']\!](s) & = & \mathcal{A}[\![a']\!](s) \, \dot{op} \, \mathcal{A}[\![a'']\!](s) \\ \text{per ipotesi induttiva e monotonia di } \dot{op} & \subseteq & \gamma(\mathcal{A}^A[\![a']\!](\alpha_s(s))) \, \dot{op} \, \gamma(\mathcal{A}^A[\![a'']\!](\alpha_s(s))) \\ \text{per correttezza locale di } op^A & \subseteq & \gamma(\mathcal{A}^A[\![a']\!](\alpha_s(s)) \, op^A \, \mathcal{A}^A[\![a'']\!](\alpha_s(s))) \\ \text{per def. di } \mathcal{A}^A & = & \gamma(\mathcal{A}^A[\![a' \, op \, a'']\!](\alpha_s(s))) \end{array}$$

□

3.3. La semantica delle espressioni booleane

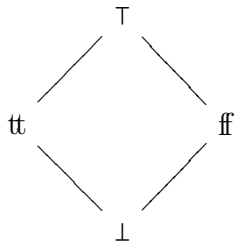
Semantica concreta. Sia `BExp` la categoria sintattica delle espressioni booleane, rappresentate da b. La sintassi è definita dalle regole

$$b ::= true \,|\, false \,|\, a = a' \,|\, a \leq a' \,|\, \neg b \,|\, b \wedge b$$

e la semantica concreta (vedi [1], pagg. 9-13) è definita dalla funzione semantica

$$\mathcal{B} : \texttt{BExp} \to (\texttt{State} \to \mathbb{T})$$

dove $(\mathbb{T}, \leq_{\mathbb{T}})$ è il seguente reticolo completo (in cui $\bot$ rappresenta la *non definizione* e $\top$ rappresenta il fatto di non conoscere il valore di una espressione booleana):



Esempio 3.13. Alcuni casi:

$\{1\} \doteq \{1\}$ restituirà tt

$\{1\} \doteq \{2\}$ restituirà ff

$\{1\} \doteq \varnothing$ restituirà $\bot$

$\{1, 2\} \doteq \{1, 2\}$ restituirà $\top$, infatti è ND... ho contemporaneamente $1 = 1$ e $1 = 2$

$if \, x = y \, then \, x + 1 \, else \, y - 1$ se non so se $x = y$, dovrò restituire $\bigsqcup\{x + 1, \, y + 1\}$...

Definiamo la funzione $\mathcal{B}$ per induzione sulla sintassi, come segue:

$$\begin{aligned}
\mathcal{B}[\![true]\!](s) &= \mathtt{tt} \\
\mathcal{B}[\![false]\!](s) &= \mathtt{ff} \\
\mathcal{B}[\![a = a']\!](s) &= \mathcal{A}[\![a]\!](s) \doteq \mathcal{A}[\![a']\!](s) \\
\mathcal{B}[\![a \leq a']\!](s) &= \mathcal{A}[\![a]\!](s) \dot{\leq} \mathcal{A}[\![a']\!](s) \\
\mathcal{B}[\![\neg b]\!](s) &= \dot{\neg}\mathcal{B}[\![b]\!](s) \\
\mathcal{B}[\![b \wedge b']\!](s) &= \mathcal{B}[\![b]\!](s) \dot{\wedge} \mathcal{B}[\![b']\!](s)
\end{aligned}$$

Fatto 3.14. *Se $s(x) \subseteq \{\{n\} \mid n \in \mathbb{Z}\}$ allora $\forall b\ \mathcal{B}[\![b]\!](s) \neq \top$, ovvero $\top$ non è immagine di alcuna espressione booleana e di alcuno stato*

Dimostrazione. È lasciata al lettore per esercizio. □

Definiamo ora gli operatori semantici su $\wp(\mathbb{Z})$.

Per l'uguaglianza avremo

$$X \doteq Y \stackrel{def}{=} \begin{cases} \bot & se\ X \equiv \varnothing\ o\ Y \equiv \varnothing \\ \mathtt{tt} & se\ X \equiv \{n\} \equiv Y \\ \mathtt{ff} & se\ X \cap Y \equiv \varnothing \\ \top & altrimenti \end{cases}$$

Per la disuguaglianza

$$X \dot{\leq} Y \stackrel{def}{=} \begin{cases} \bot & se\ X \equiv \varnothing\ o\ Y \equiv \varnothing \\ \mathtt{tt} & se\ \forall x \in X\ \forall y \in Y\ x \tilde{\leq} y \\ \mathtt{ff} & se\ \forall x \in X\ \forall y \in Y\ x \tilde{>} y \\ \top & altrimenti \end{cases}$$

Per la negazione

$\dot{\neg}$	$\bot$	$\mathtt{tt}$	$\mathtt{ff}$	$\top$
	$\bot$	$\mathtt{ff}$	$\mathtt{tt}$	$\top$

e per la congiunzione

$\dot{\wedge}$	$\bot$	$\mathtt{tt}$	$\mathtt{ff}$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\mathtt{tt}$	$\bot$	$\mathtt{tt}$	$\mathtt{ff}$	$\top$
$\mathtt{ff}$	$\bot$	$\mathtt{ff}$	$\mathtt{ff}$	$\mathtt{ff}$
$\top$	$\bot$	$\top$	$\mathtt{ff}$	$\top$

Semantica astratta. La semantica astratta sarà definita da una funzione $\mathcal{B}^A$ con tipo

$$\mathcal{B}^A : \mathtt{BExp} \to (\mathtt{State}^A \to \mathbb{T})$$

nel seguente modo

$$\begin{aligned}
\mathcal{B}^A[\![true]\!](s^A) &= \mathtt{tt} \\
\mathcal{B}^A[\![false]\!](s^A) &= \mathtt{ff} \\
\mathcal{B}^A[\![a = a']\!](s^A) &= \mathcal{A}^A[\![a]\!](s^A) =^A \mathcal{A}^A[\![a']\!](s^A) \\
\mathcal{B}^A[\![a \leq a']\!](s^A) &= \mathcal{A}^A[\![a]\!](s^A) \leq^A \mathcal{A}^A[\![a']\!](s^A) \\
\mathcal{B}^A[\![\neg b]\!](s^A) &= \neg^A \mathcal{B}^A[\![b]\!](s^A) \\
\mathcal{B}^A[\![b \wedge b']\!](s^A) &= \mathcal{B}^A[\![b]\!](s^A) \wedge^A \mathcal{B}^A[\![b']\!](s^A)
\end{aligned}$$

Osservazione 3.15. Poiché $\mathbb{T}$ è lo stesso dominio, $\neg^A \stackrel{def}{=} \dot{\neg}$ e $\wedge^A \stackrel{def}{=} \dot{\wedge}$. Dobbiamo invece ridefinire $=^A$ e $\leq^A$ su $(A, \leq)$. Avremo

$=^{\mathbf{A}}$	$\bot$	**-**	**0**	+	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
-	$\bot$	$\top$	ff	ff	$\top$
0	$\bot$	ff	tt	ff	$\top$
+	$\bot$	ff	ff	$\top$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$

e, considerando il primo argomento sulla prima colonna e il secondo argomento sulla prima riga,

$\leq^{\mathbf{A}}$	$\bot$	**-**	**0**	+	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
-	$\bot$	$\top$	tt	tt	$\top$
0	$\bot$	ff	tt	tt	$\top$
+	$\bot$	ff	ff	$\top$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$

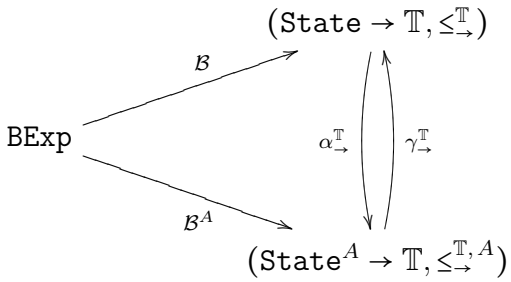
Osservazione 3.16. Tutti gli operatori astratti sono monotoni.

Teorema 3.17. *La quadrupla* $((\mathtt{State} \to \mathbb{T}, \leq^{\mathbb{T}}_{\to}), (\mathtt{State}^A \to \mathbb{T}, \leq^{\mathbb{T},A}_{\to}), \gamma^{\mathbb{T}}_{\to}, \alpha^{\mathbb{T}}_{\to})$ *definisce un'inserzione di Galois.*

Dimostrazione. È lasciata al lettore per esercizio. □

Possiamo definire le funzioni $\gamma^{\mathbb{T}}_{\to}$ e $\alpha^{\mathbb{T}}_{\to}$ nel modo seguente:

$$\begin{aligned}
\gamma^{\mathbb{T}}_{\to} &\stackrel{def}{=} p^A \to [s \to p^A(\alpha_s(s))] \\
\alpha^{\mathbb{T}}_{\to} &\stackrel{def}{=} p \to [s^A \to p(\gamma_s(s^A))]
\end{aligned}$$



Fatto 3.18. *Occorre dimostrare la correttezza locale degli operatori, per poter dimostrare poi la correttezza di $\mathcal{B}^A$ rispetto a $\mathcal{B}$, ovvero*

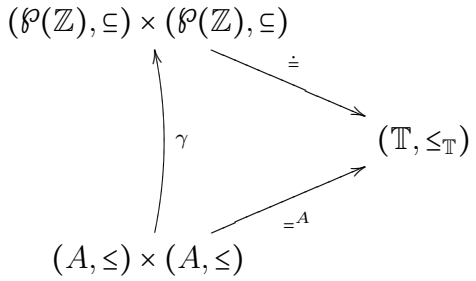
$$\mathcal{B}[\![b]\!] \leq^{\mathbb{T}}_{\rightarrow} \gamma^{\mathbb{T}}_{\rightarrow}(\mathcal{B}^A[\![b]\!])$$

Dimostrazione. Ci si riduce a dimostrare la correttezza locale delle operazioni seguenti (basta fare un'analisi per casi sulla struttura di b) □

Proposizione 3.19 (Correttezza locale di $\leq^A$ rispetto a $\dot{\leq}$ e di $=^A$ rispetto a $\dot{=}$)**.** *Per ogni $d, d' \in A$ vale che*

$$\begin{aligned} \gamma(d)\dot{=}\gamma(d') &\leq_{\mathbb{T}} id(d =^A d') \\ \gamma(d)\dot{\leq}\gamma(d') &\leq_{\mathbb{T}} id(d \leq^A d') \end{aligned}$$

dove avremmo dovuto definire le funzioni $\alpha_{\mathbb{T}}$ e $\gamma_{\mathbb{T}}$, ma ci accorgiamo immediatamente che esse equivalgono alla funzione identità, in quanto $\mathbb{T}$ è lo stesso dominio, sia astratto che concreto.



