Università degli Studi di Verona

DIPARTIMENTO DI INFORMATICA

Analisi di Sistemi informatici

Riassunto dei principali argomenti

Autore:
Davide Bianchi

Anno Accademico 2017/2018

Indice

1 Preliminari matematici

1.1 Ordini parziali

1.2 Reticoli

1.3 Teoremi di punto fisso

2 Interpretazione astratta

2.1 Introduzione

Lo scopo è quello di trovare un'approssimazione di una semantica $\langle P \rangle$ di $[\![P]\!]$ tale per cui valgano:

- *correttezza:* $[\![P]\!] \subseteq \langle P \rangle$;
- *decidibilità:* $\langle P \rangle \subseteq Q$ è decidibile (Q è un insieme di semantiche che soddisfa la proprietà di interesse).

Se entrambe le proprietà sono soddisfatte, allora vale che

$$(\langle P \rangle \subseteq Q) \Rightarrow ([\![P]\!] \subseteq Q)$$

La semantica è data da una coppia $\langle D, f \rangle$ dove D è una coppia $\langle D, \leq_D$ rappresentante un dominio semantico e $f : D \to D$ è una funzione di trasferimento con una soluzione a punto fisso.

Dato un oggetto concreto, definiamo:

- un **oggetto astratto** come una rappresentazione matematica sovra-approssimata del corrispondente concreto;
- un **dominio astratto** come un insieme di oggetti astratti con delle operazioni astratte, che approssimano quelle concrete;
- una funzione di **astrazione** α che mappa oggetti concreti in oggetti astratti;
- una funzione di **concretizzazione** γ che mappa oggetti astratti in oggetti concreti.

La caratteristica peculiare delle astrazioni è che solo alcune proprietà vengono osservate con esattezza, le altre vengono solo approssimate. In sostanza, dato un dominio astratto A, gli elementi di A sono osservati con esattezza, gli altri sono approssimati o l'informazione è persa del tutto.

Proprietà. L'insieme delle proprietà $\mathcal{P}(\Sigma)$ di oggetti in Σ è l'insieme di elementi che gode di quella proprietà. Questo insieme di proprietà costituisce un reticolo completo

$$\langle \mathcal{P}(\Sigma), \subseteq, \emptyset, \cup, \cap, \neg \rangle$$

dove:

- $\subseteq$ è l'implicazione logica;
- Σ è `true`;
- $\cup$ è la disgiunzione (oggetti che godono di P o di Q appartengono a $P \cup Q$);
- $\cap$ è la congiunzione (oggetti che godono di P e di Q appartengono a $P \cap Q$);
- $\neg$ è la negazione (oggetti che non godono di P stanno in $\Sigma \setminus P$).

Direzione dell'astrazione. Quando si approssima una proprietà concreta $P \in \mathcal{P}(\Sigma)$ usando una proprietà astratta $\overline{P}$, deve essere stabilito un criterio per definire quando $\overline{P}$ è un'approssimazione di P.

Si distinguono quindi i seguenti casi:

- approssimazione *da sopra*: $P \subseteq \overline{P}$;
- approssimazione *da sotto*: $P \supseteq \overline{P}$.

Dato un oggetto o, si vuole quindi sapere se $o \in P$:

$$P \supseteq \overline{P} : \begin{cases} \text{"Si"} & o \in \overline{P} \\ \text{"Non lo so"} & o \notin \overline{P} \end{cases} \qquad P \subseteq \overline{P} : \begin{cases} \text{"No"} & o \notin \overline{P} \\ \text{"Non lo so"} & o \in \overline{P} \end{cases}$$

Migliore approssimazione. Definiamo come *migliore approssimazione* di una proprietà P in A il glb delle over-approximation di P in A, ossia:

$$\overline{P} = \bigcap \{\overline{P'} \in A | P \subseteq \overline{P'}\} \in A$$

2.2 Connessione di Galois

Imponiamo il vincolo che α e γ siano monotone, allora concludiamo che:

- $\gamma \circ \alpha : C \to C$ è **estensiva**: $\gamma(\alpha(c)) \geq c$;
- $\alpha \circ \gamma : A \to A$ è **riduttiva**: $\alpha(\gamma(a)) \leq a$.

Le definizioni qui sopra dicono rispettivamente che:

- α perde informazione, e γ non la può recuperare;
- γ non perde informazione.