Dimostrazione. Per casi sul valore di d e d'.

Con la notazione $\boxed{a/b}$ stiamo ad indicare che $a \leq_{\mathbb{T}} b$. □

$\dot{=}/_{=^{\mathbf{A}}}$	$\bot$	**-**	**0**	**+**	$\top$
$\bot$	$\bot/\bot$	$\bot/\bot$	$\bot/\bot$	$\bot/\bot$	$\bot/\bot$
-	$\bot/\bot$	$\top/\top$	$\mathbf{ff}/\mathbf{ff}$	$\mathbf{ff}/\mathbf{ff}$	$\top/\top$
0	$\bot/\bot$	$\mathbf{ff}/\mathbf{ff}$	$\mathbf{tt}/\mathbf{tt}$	$\mathbf{ff}/\mathbf{ff}$	$\top/\top$
+	$\bot/\bot$	$\mathbf{ff}/\mathbf{ff}$	$\mathbf{ff}/\mathbf{ff}$	$\top/\top$	$\top/\top$
$\top$	$\bot/\bot$	$\top/\top$	$\top/\top$	$\top/\top$	$\top/\top$

$\dot{\leq}/_{\leq^{\mathbf{A}}}$	$\bot$	-	**0**	+	$\top$
$\bot$	$\bot/_{\bot}$	$\bot/_{\bot}$	$\bot/_{\bot}$	$\bot/_{\bot}$	$\bot/_{\bot}$
-	$\bot/_{\bot}$	$\top/_{\top}$	$\mathtt{tt}/_{\mathtt{tt}}$	$\mathtt{tt}/_{\mathtt{tt}}$	$\top/_{\top}$
0	$\bot/_{\bot}$	$\mathtt{ff}/_{\mathtt{ff}}$	$\mathtt{tt}/_{\mathtt{tt}}$	$\mathtt{tt}/_{\mathtt{tt}}$	$\top/_{\top}$
+	$\bot/_{\bot}$	$\mathtt{ff}/_{\mathtt{ff}}$	$\mathtt{ff}/_{\mathtt{ff}}$	$\top/_{\top}$	$\top/_{\top}$
$\top$	$\bot/_{\bot}$	$\top/_{\top}$	$\top/_{\top}$	$\top/_{\top}$	$\top/_{\top}$

3.4. La semantica dei comandi

Iniziamo con una sottocategoria sintattica `Stm`

$$S ::= skip \,|\, x := a \,|\, S; S$$

Semantica concreta. La semantica concreta $\mathcal{S} : \mathtt{Stm} \to (\mathtt{State} \hookrightarrow \mathtt{State})$ è definita nel modo seguente:

$$\begin{aligned}
\mathcal{S}[\![skip]\!] &= id \\
\mathcal{S}[\![x := a]\!] &= \begin{cases} s \mapsto s[x \mapsto \mathcal{A}[\![a]\!](s)] & se\ \mathcal{A}[\![a]\!](s) \neq \varnothing \\ \mathsf{undef} & altrimenti \end{cases} \\
\mathcal{S}[\![S; S']\!] &= \mathcal{S}[\![S']\!] \circ \mathcal{S}[\![S]\!]
\end{aligned}$$

Proposizione 3.20. *Bisogna osservare che nel nostro caso*

$$\mathtt{State} = \mathtt{Var} \to \wp(\mathbb{Z})$$

e con l'introduzione delle variabili se lo stato associa a una variabile un insieme non-vuoto e non-singoletto allora $\mathcal{A}$ può produrre insiemi non vuoto e non singoletto.

Analizziamo ora il reticolo completo $(\mathtt{State} \hookrightarrow \mathtt{State}, \leq_{\hookrightarrow})$. Osserviamo subito che

- $\bot \stackrel{def}{=} \varnothing$
- $\top \stackrel{def}{=} s \mapsto [x \mapsto \mathbb{Z}] \forall x$ *ovvero* $\top(s)(x) = \mathbb{Z}$

Siano $f, g \in (\mathtt{State} \hookrightarrow \mathtt{State})$. L'ordinamento parziale $\leq_{\hookrightarrow}$ è definito nel modo seguente

$$f \leq_{\hookrightarrow} g \stackrel{def}{=} \forall s \in dom(f)\ f(s) \leq_s g(s)$$

Notiamo che $\forall s \in dom(f)$ sta ad indicare "dove f è definita". Se f non è definita, la disuguaglianza è vacuamente vera.

Esempio 3.21. Consideriamo le funzioni

$$\begin{aligned}
s_1 &: \forall x \quad x \mapsto \{0\} \\
s_2 &: \forall x \quad x \mapsto \{0, 1\}
\end{aligned}$$

e ora analizziamo le funzioni $f : s \mapsto s_1$, $g : s \mapsto s_2$, $\bot$ e $\top$. Avremo

$$\begin{aligned} \bot &\leq_{\hookrightarrow} f \\ \bot &\leq_{\hookrightarrow} g \\ f &\leq_{\hookrightarrow} g \\ g &\leq_{\hookrightarrow} \top \\ f \circ g &\leq_{\hookrightarrow} g \circ f \end{aligned}$$

Semantica astratta. La semantica astratta $\mathcal{S}^A : \mathtt{Stm} \to (\mathtt{State}^A \hookrightarrow \mathtt{State}^A)$ è definita nel modo seguente:

$$\begin{aligned} \mathcal{S}^A[\![skip]\!] &= id \\ \mathcal{S}^A[\![x := a]\!] &= \begin{cases} s^A \mapsto s^A[x \mapsto \mathcal{A}^A[\![a]\!](s^A)] & se\ \mathcal{A}^A[\![a]\!](s^A) \neq \bot \\ \mathsf{undef} & altrimenti \end{cases} \\ \mathcal{S}^A[\![S; S']\!] &= \mathcal{S}^A[\![S']\!] \circ \mathcal{S}^A[\![S]\!] \end{aligned}$$

Analizziamo ora il reticolo completo $(\mathtt{State}^A \hookrightarrow \mathtt{State}^A, \leq^A_{\hookrightarrow})$.

Siano $f^A, g^A \in (\mathtt{State}^A \hookrightarrow \mathtt{State}^A)$. L'ordinamento parziale $\leq^A_{\hookrightarrow}$ è definito nel modo seguente

$$f^A \leq^A_{\hookrightarrow} g^A \stackrel{def}{=} \forall s^A \in dom(f^A)\ f^A(s^A) \leq^A_s g^A(s^A)$$

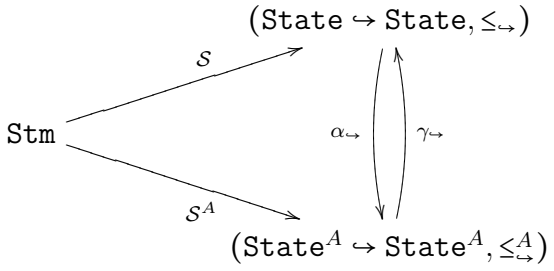
Possiamo definire le funzioni $\gamma_{\hookrightarrow}$ e $\alpha_{\hookrightarrow}$ nel modo seguente:

$$\gamma_{\hookrightarrow} \stackrel{def}{=} \begin{cases} f^A \to [s \to \gamma_s(f^A(\alpha_s(s)))] & se\, f^A(\alpha_s(s))\, è\, definita \\ f^A \to [s \to \mathsf{undef}] & altrimenti \end{cases}$$

$$\alpha_{\hookrightarrow} \stackrel{def}{=} \begin{cases} f \to [s^A \to \alpha_s(f(\gamma_s(s^A)))] & se\, f(\gamma_s(s^A))\, è\, definita \\ f \to [s^A \to \mathsf{undef}] & altrimenti \end{cases}$$

Teorema 3.22. *La quadrupla* $((\mathtt{State} \hookrightarrow \mathtt{State}, \leq_{\hookrightarrow}), (\mathtt{State}^A \hookrightarrow \mathtt{State}^A, \leq^A_{\hookrightarrow}), \gamma_{\hookrightarrow}, \alpha_{\hookrightarrow})$ *definisce un'inserzione di Galois.*

Dimostrazione. È lasciata al lettore per esercizio. □



Teorema 3.23 (Correttezza di $\mathcal{S}^A$ rispetto a $\mathcal{S}$)**.** *Se $\mathcal{S}[\![S]\!]$ è definita (altrimenti il teorema è banalmente vero),*

$$\mathcal{S}[\![S]\!] \leq_{\hookrightarrow} \gamma_{\hookrightarrow}(\mathcal{S}^A[\![S]\!])$$

ovvero

$$\forall s\, t.c.\, s \in dom(\mathcal{S}[\![S]\!])\ \ \mathcal{S}[\![S]\!](s) \leq_s \gamma_s(\mathcal{S}^A[\![S]\!](\alpha_s(s))$$

Dimostrazione. Per induzione strutturale su S.