Definizione 2.2.1 (Connessione di Galois)**.** *Dati due poset $\langle A, \leq_A \rangle$ e $\langle C, \leq_C \rangle$, e due funzioni monotone $\alpha : C \to A$ e $\gamma : A \to C$, diciamo che $\langle C, \alpha, \gamma, A \rangle$ è una connessione di Galois se:*

- $\forall c \in \mathcal{C} : c \leq_C \gamma(\alpha(c))$
- $\forall a \in \mathcal{A} : \alpha(\gamma(a)) \leq_A a$

Se inoltre vale che $\forall a \in \mathcal{A} : \alpha(\gamma(a)) = a$, allora $\langle C, \alpha, \gamma, A \rangle$ è un'inserzione di Galois.

Una connessione e un'inserzione di Galois sono rappresentate rispettivamente come

$$C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A \qquad C \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\!\!\rightarrow}} A$$

La funzione α è detta *aggiunta sinistra*, mentre la funzione γ è detta *aggiunta destra.*

Teorema 2.2.1. *Data una connessione di Galois $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$, sono equivalenti:*

- $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\!\!\rightarrow}} A$*;*
- *α è suriettiva;*
- *γ è iniettiva.*

Inoltre, dati due domini astratti, non esistono due coppie (α, γ) che formino una connessione di Galois; quindi la connessione di Galois tra due domini è **unica**, e le funzioni sono identificabili attraverso:

$$\alpha(c) = \bigwedge \{a \in A | c \leq_C \gamma(a)\}$$
$$\gamma(a) = \bigvee \{c \in C | \alpha(c) \leq_A a\}$$

2.3 Famiglie di Moore

Definizione 2.3.1 (Famiglia di Moore)**.** *Sia L un reticolo completo. $X \subseteq L$ è una famiglia di Moore di L se*

$$X = \mathcal{M}(X) = \left\{ \bigwedge S \mid S \subseteq X \right\}$$

dove

$$\bigwedge \emptyset = \top \in \mathcal{M}(X)$$

Da questa definizione segue che, ipotizzando che ogni proprietà concreta abbia una migliore astrazione $\overline{P} \in A$, implica che il dominio A è una famiglia di Moore.

2.4 Upper closure operator

Definizione 2.4.1 (Upper closure operator)**.** *Una funzione $f : P \to P$ su un poset $\langle P, \leq_P \rangle$ è un upper closure operator (uco) se soddisfa le seguenti proprietà:*

- *estensività:* $\forall x \in P : x \leq_P \rho(x)$
- *monotonia:* $\forall x, y \in P : (x \leq_P y) \Rightarrow (\rho(x) \leq_P \rho(y)$
- *idempotenza:* $\forall x \in P : \rho(x) = \rho(\rho(x))$

I lower closure operator sono definiti in modo duale, specificando che ρ deve essere *riduttiva*, ovvero che $\forall x \in P : x \geq_P \rho(x)$.

Teorema 2.4.1. *Data una connessione di Galois $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ si ha che $\gamma \circ \alpha$ è un uco e $\alpha \circ \gamma$ è un lco.*

Teorema 2.4.2. *$C \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\!\rightarrow}} A$ se e solo se A è isomorfo [1] ad una Moore family di C.*

Teorema 2.4.3. *Sia $\rho \in uco(c)$. Allora $\forall A \simeq \rho(C)$ si ha che $\exists \alpha, \gamma : C \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\!\rightarrow}} A$*

2.5 Reticolo delle interpretazioni astratte

I vari domini astratti possono essere comparati sulla base della loro precisione. In generale si può dire che un dominio astratto A_1 è più preciso di A_2 (indicato attraverso $A_1 \sqsubseteq A_2$) quando

$$\forall a_2 \in A_2, \exists a_1 \in A_1 \quad \text{tali che} \quad \gamma_1(a_1) = \gamma_2(a_2)$$

ovvero quando

$$\gamma(A_2) \subseteq \gamma(A_1)$$

Collegando agli uco, possiamo dire che

$$A_1 \sqsubseteq A_2 \Leftrightarrow \rho_1 \sqsubseteq \rho_2 \Leftrightarrow \rho_2(C) \subseteq \rho_1(C)$$

Definizione 2.5.1 (Reticolo delle int. astratte)**.** *Se C è un reticolo completo o un cpo, allora*

$$\langle uco(C), \sqsubseteq, \sqcup, \sqcap, \lambda x.\top, \lambda x.x \rangle$$