Caso base $S = skip$**.** Avremo

$$\begin{array}{rcl} \mathcal{S}[\![skip]\!](s) & = & s \\ \text{per definizione di GC} & \leq_s & \gamma_s(\alpha_s(s)) \\ \text{per definizione di } \mathcal{S}^A & = & \gamma_s(\mathcal{S}^A[\![S]\!](\alpha_s(s))) \end{array}$$

Caso base $S = x := a$**.** Avremo e

$$\begin{array}{rcl} \mathcal{S}[\![x := a]\!](s) & = & s[x \mapsto \mathcal{A}[\![a]\!](s)] \\ \text{per correttezza di } \mathcal{A}^A\text{rispetto ad } \mathcal{A} & \leq_s & s[x \mapsto \gamma(\mathcal{A}^A[\![a]\!](\alpha_s(s)))] \\ \text{per GC} & \leq_s & \gamma_s(\alpha_s(s[x \mapsto \gamma(\mathcal{A}^A[\![a]\!](\alpha_s(s)))])) \\ \text{per def. di } \alpha_s & = & \gamma_s(\alpha_s(s)[x \mapsto \alpha(\gamma(\mathcal{A}^A[\![a]\!](\alpha_s(s))))]) \\ \text{per GI} & = & \gamma_s(\alpha_s(s)[x \mapsto \mathcal{A}^A[\![a]\!](\alpha_s(s))]) \\ \text{per definizione di } \mathcal{S}^A & = & \gamma_s(\mathcal{S}^A[\![x := a]\!](\alpha_s(s)) \end{array}$$

Caso induttivo $S = S'; S''$**.** Per IH sappiamo già che

$$\begin{array}{rcl} \mathcal{S}[\![S']\!](s) & \leq_s & \gamma_s(\mathcal{S}^A[\![S']\!](\alpha_s(s)) \\ \mathcal{S}[\![S'']\!](s) & \leq_s & \gamma_s(\mathcal{S}^A[\![S'']\!](\alpha_s(s)) \end{array}$$

e diremo che

$$\begin{array}{rcl} \mathcal{S}[\![S'; S'']\!](s) & = & \mathcal{S}[\![S'']\!](\mathcal{S}[\![S']\!](s)) \\ \text{per IH1 e monotonia di } \mathcal{S} & \leq_s & \mathcal{S}[\![S'']\!](\gamma_s(\mathcal{S}^A[\![S']\!](\alpha_s(s))) \\ \text{per IH2} & \leq_s & \gamma_s(\mathcal{S}^A[\![S'']\!](\alpha_s(\gamma_s(\mathcal{S}^A[\![S']\!](\alpha_s(s)))) \\ \text{per GC e monotonia di } \mathcal{S}^A & \leq_s & \gamma_s(\mathcal{S}^A[\![S'']\!](\mathcal{S}^A[\![S']\!](\alpha_s(s)) \\ \text{per def. di } \mathcal{S}^A & = & \gamma_s(\mathcal{S}^A[\![S'; S'']\!](\alpha_s(s))) \end{array}$$

□

Osservazione. Non essendoci operatori, non ci siamo più serviti della *correttezza locale* in queste dimostrazioni.

3.5. La semantica del comando condizionale

Aggiungiamo il costrutto condizionale alla categoria `Stm`

$$S ::= \ldots \mid if\ b\ then\ S\ else\ S$$

Semantica concreta. Estendiamo la semantica concreta $\mathcal{S} : \texttt{Stm} \to (\texttt{State} \hookrightarrow \texttt{State})$ nel modo seguente:

$$\mathcal{S}[\![if\ b\ then\ S\ else\ S']\!] \quad = \quad \dot{cond}(\mathcal{B}[\![b]\!], \mathcal{S}[\![S]\!], \mathcal{S}[\![S']\!])$$

in cui $\dot{cond}$ è la funzione semantica che ha tipo

$$\dot{cond} : ((\texttt{State} \to \mathbb{T}) \times (\texttt{State} \hookrightarrow \texttt{State}) \times (\texttt{State} \hookrightarrow \texttt{State})) \to (\texttt{State} \hookrightarrow \texttt{State})$$

e, dove $p(s)$ è definito, sarà definita come

$$\dot{cond}(p,\ f,\ g)(s) = \begin{cases} f(s) & se\ p(s) = tt \\ g(s) & se\ p(s) = ff \\ f(s) \sqcup g(s) & se\ p(s) = \top \end{cases}$$

Semantica astratta. La semantica astratta $\mathcal{S}^A : \texttt{Stm} \to (\texttt{State}^A \hookrightarrow \texttt{State}^A)$ sarà analogamente estesa come:

$$\mathcal{S}^A[\![if\ b\ then\ S\ else\ S']\!] \quad = \quad cond^A(\mathcal{B}^A[\![b]\!], \mathcal{S}^A[\![S]\!], \mathcal{S}^A[\![S']\!])$$

in cui $cond^A$ è la funzione che ha tipo

$$cond^A : ((\texttt{State}^A \to \mathbb{T}) \times (\texttt{State}^A \hookrightarrow \texttt{State}^A) \times (\texttt{State}^A \hookrightarrow \texttt{State}^A)) \to (\texttt{State}^A \hookrightarrow \texttt{State}^A)$$

e, dove $p^A(s^A)$ è definito, sarà definita come

$$cond^A(p^A,\ f^A,\ g^A)(s^A) = \begin{cases} f^A(s^A) & se\ p^A(s^A) = tt \\ g^A(s^A) & se\ p^A(s^A) = ff \\ f^A(s^A) \sqcup g^A(s^A) & se\ p^A(s^A) = \top \end{cases}$$

Esempio 3.24. TODO ricopia dal quaderno

Proposizione 3.25 (Correttezza locale di $cond^A$)**.** *Vale che*

$$\dot{cond}(p,\ f,\ g) \leq_{\hookrightarrow} \gamma_{\hookrightarrow}(cond^A(\alpha^{\mathbb{T}}_{\to}(p),\ \alpha_{\hookrightarrow}(f),\ \alpha_{\hookrightarrow}(g)))$$

Dimostrazione. Per definizione di $\leq_{\hookrightarrow}$ e $\gamma_{\hookrightarrow}$, dobbiamo provare che, $\forall s \in \texttt{State}$, se $cond(p,\ f,\ g)(s)$ è definita, allora

$$\dot{cond}(p,\ f,\ g) \leq_s \gamma_s(cond^A(\alpha^{\mathbb{T}}_{\to}(p),\ \alpha_{\hookrightarrow}(f),\ \alpha_{\hookrightarrow}(g))(\alpha_s(s)))$$

Procediamo quindi *per casi* sui possibili risultati di $p(s)$.

Caso $p(s) = \bot$**.** $\dot{cond}$ è undef e dunque non ho nulla da dimostrare.

Caso $p(s) = \mathbb{tt}$**.** Avremo

$$\begin{array}{rcl} \dot{cond}(p, f, g)(s) & = & f(s) \\ \text{per monotonia e GC} & \leq_s & \gamma_{\hookrightarrow}(\alpha_{\hookrightarrow}(f))(s) \\ \text{per definizione di } \gamma_{\hookrightarrow} & = & \gamma_s(\alpha_{\hookrightarrow}(f)(\alpha_s(s))) \\ \text{per } (*) & \leq_s & \gamma_s(cond^A(\alpha^{\mathbb{T}}_{\rightarrow}(p), \alpha_{\hookrightarrow}(f), \alpha_{\hookrightarrow}(g))(\alpha_s(s))) \end{array}$$

Rimane da provare (*). Intuitivamente, essendo nel caso $p(s) = \mathbb{tt}$, $\alpha^{\mathbb{T}}_{\rightarrow}(p)(s^A)$ sarà $\mathbb{tt}$ o $\top$ e dunque il $cond^A$ restituirà o il valore preciso o il LUB tra i due casi, rimanendo comunque *più grande*. Formalmente

$$\begin{array}{rcl} \mathbb{tt} & = & p(s) \\ \text{per monotonia e GC} & \leq_{\mathbb{T}} & \gamma^{\mathbb{T}}_{\rightarrow}(\alpha^{\mathbb{T}}_{\rightarrow}(p))(s) \\ \text{per definizione di } \gamma^{\mathbb{T}}_{\rightarrow} & = & \alpha^{\mathbb{T}}_{\rightarrow}(p)(\alpha_s(s)) \end{array}$$

e si estende banalmente a $cond^A$.

Caso $p(s) = \mathbb{ff}$**.** Del tutto analogo al precedente. □

Lemma 3.26. *Come immediata conseguenza della Proposizione, otteniamo che*

$$\mathcal{S}[\![if\ b\ then\ S\ else\ S']\!] \leq_{\hookrightarrow} \gamma_{\hookrightarrow}(\mathcal{S}^A[\![if\ b\ then\ S\ else\ S']\!])$$

Esempio 3.27. Consideriamo il comando

$$T = if\ 0 \leq x\ then\ z := x + 1\ else\ z := x * x$$

e vogliamo dimostrare che T valuta sempre a uno stato in cui z ha un valore *strettamente positivo.* Cominciamo con l'analizzare la semantica concreta di T su alcuni stati di esempio

$$\begin{array}{rcl} s_1 & = & [x \mapsto \{0\}] \\ s_2 & = & [x \mapsto \{-1, 1\}] \\ s_3 & = & [x \mapsto \{-1\}] \end{array}$$

e calcoliamo ora il valore della guardia nei 3 diversi casi

$$\begin{array}{rcl} \mathcal{B}[\![0 \leq x]\!](s_1) & = & \mathbb{tt} \\ \mathcal{B}[\![0 \leq x]\!](s_2) & = & \top \\ \mathcal{B}[\![0 \leq x]\!](s_1) & = & \mathbb{ff} \end{array}$$

e calcoliamo ora la semantica concreta

$$\begin{aligned}
\mathcal{S}[\![T]\!](s_1) &= \mathcal{S}[\![z := x+1]\!](s_1) = s_1[z \mapsto \{1\}] \\
\mathcal{S}[\![T]\!](s_2) &= \mathcal{S}[\![z := x+1]\!](s_2) \sqcup \mathcal{S}[\![z := x * x]\!](s_2) \\
&= s_2[z \mapsto \{0,2\}] \sqcup s_2[z \mapsto \{1\}] \\
&= s_2[z \mapsto \{0,1,2\}] \\
\mathcal{S}[\![T]\!](s_3) &= \mathcal{S}[\![z := x+1]\!](s_3) = s_3[z \mapsto \{1\}]
\end{aligned}$$

Svolgiamo di nuovo gli stessi calcoli con la semantica astratta:

$$\begin{aligned}
\alpha_s(s_1) &= [x \mapsto 0] \\
\alpha_s(s_2) &= [x \mapsto \top] \\
\alpha_s(s_3) &= [x \mapsto -]
\end{aligned}$$

e calcoliamo ora il valore della guardia nei 3 diversi casi

$$\begin{aligned}
\mathcal{B}^A[\![0 \leq x]\!](\alpha_s(s_1)) &= \mathtt{tt} \\
\mathcal{B}^A[\![0 \leq x]\!](\alpha_s(s_2)) &= \top \\
\mathcal{B}^A[\![0 \leq x]\!](\alpha_s(s_1)) &= \mathtt{ff}
\end{aligned}$$

e calcoliamo ora la semantica

$$\begin{aligned}
\mathcal{S}^A[\![T]\!](\alpha_s(s_1)) &= \alpha_s(s_1)[z \mapsto +] \\
\mathcal{S}^A[\![T]\!](\alpha_s(s_2)) &= \mathcal{S}^A[\![z := x+1]\!](\alpha_s(s_2)) \sqcup \mathcal{S}^A[\![z := x * x]\!](\alpha_s(s_2)) \\
&= (\alpha_s(s_2))[z \mapsto \top] \sqcup (\alpha_s(s_2))[z \mapsto \top] \\
&= (\alpha_s(s_2))[z \mapsto \top] \\
\mathcal{S}^A[\![T]\!](\alpha_s(s_3)) &= (\alpha_s(s_3))[z \mapsto +]
\end{aligned}$$

Osservazione. Se lo stato concreto associa alle variabili solo singoletti o l'insieme vuoto, come già osservato in 1.3, 3.1, 3.14 e 3.20, anche la semantica rimane "coerente", cioè associa solo insieme vuoto o singoletti alle variabili.
Moralmente: la semantica concreta definita è conforme a quella studiata nel primo semestre, ed è anzi più elegante in quanto per la non terminazione utilizza funzioni totali e insieme vuoto, insieme che funzioni parziali.
Questo non ci dice niente sull'astrazione (che potrebbe comunque essere *meno precisa*)!

3.6. La semantica del while

Aggiungiamo il costrutto iterativo alla categoria `Stm`

$$S ::= \ldots \mid \mathit{while}\, b\, \mathit{do}\, S$$

Semantica concreta. Estendiamo la semantica concreta $\mathcal{S} : \texttt{Stm} \to (\texttt{State} \hookrightarrow \texttt{State})$ nel modo seguente:

$$\begin{aligned} \mathcal{S}[\![while\, b\, do\, S]\!] &= Fix(F) \\ dove &\quad F(g) = \dot{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}[\![S]\!], id) \end{aligned}$$

e in cui

$$Fix(F) = \bigsqcup_{n\in\mathbb{N}} F^n(\bot)$$

Semantica astratta. La semantica astratta $\mathcal{S}^A : \texttt{Stm} \to (\texttt{State}^A \hookrightarrow \texttt{State}^A)$ sarà analogamente estesa come:

$$\begin{aligned} \mathcal{S}^A[\![while\, b\, do\, S]\!] &= Fix(F_A) \\ dove &\quad F_A(g^A) = cond^A(\mathcal{B}^A[\![b]\!], g^A \circ \mathcal{S}^A[\![S]\!], id^A) \end{aligned}$$

e in cui

$$Fix(F_A) = \bigsqcup_{n\in\mathbb{N}} F_A^n(\bot)$$

Lemma 3.28. *Se so dimostrare che*

$$\forall n \in \mathbb{N}\ f_n \leq g_n$$

allora avrò che

$$\bigsqcup_{n\in\mathbb{N}} f_n \leq \bigsqcup_{n\in\mathbb{N}} g_n$$

Dimostrazione. È sufficiente provare che $\bigsqcup_{n\in\mathbb{N}} g_n$ è un maggiorante della catena $\{f_1, f_2 ...\}$ cioè

$$\forall i\ f_i \leq \bigsqcup_{n\in\mathbb{N}} g_n$$

Per ipotesi sappiamo che $f_i \leq g_i$ e per definizione di $\sqcup$ sappiamo che $g_i \leq \bigsqcup_{n\in\mathbb{N}} g_n$. Per transitività otteniamo quanto cercato. □

Teorema 3.29 (Correttezza di $\mathcal{S}^A$)**.** *Vale che*

$$\begin{array}{rcl} \mathcal{S}[\![while\, b\, do\, S]\!] & \leq_{\hookrightarrow} & \gamma_{\hookrightarrow}(\mathcal{S}^A[\![while\, b\, do\, S]\!]) \\ = & & = \\ Fix(F) & & \gamma_{\hookrightarrow}(Fix(F_A)) \\ = & & = \\ \bigsqcup_{n\in\mathbb{N}} F^n(\bot) & & \gamma_{\hookrightarrow}(\bigsqcup_{n\in\mathbb{N}} F_A^n(\bot)) \end{array}$$

Osservazione. Le due funzioni $\bot$ della parte sinistra e destra sono rispettivamente il *bottom* dei reticoli $(\texttt{State} \hookrightarrow \texttt{State}, \leq_{\hookrightarrow})$ e $(\texttt{State}^A \hookrightarrow \texttt{State}^A, \leq^A_{\hookrightarrow})$

DIMOSTRAZIONE. Dal lemma 3.28 mi riduco a dimostrare l'enunciato seguente (ovvero: senza guardare i punti fissi, mi riduco a dimostrare la correttezza sugli approssimanti finiti):

$$F^n(\bot) \quad \leq_{\hookrightarrow} \quad \gamma_{\hookrightarrow}(F^n_A(\bot))$$

NB: siamo all'interno di una dimostrazione per induzione su S ($\mathcal{S}[\![S]\!] \leq_{\hookrightarrow} \gamma_{\hookrightarrow}(\mathcal{S}^A[\![S]\!])$) e procediamo ora *per induzione matematica* su n.

Caso base $n = 0$**.** $F^0(\bot) = \bot \leq_{\hookrightarrow} \gamma_{\hookrightarrow}(F^0_A(\bot))$. Vale per ogni $\gamma_{\hookrightarrow}(F^n_A(\bot))$ per definizione di $\bot$ come elemento più piccolo rispetto a $\leq_{\hookrightarrow}$.

Caso induttivo $n+1$**.** Per IH sappiamo che $F^n(\bot) \leq_{\hookrightarrow} \gamma_{\hookrightarrow}(F^n_A(\bot))$. Avremo

$$\begin{aligned}
F^{n+1}(\bot) &= \dot{cond}(\mathcal{B}[\![b]\!], F^n(\bot) \circ \mathcal{S}[\![S]\!], id) \\
\text{per IH, mon. di } \circ \text{e } \dot{cond} \text{ per f. 3.30} \quad &\leq_{\hookrightarrow} \dot{cond}(\mathcal{B}[\![b]\!], \gamma_{\hookrightarrow}(F^n_A(\bot)) \circ \mathcal{S}[\![S]\!], \gamma_{\hookrightarrow}(id^A)) \\
\text{per IH (su } \mathcal{S} \text{e } \mathcal{S}^A\text{), monot. di } \circ \text{e } \dot{cond} \quad &\leq_{\hookrightarrow} \dot{cond}(\mathcal{B}[\![b]\!], \gamma_{\hookrightarrow}(F^n_A(\bot)) \circ \gamma_{\hookrightarrow}(\mathcal{S}^A[\![S]\!]), \gamma_{\hookrightarrow}(id^A)) \\
\text{per correttezza locale di } \circ \text{e } \gamma_{\hookrightarrow} \text{ (??)} \quad &\leq_{\hookrightarrow} \dot{cond}(\mathcal{B}[\![b]\!], \gamma_{\hookrightarrow}(F^n_A(\bot) \circ \mathcal{S}^A[\![S]\!]), \gamma_{\hookrightarrow}(id^A)) \\
\text{per corr. di } \mathcal{B} \text{e } \mathcal{B}^A \text{e monot. di } \dot{cond} \quad &\leq_{\hookrightarrow} \dot{cond}(\gamma^{\mathbb{T}}_{\rightarrow}(\mathcal{B}^A[\![b]\!]), \gamma_{\hookrightarrow}(F^n_A(\bot) \circ \mathcal{S}^A[\![S]\!]), \gamma_{\hookrightarrow}(id^A)) \\
\text{per correttezza locale di } cond^A \quad &\leq_{\hookrightarrow} \gamma_{\hookrightarrow}(cond^A(\alpha^{\mathbb{T}}_{\rightarrow}(\gamma^{\mathbb{T}}_{\rightarrow}(\mathcal{B}^A[\![b]\!])), \alpha_{\hookrightarrow}(\gamma_{\hookrightarrow}(F^n_A(\bot) \circ \mathcal{S}^A[\![S]\!])), \alpha_{\hookrightarrow}(\gamma_{\hookrightarrow}(id^A)))) \\
\text{per monot. di } cond^A\text{,di } \gamma_{\hookrightarrow} \text{e per GC} \quad &\leq_{\hookrightarrow} \gamma_{\hookrightarrow}(cond^A(\mathcal{B}^A[\![b]\!], F^n_A(\bot) \circ \mathcal{S}^A[\![S]\!], id^A)) \\
\text{per definizione di } F_A \quad &= \gamma_{\hookrightarrow}(F_A(F^n_A(\bot))) \\
&= \gamma_{\hookrightarrow}(F^{n+1}_A(\bot))
\end{aligned}$$

□

Fatto 3.30. *Si ha che* $id \leq_{\hookrightarrow} \gamma_{\hookrightarrow}(id^A)$.