è un reticolo completo dove $\forall \rho, \eta \in uco(C), \{\rho_i\}_{i \in I} \subseteq uco(C)$ e $x \in C$:

- $\rho \sqsubseteq \eta \Leftrightarrow \forall y \in C.\rho(y) \leq \eta(y) \Leftrightarrow \eta(C) \subseteq \rho(C)$
- $\left(\bigsqcap_{i \in I} \rho_i\right)(x) = \bigwedge_{i \in I} \rho_i(x)$
- $\left(\bigsqcup_{i \in I} \rho_i\right)(x) = x \Leftrightarrow \forall i \in I.\rho_i(x) = x$
- *$\lambda x.\top, \lambda x.x$ sono rispettivamente top e bottom.*

[1] Con isomorofismo si intendono reticoli con la stessa struttura.

2.6 Computazioni astratte e concrete

Definizione 2.6.1 (Correttezza)**.** *Data un'inserzione di Galois* $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$*, una funzione concreta* $f : C \to C$ *e una funzione astratta* $f^\sharp : A \to A$ *diciamo che* $f^\sharp$ *è un'approssimazione corretta di* f *se*

$$\forall c \in C : \alpha(f(c)) \leq_A f^\sharp(\alpha(c)) \quad \textit{backward}$$

o equivalentemente

$$\forall a \in A : f(\gamma(a)) \leq_C \gamma(f^\sharp(a) \quad \textit{forward}$$

Rinforzando la definizione e imponendo uguaglianza si perde l'equivalenza delle due espressioni sopra.

Definizione 2.6.2 (Completezza)**.** *Data un'inserzione di Galois* $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$*, una funzione concreta* $f : C \to C$ *e una funzione astratta* $f^\sharp : A \to A$ *diciamo che* $f^\sharp$ *è:*

- *backward-completa per* f *se* $\forall c \in C : \alpha(f(c)) = f^\sharp(\alpha(c))$
- *forward-completa per* f *se* $\forall a \in A : f(\gamma(a)) = \gamma(f^\sharp(a)$

La definizione rappresenta una situazione ideale in cui non si ha perdita di precisione durante il calcolo astratto. Inoltre la backward-completezza lavora sull'astrazione dell'input delle operazioni, la forward-completezza sull'output.

Le definizioni di completezza possono essere date anche usando gli uco:

- $\rho \in uco(C)$ è backward-completo per f se $\rho \circ f = \rho \circ f \circ \rho$
- $\rho \in uco(C)$ è forward-completo per f se $f \circ \rho = \rho \circ f \circ \rho$

Inoltre quando ρ è sia backward che forward-completo allora vale che $\rho \circ f = f \circ \rho$.

Teorema 2.6.1. *Data* $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$*, una funzione concreta* $f : C \to C$ *e una funzione astratta* $f^\sharp : A \to A$ *allora*

$$\forall c \in C : \alpha(f(c)) \leq_A f^\sharp(\alpha(c)) \Leftrightarrow \alpha \circ f \circ \gamma \sqsubseteq f^\sharp$$

Definizione 2.6.3 (Best correct approximation)**.** *Data* $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ *e una funzione concreta* $f : C \to C$ *allora* $\alpha \circ f \circ \gamma : A \to A$ *è la best correct approximation di* f *in* A*.*

2.7 Accelerazione della convergenza

2.7.1 Widening

Un widening

$$\nabla : P \times P \to P$$

su un poset $\langle P, \leq_P \rangle$ è una funzione che soddisfa:

- $\forall x, y \in P : x \sqsubseteq (x \nabla y) \wedge y \sqsubseteq (x \nabla y)$
- per ogni catena ascendente $x_0 \sqsubseteq x_1 \sqsubseteq ... \sqsubseteq x_n$ la catena definita come $y_0 = x_0, ..., y_{n+1} = y_n \nabla x_{n+1}$ non è strettamente crescente.

Dato che in interpretazione astratta è necessario garantire/accelerare la convergenza, viene usato il widening (che si sostituisce al least upper bound), dal momento che anche il calcolo astratto può divergere. Il risultato di un widening è un post-puntofisso di F^∇, ovvero una sovra-approssimazione del punto fisso più piccolo di f $lfp^\sqsubseteq F$.