Osservazione 3.31. Da quanto appena dimostrato sappiamo che

$$\forall n \quad F^n(\bot) \quad \leq_{\hookrightarrow} \quad \gamma_{\hookrightarrow}(F^n_A(\bot))$$

dunque, per il lemma 3.28 sappiamo anche che

$$\bigsqcup_{n\in\mathbb{N}} F^n(\bot) \quad \leq_{\hookrightarrow} \bigsqcup_{n\in\mathbb{N}} \gamma_{\hookrightarrow}(F^n_A(\bot))$$

mentre per *monotonia di* $\gamma_{\rightarrow}$ *e per il Lemma 4.30 pag 102 da* [1]

$$\bigsqcup_{n\in\mathbb{N}} \gamma_{\hookrightarrow}(F^n_A(\bot)) \quad \leq_{\hookrightarrow} \gamma_{\hookrightarrow}(\bigsqcup_{n\in\mathbb{N}} F^n_A(\bot))$$

provando così la correttezza voluta.

CAPITOLO 4

Java™ Bytecode Verifier

Java è sicuramente uno dei linguaggi più diffusi, in particolare come "linguaggio di Internet". In particolare, il codice Java viene dapprima compilato da un *front end* che produce bytecode. Il bytecode viene poi interpretato da un *back end*, chiamato Java Virtual Machine (JVM). La particolarità di Java sta nel fatto che esistono implementazioni della JVM per ogni architettura. Ma cosa garantisce un livello di protezione del bytecode nella JVM? Due cose in particolare:

- una sandbox, cioè un ambiente limitato in cui il codice viene eseguito; il codice può comunque richiedere l'allocazione di più memoria;
- un verificatore, il *Java Bytecode Verifier*, che fa una verifica statica sul bytecode prima di decidere se metterlo o no in esecuzione

4.1. Il linguaggio JVML

Vediamo ora come è definito il linguaggio JVML.

Un **programma P** è una *funzione parziale* da insiemi di indirizzi $Addr$ a istruzioni $Instr$. Avremo:

$$Instr ::= \texttt{inc} \mid \texttt{pop} \mid \texttt{push0} \mid \texttt{load}\ x \mid \texttt{store}\ x \mid \texttt{if}\ L \mid \texttt{return} \mid$$
$$\texttt{getfield}\ \sigma.a\ \tau \mid \texttt{putfield}\ \sigma.a\ \tau$$

dove `x` è una variabile in Var (e le variabili saranno interi positivi $0, 1, 2...$), `L` è un indirizzo in $Addr$, σ e τ sono tipi in $Types$

4.1.1. I tipi. L'insieme $Types$ dei tipi contiene i tipi `Object`, `int` e i "tipi oggetto", cioè le classi. Come convenzione, useremo il simbolo τ per indicare un qualsiasi tipo, mentre il simbolo σ verrà usato per indicare i tipi degli oggetti.

Per semplicità considereremo classi con uno solo metodo, che avrà tipo

$$\tau_1 \times \tau_2 \times ... \times \tau_n \to \texttt{void}$$

Assumiamo inoltre di avere un insieme infinito di nomi di oggetti (puntatori) per ogni tipo σ.

Sui tipi di $Types$ è definita una relazione di sottotipo $<:$, che è riflessiva, antisimmetrica e transitiva, e in particolare contiene

$$\forall \tau\ \ \tau <: \texttt{Object}$$

cioè `Object` è la classe iniziale (il super-tipo di ogni tipo).

4.1.2. Semantica operazionale. La JVM utilizza una pila s e un insieme di variabili locali f, oltre ad un registro pc in cui è memorizzato l'indirizzo della prossima istruzione da eseguire. Una generica configurazione è

$$h : (pc,\, s,\, f)$$

dove

- h è lo heap, cioè il lugo in cui sono memorizzati gli oggetti; ad esempio, associo all'oggetto x i suoi campi con i rispettivi valori:

  $$x \mapsto [a_1 : v_1 ... a_k : v_k]$$

  Formalmente, h è una funzione da nomi di oggetti (indirizzi) in record values.
  - i record values hanno la forma di associazione nomi di campi-valori: $[a_1 : v_1 ... a_k : v_k]$
  - i valori v_i possono essere interi o a loro volta essere record values
  - h_0 è lo heap vuoto
- pc è un indirizzo di memoria (il program counter)
- s è uno stack (pila) di valori (interi oppure indirizzi di oggetti),
  - noi lo indicheremo con una stringa in cui l'elemento più a sinistra rappresenta la testa dello stack (che ovviamente funzionerà LIFO)
  - lo stack vuoto verrà indicato con ε
  - la concatenazione di elementi a e b sarà indicata con $a{\cdot}b$
- f è una funzione parziale da Var in valori.

La **configurazione iniziale** di un programma (= un metodo di una classe, nel nostro caso) P sarà

$$h' : (0,\, \varepsilon,\, f')$$

dove

- h' è uno heap che contiene almeno tutti gli oggetti memorizzati nelle variabili di f' (e in particolare il $this$, che avrà tipo σ)
- f' sarà una funzione che mappa nella variabile 0 il $this$, nella variabile 1 il primo argomento del metodo... nella variabile n, l'n-esimo argomento. Formalmente

  $$f' = f_0[0 : v_0,\, 1 : v_1 ... n : v_n] \;\; t.c. \;\; v_0 : \sigma, v_1 : \tau_1 ... v_n : \tau_n$$

- la forma delle transirizioni sarà

  $$h_1 : (pc_1,\, s_1,\, f_1) \to h_2 : (pc_2,\, s_2,\, f_2)$$

  mentre il programma è lasciato implicito

Il **significato** da dare alle istruzioni è il seguente:

- `inc`, `pop`, `push0`, `load` x eseguono le ovvie istruzioni sulla pila
- `store` x memorizza l'elemento il testa alla pila nella varaibile x e poi lo elimina dalla pila

- `if` L è l'istruzione di salto: se il valore sulla pila è **diverso** da 0 allora la prossima istruzione da eseguire è quella all'indirizzo L, altrimenti si procede con l'istruzione successiva in sequenza; al solito, il valore sulla pila è eliminato
- `putfield` $\sigma.a\ \tau$ e `getfield` $\sigma.a\ \tau$ inseriscono o prelevano il valore di un campo di un certo oggetto
- `return` termina l'esecuzione

Vediamo ora formalmente le transizioni che definiscono la semantica operazionale di JVML.

$$\frac{P[pc] = \texttt{inc}}{h : (pc,\ n{\cdot}s,\ f) \longrightarrow h : (pc+1,\ (n+1){\cdot}s,\ f)}$$

$$\frac{P[pc] = \texttt{push0}}{h : (pc,\ s,\ f) \longrightarrow h : (pc+1,\ 0{\cdot}s,\ f)}$$

Stiamo ovviamente facendo alcune assunzioni: la pila non è vuota, n è un intero, $pc+1$ esiste...

$$\frac{P[pc] = \texttt{pop}}{h : (pc,\ v{\cdot}s,\ f) \longrightarrow h : (pc+1,\ s,\ f)}$$

$$\frac{P[pc] = \texttt{load}\ x}{h : (pc,\ s,\ f) \longrightarrow h : (pc+1,\ f(x){\cdot}s,\ f)}$$

$$\frac{P[pc] = \texttt{store}\ x}{h : (pc,\ v{\cdot}s,\ f) \longrightarrow h : (pc+1,\ s,\ f[x : v])}$$

$$\frac{P[pc] = \texttt{if}\ L}{h : (pc,\ 0{\cdot}s,\ f) \longrightarrow h : (pc+1,\ s,\ f)}$$

$$\frac{P[pc] = \texttt{if}\ L \qquad n \neq 0}{h : (pc,\ n{\cdot}s,\ f) \longrightarrow h : (L,\ s,\ f)}$$

$$\frac{P[pc] = \texttt{putfield}\ \sigma.a\ \tau \qquad h' = h[o.a : v]}{h : (pc,\ v{\cdot}o{\cdot}s,\ f) \longrightarrow h' : (pc+1,\ s,\ f)}$$

in cui, per quanto riguarda i tipi, $v : \tau$ e $o : \sigma$

$$\frac{P[pc] = \texttt{getfield}\ \sigma.a\ \tau \qquad h(o.a) = v}{h : (pc,\ o{\cdot}s,\ f) \longrightarrow h : (pc+1,\ v{\cdot}s,\ f)}$$

Esempio 4.1. Vediamo alcuni esempi di programmi.

Il programma

```
0 : push0
1 : inc
2 : if 0
3 : return
```

è un programma lecito che non termina mai.

Il programma

```
0 : push0
1 : push0
2 : inc
3 : if 0
4 : return
```

viene rifiutato perché lo stack cresce all'infinito.

Il programma

```
0 : push0
1 : inc
2 : load 0
3 : if 0
4 : return
```

viene rifiutato perché all'istruzione 3 sulla pila c'è `this` (memorizzato in `0`) e dunque, non essendo un intero, non posso fare il salto condizionale.

Il programma

```
0 : push0
1 : inc
2 : store 0
3 : load 0
4 : if 0
5 : return
```

è un programma lecito ma non termina mai.

4.2. Semantica concreta

La semantica concreta di un programma scritto in Bytecode Java è un **sistema di transizione** (TS).