Ad esempio, il widening su intervalli funziona come segue:

$$[a, b] \ \nabla \ [c, d] = [e, f] \qquad \text{tale che}$$

$$e = \begin{cases} -\infty & \text{se } c < a \\ a & \text{altrimenti} \end{cases} \text{ e } f = \begin{cases} +\infty & \text{se } b < d \\ b & \text{altrimenti} \end{cases}$$

2.7.2 Narrowing

Dato che il widening raggiunge un post-fixpoint, piuò capitare che si abbiano eccessive perdite di informazione, in questo caso viene usato il narrowing.

Definizione 2.7.1. *Il narrowing è una funzione* $\triangle : P \times P \to P$ *tale che:*

- $\forall x, y \in \mathcal{P} : y \leq x \implies y \leq x \triangle y \leq x$
- *Per ogni catena discendente* $x_0 \geq x_1 \geq ...$*, la catena discendente* $y_0 = x_0, ..., y_{i+1} = y_i \triangle x_{i+1}$ *non è strettamente decrescente.*

Per gli intervalli il narrowing funziona come segue:

$$[a, b] \triangle [c, d] = [e, f] \qquad \text{tale che}$$

$$e = \begin{cases} c & \text{se } a = -\infty \\ a & \text{altrimenti} \end{cases} \text{ e } f = \begin{cases} d & \text{se } b = +\infty \\ b & \text{altrimenti} \end{cases}$$

2.8 Linguaggio e semantica

Introduciamo in questa sezione il linguaggio che verrà usato nel resto della dispensa e la sua semantica.

Statement	Codice
Variabili	`x`
Espressioni aritmetiche	`e`
Assegnamenti	`x <- e`
Lettura da memoria	`x <- M[e]`
Scrittura in memoria	`M[e]`$_1$ `<-e`$_2$
Condizionali	`if (e)` S_1 `else` S_2
Salto incondizionato	`goto L`

La memoria M è vista come un array arbitrariamente lungo dove i valori possono essere inseriti e letti successivamente.

Ogni passo di computazione della semantica operazionale trasforma stati del programma (ρ, μ) dove $\rho : Var \to int$ e $\mu : \mathbb{N} \to int$. La funzione ρ mappa variabili di programma al loro valore attuale, mentre la funzione μ mappa ogni cella dell'array al suo contenuto.

Durante le computazioni sui control flow graph (CFG) ogni arco viene marcato con un'etichetta, che definisce la trasformazione di stato. Qui diamo la semantica delle possibili etichette degli archi sui CFG.

$$\begin{aligned}
[\![;]\!](\rho, \mu) &= (\rho, \mu) \\
[\![\texttt{NonZero}(e)]\!](\rho, \mu) &= (\rho, \mu) \text{ se } [\![e]\!]\rho \neq 0 \\
[\![\texttt{Zero}(e)]\!](\rho, \mu) &= (\rho, \mu) \text{ se } [\![e]\!]\rho = 0 \\
[\![\texttt{x <- e}]\!](\rho, \mu) &= (\rho[x \mapsto [\![e]\!]\rho], \mu) \\
[\![\texttt{x <- M[e]}]\!](\rho, \mu) &= (\rho[x \mapsto \mu([\![e]\!]\rho)], \mu) \\
[\![\texttt{M[e}_1\texttt{] <- e}_2]\!](\rho, \mu) &= (\rho, \mu[[\![e_1]\!]\rho \mapsto [\![e_2]\!]\rho])
\end{aligned}$$

Definiamo la modifica della memoria come

$$\rho[x \mapsto d](y) = \begin{cases} d & \text{se } x = y \\ \rho(y) & \text{altrimenti} \end{cases}$$

2.8.1 Semantica come punto fisso

Una computazione su un control flow graph è un percorso $\pi = k_1, ..., k_n$ sui suoi archi, ognuno dei quali è composto da un nodo iniziale, un'etichetta e un nodo finale $\langle u_i, lab, u_{i+1} \rangle$, con $u_i \in Edges$. Ogni trasformazione di stato è quindi la composizione degli effetti degli archi inclusi in π:

$$[\![\pi]\!] = [\![k_1]\!] \circ \cdots \circ [\![k_n]\!]$$

Definizione 2.8.1 (Fix-point semantics)**.** *Una semantica come fix-point è specificata da una coppia* $\langle D, F \rangle$ *dove:*

- *il poset* $\langle D, \leq_D \rangle$ *è il dominio semantico;*
- $F : D \to D$ *funzione totale, monotona e iterabile, è il trasformatore semantico.*

La semantica è definita attraverso sistemi di transizione.