Definizione 4.2. Un *sistema di transizione* di Bytecode Java è una tripla $(C, \longrightarrow, c_0)$ dove

- C è un insieme di configurazioni $h : (pc, s, f)$ in cui
  - h è una funzione da oggetti a record (sequenze di campi con valori)
  - $pc \in 0..n$ per qualche n
  - s è una sequenza di valori (pila)
  - f è una funzione parziale da Var (che si assume essere un insieme *finito* di variabili) a valori
- la *relazione di transizione* $\longrightarrow$ è una *relazione tra configurazioni* (è quella vista in precedenza nella semantica operazionale)
- c_0 è la configurazione iniziale

Esempio 4.3. I sistemi di transizione possono essere finiti o infiniti. Il codice:

```
0 : push0
1 : inc
2 : store 1
3 : load 1
4 : load 1
5 : if 1
6 : return
```

genera un sistema un TS infinito (infinite state), che può essere rappresentato nel modo seguente:

TODO

mentre il codice

```
0 : push0
1 : inc
2 : if 0
3 : return
```

genera un TS finito (finite state) che è il seguente:

TODO

Chiamiamo $\mathscr{C}$ l'*insieme dei sistemi di transizione di Bytecode Java*, allora possiamo definire la semantica come la funzione che ha tipo

$$\mathcal{S} : Programs \to ((Heaps \times Functions) \to \mathscr{C})$$

in cui $Programs$ è l'insieme dei programmi scrtti in bytecode, $Heaps$ contiene gli heap che contengono gli oggetti con relativi campi e valori, $Functions$ sono le funzioni in cui almeno la variabile 0 è associata al `this` e le altre variabili con gli argomenti del metodo.

Dunque definiamo

$$\mathcal{S}[\![P]\!](h, f)$$

come il **sistema di transizione** del bytecode P con heap iniziale h e funzione iniziale f (la pila è sempre ε).

Osservazione 4.4. Poiché Java è ben tipato, h ed f non possono essere un qualunque elemento di $Heaps$ e di $Functions$, come già visto in precedenza.

4.3. La semantica astratta

La semantica astratta del Bytecode Java che ci accingiamo a studiare intende verificare le seguenti proprietà (che sono verificate dal Java Bytecode Verifier [8]):

(1) Il bytecode usa una quantità finita di pila
(2) I salti avvengono ad indirizzi legali del bytecode
(3) Le operazioni sono applicate ad argomenti di tipo opportuno

Esempio 4.5. Vediamo due esempi di programmi errati.

Il programma

```
0 : push0
1 : push0
2 : inc
3 : if 0
4 : return
```

viene rifiutato perché lo stack cresce all'infinito.

Il programma

```
0 : load 0
1 : inc
2 : return
```

viene rifiutato perché sto cercando di incrementare l'oggetto this.

Osservazione 4.6. Alcune osservazioni.

- Il punto 2. è necessario per motivi di sicurezza: il bytecode può essere modificato a mano e trasmesso.

- Un programma che non usa `if` o usa solo `if` (salti) in avanti usa un numero finito di risorse. Il problema sono i salti all'indietro. Per risolver questo problema, dobbiamo garantire che lo stato della pila nel corpo del while sia *invariante*.

- Se avessi solo i punti 1. e 2. mi basterebbe un intero (che conta il numero di elementi) per verificare la pila. Ma ho il punto 3. e dunque ho a che fare con i tipi. Ad sempio devo verificare che valgano i seguenti tipi (intuitivi):

$$\begin{aligned} \texttt{push0} &: Pila \to int{\cdot}Pila \\ \texttt{inc} &: int{\cdot}Pila \to int{\cdot}Pila \\ \texttt{load 0} &: Pila \to Class.Pila \end{aligned}$$

Per definire la semantica astratta di un programma P con n istruzioni si utilizzerà una tabella $\mathcal{T}_P$ con $n+1$ righe. Ciascuna riga contiene una coppia (S, F) in cui S è una **pila astratta**, cioè una sequenza di tipi Java ($Stacks_P$), mentre F è una **funzione astratta** da Var a $Types_P$, cioè tipi Java presenti nel programma P.

Definizione 4.7. Sia $\mathscr{T}$ l'insieme delle tabelle della tipologia appena definita, allora la **semantica astratta di bytecode Java** è la funzione che ha tipo

$$\mathcal{S}^A : Programs \to \mathscr{T}$$

Osservazione 4.8. Osserviamo che

a. Non ci sono *heap* e $function$, poichè sono costanti e implicite nel codice

b. Nella tabella $\mathcal{S}^A[\![P]\!]$ ci sono solo i tipi di P (cioè elementi T_i di $Types_P$)

Definizione 4.9. Una sequenza di tipi S è più piccola di una sequenza S' ($S \leq S'$) se

- Le due sequenze hanno la stessa lunghezza ($S = T_1{\cdot}...{\cdot}T_k$ e $S' = T'_1{\cdot}...{\cdot}T'_k$) e, per ogni i, $T_i <: T'_i$
- oppure se $S' = \top$ (cattura errori di tipo)
- oppure se $S = \bot$ (per istruzioni irraggiungibili)

Proposizione 4.10. *Sia $Stacks_P$ l'insieme di tutte le pile (sequenze di tipi) di un programma P con l'aggiunta di $\bot$ e $\top$. Allora $(Stacks_P, \leq)$ è un reticolo completo.*

DIMOSTRAZIONE. Siano $\lfloor\cdot\rfloor$ e $\lceil\cdot\rceil$ rispettivamente il LUB e il GLB sull'insieme dei Tipi Java rispetto all' ordinamento <:. Dapprima si definiscono

$$\bigsqcup_{i\in I} S_i \stackrel{def}{=} \begin{cases} \top & \text{se esistono } S_j,S_k \text{di lunghezza differente} \\ \lfloor\cdot\rfloor_{i\in I} S_{i_1}\cdot\lfloor\cdot\rfloor_{i\in I} S_{i_2}\cdot\ldots\cdot\lfloor\cdot\rfloor_{i\in I} S_{i_n} & \text{se ogni } S_i \text{ha lunghezza } n \text{e } S_i = S_{i_1}\cdot S_{i_2}\cdot\ldots\cdot S_{i_n} \end{cases}$$

$$\bigsqcap_{i\in I} S_i \stackrel{def}{=} \begin{cases} \bot & \text{se esistono } S_j,S_k \text{di lunghezza differente oppure se} \\ & S_j,S_k \text{stessa lung. ma esistono elem. senza sottotipo comune} \\ \lceil\cdot\rceil_{i\in I} S_{i_1}\cdot\lceil\cdot\rceil_{i\in I} S_{i_2}\cdot\ldots\cdot\lceil\cdot\rceil_{i\in I} S_{i_n} & \text{altrimenti} \end{cases}$$

La dimostrazione che $\bigsqcup_{i\in I} S_i$ e $\bigsqcap_{i\in I} S_i$ sono definizioni corrette e che sono effettivamente il LUB e il GLB è lasciata come esercizio. □

Definizione 4.11. Una funzione $F : Var \to Types_P$ è più piccola di una funzione F' (e, con overloading di notazione scriveremo $F \leq F'$) se si ha che

$$dom(F) \subseteq dom(F') \; e \; \forall x \; F(x) <: F'(x)$$

Proposizione 4.12. *Sia $Functions_P$ l'insieme delle funzioni di un programma P. Allora $(Functions_P, \leq)$ è un reticolo completo. La funzione più piccola è $\varnothing$, mentre la funzione più grande è definita come $\top = \forall x\ [x \mapsto \texttt{Object}]$*

DIMOSTRAZIONE. Standard, utilizzando le proprietà di <: sui tipi di Java. In particolare,

$$\begin{aligned} F \sqcup F' &= \text{per ogni } x \text{ritorna il tipo più grande} \\ F \sqcap F' : &= \text{per ogni } x \text{ritorna il più grande sottotipo comune, se esiste} \end{aligned}$$

Formalmente

$$\begin{aligned} F \sqcup F' &= ? \\ F \sqcap F' : &= \begin{cases} x \mapsto F(x) \sqcap F'(x) & se\, x \in dom(F) \cap dom(F') ed\, esiste\, F(x) \sqcap F'(x) \\ \mathsf{undef} & altrimenti \end{cases} \end{aligned}$$

□

Siamo ora in grado di dare la definizione di $\mathcal{S}^A[\![P]\!]$.

La tabella $\mathcal{S}^A[\![P]\!]$ è definita attraverso un'operazione di punto fisso (che termina in tempo finito). A tal fine definiamo il seguente ordinamento in $\mathscr{T}_P$, cioè l'insieme di tabelle relative a P. Avremo che

$$\mathcal{T}_P \leq \mathcal{T}'_P \stackrel{def}{=} \forall i \;\; \mathcal{T}_P[i].stack \leq \mathcal{T}'_P[i].stack \;\wedge\; \mathcal{T}_P[i].fun \leq \mathcal{T}'_P[i].fun$$

dove $.stack$ e $.fun$ restituiscono il primo e il secondo elemento di una coppia (S, F).

Teorema 4.13. *Si dimostra che* $(\mathscr{T}_P, \leq)$ *è un reticolo completo. In particolare,*

$$\begin{aligned} \mathcal{T}_P^{\perp} &= i \mapsto (\perp, \varnothing) \\ \mathcal{T}_P^{\top} &= i \mapsto (\top, \top) \end{aligned}$$

Con la notazione $\mathcal{T}[i]$ indichiamo l'accesso all'i-esima riga della tabella, ottenendo un elemento della forma (S, F). Per un programma di n istruzioni, otteniamo una tabella di $n+1$ righe. In ogni riga otteniamo lo stato in cui mi trovo *prima* di eseguire quell'istruzione.

Sia $I_P : \mathscr{T}_P \to \mathscr{T}_P$ definito come segue (con $i \in Addr_P$):

$$P[i] = \texttt{inc} \Rightarrow \begin{cases} I_P(\mathcal{T})[i+1] = (\texttt{int}\cdot S, F) \sqcup \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\texttt{int}\cdot S, F)\ e\ i+1 \in Addr_P \\ I_P(\mathcal{T})[i+1] = \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\bot, F)\ e\ i+1 \in Addr_P \\ I_P(\mathcal{T})[i+1] = (\top, \top) & altrimenti \end{cases}$$

$$P[i] = \texttt{push0} \Rightarrow \begin{cases} I_P(\mathcal{T})[i+1] = (\texttt{int}\cdot S, F) \sqcup \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (S, F)\ e\ i+1 \in Addr_P \\ I_P(\mathcal{T})[i+1] = \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\bot, F)\ e\ i+1 \in Addr_P \\ I_P(\mathcal{T})[i+1] = (\top, \top) & altrimenti \end{cases}$$

$$P[i] = \texttt{pop} \Rightarrow \begin{cases} I_P(\mathcal{T})[i+1] = (S, F) \sqcup \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\tau\cdot S, F)\ e\ i+1 \in Addr_P \\ I_P(\mathcal{T})[i+1] = \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\bot, F)\ e\ i+1 \in Addr_P \\ I_P(\mathcal{T})[i+1] = (\top, \top) & altrimenti \end{cases}$$

$$P[i] = \texttt{load}\ x \Rightarrow \begin{cases} I_P(\mathcal{T})[i+1] = (F(x)\cdot S, F) \sqcup \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (S, F)\ e\ i+1 \in Addr_P\ e\ x \in d(F) \\ I_P(\mathcal{T})[i+1] = \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\bot, F)\ e\ i+1 \in Addr_P\ e\ x \in d(F) \\ I_P(\mathcal{T})[i+1] = (\top, \top) & altrimenti \end{cases}$$

$$P[i] = \texttt{store}\ x \Rightarrow \begin{cases} I_P(\mathcal{T})[i+1] = (S, F[x \mapsto \tau]) \sqcup \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\tau\cdot S, F)\ e\ i+1 \in Addr_P \\ I_P(\mathcal{T})[i+1] = \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\bot, F)\ e\ i+1 \in Addr_P \\ I_P(\mathcal{T})[i+1] = (\top, \top) & altrimenti \end{cases}$$

$$P[i] = \texttt{if}\ L \Rightarrow \begin{cases} I_P(\mathcal{T})[i+1] = (S, F) \sqcup \mathcal{T}[i+1] \sqcup \mathcal{T}[L] & se\ \mathcal{T}[i] = (\texttt{int}\cdot S, F)\ e\ i+1,\ L \in Addr_P \\ I_P(\mathcal{T})[L] = (S, F) \sqcup \mathcal{T}[i+1] \sqcup \mathcal{T}[L] & \text{(stessa continuazione per i+1 ed L)} \\ I_P(\mathcal{T})[i+1] = \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\bot, F)\ e\ i+1,\ L \in Addr_P \\ I_P(\mathcal{T})[i+1] = (\top, \top) & altrimenti \\ I_P(\mathcal{T})[L] = (\top, \top) & \end{cases}$$

$$P[i] = \texttt{putfield}\ \sigma.a\ \tau \Rightarrow \begin{cases} I_P(\mathcal{T})[i+1] = (S, F) \sqcup \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\tau\cdot\sigma\cdot S, F)\ e\ i+1 \in Addr_P \\ I_P(\mathcal{T})[i+1] = \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\bot, F)\ e\ i+1 \in Addr_P \\ I_P(\mathcal{T})[i+1] = (\top, \top) & altrimenti \end{cases}$$

$$P[i] = \texttt{getfield}\ \sigma.a\ \tau \Rightarrow \begin{cases} I_P(\mathcal{T})[i+1] = (\tau\cdot S, F) \sqcup \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\sigma\cdot S, F)\ e\ i+1 \in Addr_P \\ I_P(\mathcal{T})[i+1] = \mathcal{T}[i+1] & se\ \mathcal{T}[i] = (\bot, F)\ e\ i+1 \in Addr_P \\ I_P(\mathcal{T})[i+1] = (\top, \top) & altrimenti \end{cases}$$

$$P[i] = \texttt{return} \Rightarrow \text{non c'è semantica per il return}$$

Notiamo che nelle `putfield .` e `getfield .`, nonostante esse usino lo heap, tutte le informazioni che ci servono sono codificate nella sintassi, poiché i tipi sono esplicitati. Dunque lo heap non serve per la semantica astratta, che usa solo le informazioni sui tipi.

Esempio 4.14. Vediamo alcuni esempi di uso di questo algoritmo TODO.

Lemma 4.15. *La funzione $I_P : \mathscr{T}_P \to \mathscr{T}_P$ è monotona e continua.*

Dimostrazione. Per la monotonia, supponiamo che $\mathcal{T} \leq \mathcal{T}'$ e dobbiamo dimostrare che $I_P(\mathcal{T}) \leq I_P(\mathcal{T}')$. Procediamo per casi.

Caso $P[i] = \texttt{inc}$. Dobbiamo verificare che $I_P(\mathcal{T})[i+1] \leq I_P(\mathcal{T}')[i+1]$.
Se $\mathcal{T}[i] = (\texttt{int}\cdot S, F)$ allora $\mathcal{T}'[i] = (\texttt{int}\cdot S', F')$, con $S \leq S'$ e $F \leq F'$ oppure $\mathcal{T}'[i] = (\texttt{Object}\cdot S', F')$ con $S \leq S'$ e $F \leq F'$ oppure $\mathcal{T}'[i] = (\top, F')$ con $F \leq F'$. In tutti i casi, $I_P(\mathcal{T})[i+1] \leq I_P(\mathcal{T}')[i+1]$ (infatti avremo come risultato $(\texttt{int}\cdot S', F') \sqcup \mathcal{T}'[i+1]$ che è $\geq$ per ipotesi, oppure $(\top, \top)$).
Se $\mathcal{T}[i] = (\bot, F)$ allora $I_P(\mathcal{T})[i+1] = \mathcal{T}[i+1] \leq \mathcal{T}'[i+1] \leq I_P(\mathcal{T}')[i+1]$ (per ipotesi e poi per come è definita I_p).
Altrimenti, $I_P(\mathcal{T})[i+1] = (\top, \top)$ ma, in questo caso, poiché $\mathcal{T} \leq \mathcal{T}'$ allora anche $I_P(\mathcal{T}')[i+1] = (\top, \top)$

Caso... gli altri casi sono analoghi.

Per la continuità, sia $\mathcal{T}_0 \leq \mathcal{T}_1 \leq \mathcal{T}_2 \leq \ldots$ una catena di tabelle di P, dobbiamo dimostrare che $I_P(\bigsqcup_{j\in\mathbb{N}} \mathcal{T}_j) = \bigsqcup_{j\in\mathbb{N}} I_P(\mathcal{T}_j)$. A tal fine proviamo che

$$\begin{aligned} I_P\Big(\bigsqcup_{j\in\mathbb{N}} \mathcal{T}_j\Big)[i+1] &= \Big(\bigsqcup_{j\in\mathbb{N}} I_P(\mathcal{T}_j)\Big)[i+1] \\ \text{poiché è un LUB riga per riga} &= \Big(\bigsqcup_{j\in\mathbb{N}} I_P(\mathcal{T}_j)[i+1]\Big) \\ &\quad \text{su ogni tabella} \end{aligned}$$

Procediamo per casi.

Caso $P[i] = \texttt{inc}$. Ci sono diversi sottocasi.
- $(\bigsqcup_{j\in\mathbb{N}} \mathcal{T}_j)[i] = (\texttt{int}\cdot S, F)$ e $i+1 \in Adrr_P$. Allora $I_P(\bigsqcup_{j\in\mathbb{N}} \mathcal{T}_j)[i+1] = (\texttt{int}\cdot S, F) \sqcup (\bigsqcup_{j\in\mathbb{N}} \mathcal{T}_j)[i+1]$.
Se $(\bigsqcup_{j\in\mathbb{N}} \mathcal{T}_j)[i] = (\texttt{int}\cdot S, F)$ significa che esiste un k tale che $\forall j' \geq k$ vale $\mathcal{T}_{j'}[i] = (\texttt{int}\cdot S_{j'}, F_{j'})$ [Questa proprietà deriva dal sistema di tipi di Java. Infatti i $\mathcal{T}_j$ sono una catena crescente e, poiché il sistema di tipi è un *ordine parziale finito* allora inizierà con $(\bot, \varnothing)$ ma prima o poi dovrò per forza incontrare un $(\texttt{int}\cdot S, F)$].

Dunque

$$\begin{aligned}
\bigsqcup_{j' \geq k} I_P(\mathcal{T}_{j'})[i+1] &= \bigsqcup_{j' \geq k} (\texttt{int}\cdot S_{j'}, F_{j'}) \sqcup \mathcal{T}_{j'}[i+1] \\
\text{visto che i } j' < k \text{sono cmq } \bot \text{non influiscono sul LUB} &= \bigsqcup_{j \in \mathbb{N}} (\texttt{int}\cdot S_j, F_j) \sqcup \bigsqcup_{j \in \mathbb{N}} \mathcal{T}_j[i+1] \\
&= (\texttt{int}\cdot S, F) \sqcup (\bigsqcup_{j \in \mathbb{N}} \mathcal{T}_j)[i+1] \\
\text{per definizione} &= I_P(\bigsqcup_{j \in \mathbb{N}} \mathcal{T}_j)[i+1]
\end{aligned}$$

- $(\bigsqcup_{j\in\mathbb{N}} \mathcal{T}_j)[i] = (\bot, F)$, simile al precedente
- $(\bigsqcup_{j\in\mathbb{N}} \mathcal{T}_j)[i] \neq (\texttt{int}\cdot S, F), (\bot, F)$, simile al precedente

Caso... gli altri casi sono analoghi. □

Per il Teorema di Kleene (Thm. 4.37, pag. 104 da [1]), la funzione I_P ha minimo punto fisso $\bigsqcup_{n\in\mathbb{N}} I_P^n(\mathcal{T}_0)$ in cui

$$\mathcal{T}_0 \ : \ \begin{cases} 0 \mapsto (\varepsilon,\, F_0) \\ i+1 \mapsto (\bot, \bot) \end{cases}$$

ed

$$F_0 \ : \ \begin{cases} 0 \mapsto \text{il tipo di this} \\ 1 \mapsto \text{il tipo del I arg.} \\ \vdots \\ n \mapsto \text{il tipo dell'n. arg.} \end{cases}$$

Proposizione 4.16. *Sia $FIX(I_P)$ il suddetto punto fisso, allora*

$$\mathcal{S}^A[\![P]\!] = FIX(I_P)$$

ed un programma in Bytecode Java è corretto se il tipo $\top$ non compare mai nella tabella $FIX(I_P)$.

Osservazione 4.17. Il Teorema di Kleene ci imporrebbe di partire dalla tabella minima $\mathcal{T} = i \mapsto (\bot, \bot)$, ma in questo caso la funzione I_P si comporta come l'identità. Se vogliamo partire dalla tabella $\mathcal{T}_0$ dobiamo assicurarci che da essa parta una catena "non discendente". Questo è assicurato dal seguente fatto.

Fatto 4.18. *Per ogni $\mathcal{T}$, $\mathcal{T} \leq I_P(\mathcal{T})$*

Dimostrazione. Evidente dalla definizione di I_P □

Esempio 4.19. Vediamo un esempio completo. Sia C una classe con un campo a che contiene interi ed un metodo che modifica a come segue: se il valore in a è 0 allora ci memorizza 1, altrimenti lo lascia inalterato. In Java

```
public class C {
    int a;
    void one(){
        if (this.a == 0){
            this.a = 1;
        }
        return;
    }
}
```

Il corrispondente bytecode potrebbe essere il seguente:

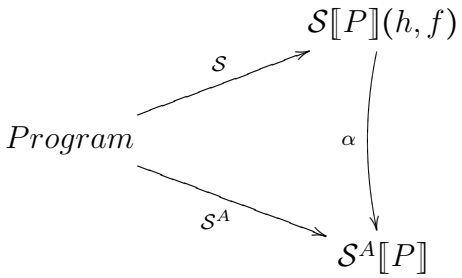
```
 0 : load 0
 1 : getfield C.a int
 2 : store 1
 3 : load 1
 4 : if 11
 5 : push0
 6 : inc
 7 : store 1
 8 : load 0
 9 : load 1
10 : putfield C.a int
11 : return
```

Calcoliamo ora I_P (con il "turbo punto fisso")

Istr.	Ite. 0	Ite. 1	Ite. 2
0	$(\varepsilon, [0 \mapsto C])$	$(\varepsilon, [0 \mapsto C])$	idem
1	$(\bot, \varnothing)$	$(C, [0 \mapsto C])$	"
2	$(\bot, \varnothing)$	$(\texttt{int}, [0 \mapsto C])$	"
3	$(\bot, \varnothing)$	$(\varepsilon, [0 \mapsto C, 1 \mapsto \texttt{int}])$	"
4	$(\bot, \varnothing)$	$(\texttt{int}, [0 \mapsto C, 1 \mapsto \texttt{int}])$	"
5	$(\bot, \varnothing)$	$(\varepsilon, [0 \mapsto C, 1 \mapsto \texttt{int}])$	"
6	$(\bot, \varnothing)$	$(\texttt{int}, [0 \mapsto C, 1 \mapsto \texttt{int}])$	"
7	$(\bot, \varnothing)$	$(\texttt{int}, [0 \mapsto C, 1 \mapsto \texttt{int}])$	"
8	$(\bot, \varnothing)$	$(\varepsilon, [0 \mapsto C, 1 \mapsto \texttt{int}])$	"
9	$(\bot, \varnothing)$	$(C, [0 \mapsto C, 1 \mapsto \texttt{int}])$	"
10	$(\bot, \varnothing)$	$(\texttt{int}\cdot C, [0 \mapsto C, 1 \mapsto \texttt{int}])$	"
11	$(\bot, \varnothing)$	$(\varepsilon, [0 \mapsto C, 1 \mapsto \texttt{int}])$	"
	$(\bot, \varnothing)$	$(\bot, \varnothing)$	"

4.4. La correttezza del Java Bytecode Verifier

Vogliamo definire una funzione di astrazione α, che valga $\forall h, f$ come descritto nel seguente diagramma.



Dobbiamo cioè mappare infinite configurazioni (ricorda: la semantica concreta è un Transition System potenzialmente infinito) in un numero finito di righe della tabella. In particolare, considererò $n+1$ righe per un programma di n righe, e "accorperò" (calcolandone il LUB) tutte le configurazioni con lo stesso *program counter*.

Definizione 4.20. Sia α la funzione di astrazione di tipo $\alpha : \mathscr{C} \to \mathscr{T}$ definita come segue

$$\alpha(C, \to, c_0)[i] = \begin{cases} (\bigsqcup\{S \mid h : (i, s, f) \in \mathscr{C} \wedge s : S\}, \bigsqcup\{F \mid h : (i, s, f) \in \mathscr{C} \wedge f : F\}) & \text{se } i \in Addr_P \\ (\bot, \varnothing) & \text{se } i = n+1 \end{cases}$$

dove P è un programma di n istruzioni e $s : S$, $f : F$ sono i tpi Java, cioè

- $s : S$ significa che $s = s_1 \cdot \ldots \cdot s_k$ e $S = S_1 \cdot \ldots \cdot S_k$ e vale $s_i : S_i$ dove S_i è **il più piccolo tipo** per cui $s_i : S_i$.
- $f : F$ significa che $dom(f) = dom(F)$ e $\forall x \;\; f(x) : F(x)$

Teorema 4.21. *(TODO: confuzione con F,F') Per ogni h, f ben tipati (rispetto a P)*

$$\alpha(\mathcal{S}[\![P]\!](h,f)) \leq \mathcal{S}^A[\![P]\!]$$

Dimostrazione. Osserviamo che, poiché h ed f sono ben tipati, lo heap e la funzione astratta sono *costanti*, e dunque non li consideriamo.

Data una configuraizone $h : (pc,\ s,\ f)$ diciamo che (S, F) è un suo tipo (notazione: $h : (pc,\ s,\ f) : (S, F)$) se S ed F sono *la più piccola pila astratta e la più piccola funzione astratta* che soddisfano la relazione.

Dimostriamo che se $h : (pc,\ s,\ f) \longrightarrow h' : (pc',\ s',\ f')$ (con $h : (pc,\ s,\ f) : (S, F)$ e $h' : (pc',\ s',\ f') : (S', F')$ e $(S, F) \leq \mathcal{S}^A[\![P]\!][pc]$ allora $(S', F') \leq \mathcal{S}^A[\![P]\!][pc']$. Ragioniamo per casi sull'istruzione $P[pc]$.

Caso $P[i]$ = `inc`. In questo caso quindi $pc' = pc + 1$. Dalla semantica operazionale, so che $h : (pc,\ s,\ f) : (\texttt{int}\cdot S', F')$ e che $pc + 1 \in Addr_P$. Per ipostei so che $(\texttt{int}\cdot S', F') \leq \mathcal{S}^A[\![P]\!][pc]$. Si presentano tre casi
- (a) $\mathcal{S}^A[\![P]\!][pc] = (\texttt{int}\cdot S'', F'')$, con $S' <: S''$ e $F' <: F''$
- (b) $\mathcal{S}^A[\![P]\!][pc] = (\texttt{Object}\cdot S'', F'')$, con $S' <: S''$ e $F' <: F''$
- (c) $\mathcal{S}^A[\![P]\!][pc] = (\top, F'')$, con $F' <: F''$

Sempre per la semantica operazioanle, $h' : (pc + 1,\ s',\ f') : (\texttt{int}\cdot S, F)$ ed è facile dimostrare che, nei 3 casi (a),(b) e (c) vale sempre $(\texttt{int}\cdot S, F) \leq \mathcal{S}^A[\![P]\!][pc + 1]$

Caso... gli altri casi sono analoghi.

Osserviamo ora che lo stato iniziale $h : (0,\ \varepsilon,\ f)$ di $\mathcal{S}[\![P]\!](h, f)$ ha tipo (ε, F_0) e che $(\varepsilon, F_0) \in \mathcal{S}^A[\![P]\!][0]$, poiché (ε, F_0) è il valore contenuto alla riga 0 della tabella iniziale di $FIX(I_P)$ e sappiamo che $\mathcal{T} \leq I_P(\mathcal{T})$

Quindi, poiché per ogni $h : (i,\ s,\ f) \in \mathcal{S}[\![P]\!](h', f')$, il suo tipo (S, F) soddisfa $(S, F) \leq \mathcal{S}^A[\![P]\!][i]$ e dunque il LUB di questi tipi (S, F) sarà a sua volta più piccolo di $\mathcal{S}^A[\![P]\!][i]$. □

Bibliografia

[1] Hanne Riis Nielson, Flemming Nielson: Semantics with Applications: A Formal Introduction. 0.3, 1.3, 3.3, 3.31, 4.3

[2] F. Nielson, H.R. Nielson, C. Hankin: "Principles of Program Analysis", Springer Verlag, 2005

[3] K. R. Apt: "Ten years of Hoare logic: A survey – Part 1", ACM Transactions on Programmming Languages, Vol. 3(4), pp. 431–483, 1981.

[4] N. D. Jones, F. Nielson: "Abstract interpretation, a semantic based tool for program analysis", 1994

[5] Glynn Winskel: Lecture notes on Denotational semantics.

[6] Dispense del docente. http://www.cs.unibo.it/~laneve/analisidiprogrammi.html. (document), 2.8

[7] B. A. Davey, H. A. Priestley: Introduction to Lattices and Order.

[8] Tim Lindholm, Frank Yellin: The Java™ Virtual Machine Specification, Second Edition. http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html. 4.3

[10] Évariste Galois. (27 gennaio 2011). Wikipedia, L'enciclopedia libera. Tratto il 19 marzo 2011, 13:08 da http://it.wikipedia.org/w/index.php?title=%C3%89variste_Galois&oldid=38098005. 2.3

[?] Altri riferimenti