Definizione 2.8.2 (Sistema di transizione)**.** *Un sistema di transizione è una coppia* $\langle \Sigma, \tau \rangle$ *dove:*

- Σ *è un insieme non vuoto di stati;*
- $\tau \subseteq \Sigma \times \Sigma$ *è la relazione di transizione.*

2.9 Tracce e collecting semantics

3 Analisi statica

3.1 Introduzione

L'analisi statica mira, osservando le proprietà semantica, a dire se una certa proprietà vale o no per un dato programma.

Le tipologie di analisi statica si dividono in 2 categorie:

- Control flow analysis;
- Data flow analysis (distributive e non);

3.2 Analisi su CFG

Viene generato un CFG per ogni procedura. Le analisi che vengono eseguite sono localizzate a 3 livelli:

1. **Locali al blocco:** sono eseguite all'interno di uno stesso blocco di codice;
2. **Intra-procedurali:** sono eseguite all'interno di una stessa procedura;
3. **Inter-procedurali:** sono eseguite su più funzioni/procedure.

Il pattern delle analisi di data flow consiste nel dire come l'informazione viene manipolata in un blocco. La soluzione è data da un'equazione di punto fisso.

Le analisi di data-flow possono procedere in due direzioni:

- stabilendo l'output di un blocco precedente partendo dall'input di quello corrente (*backward*);
- stabilendo l'input di un blocco successivo sulla base di cosa esce da quello corrente(*forward*).

Generalmente le analisi sono definite in un maniera simile al seguente template:

$$FAin = \begin{cases} ? & n = n_0 \\ \bigoplus\limits_{m \in pred(m)} & FAout(m) \end{cases} \qquad \underline{\text{Forward}}$$

$$BAout(n) = \begin{cases} ? & n = n_f \\ \bigoplus\limits_{m \in Succ(n)} & BAin(m) \end{cases} \qquad \underline{\text{Backward}}$$

Inoltre si definiscono *definite* la analisi dove $\bigoplus = \cap$, mentre se definiscono *possible* le analisi dove $\bigoplus = \cup$.

3.2.1 Informazione di dominanza

L'informazione di dominanza viene calcolata attraverso il punto fisso di un semplice algoritmo:

```
n <- |N| - 1
Dom(0) <-  { 0 }
for i <- 0 to n
  Dom(i) <- N

changed <- true
while (changed)
  changed <- false
  for i <- 1 to n
    temp <-  { i } ⋃ (⋂_Dom(j) j ∈ Pred(i) )
    if temp != Dom(i) then
      changed <- true
```

Al termine dell'algoritmo, il nodo d sarà in $Dom(n)$ se e solo se d dominerà n. (??)

3.2.2 Available expressions

Intuitivamente, si può dire che un'espressione e è disponibile alla variabile x ad un qualche punto di programma se:

- è stata valutata precedentemente;
- il risultato è stato assegnato ad x;
- x ed e non sono state modificate nel frattempo.

Definizione 3.2.1 (Available expression)**.** *Sia $\pi = k_1, k_2, \ldots, k_n$ un percorso dall'entry point del programma al punto v. L'espressione e è disponibile in x al punto v se:*

- *il percorso π contiene un arco k_i, etichettato con un assegnamento* `x <- e`*;*
- *nessun arco $k_{i+1}, \ldots, k_n$ è etichettato con un assegnamento ad una delle variabili $Var(e) \cup \{x\}$.*

Dato un certo blocco è possibile quindi specificare l'informazione in entrata, $AvailIn(n)$, e quella in uscita da quel blocco, $AvailOut(n)$.

$$AvailIn(n) = \begin{cases} \emptyset & \text{if } n_0 = n \\ \bigcap_{m \in Pred(n)} AvailOut(m) & \text{otherwise} \end{cases}$$

$$AvailOut(m) = Gen(m) \cup (AvailIn(m) \setminus Kill(m))$$

Per calcolare le espressioni disponibili si inizializza $AvailIn(n_0) = \emptyset$ e $AvailIn(n) = \{\text{tutte le espressioni}\} n \neq n_0$

Prima di dare l'equazione da risolvere per arrivare alla soluzione, definiamo $DEExpr(n)$: un'espressione e vi è contenuta se il blocco n valuta e e nessuno degli operandi di e è modificato tra l'ultima valutazione di e e la fine del blocco n.

L'equazione da risolvere è quindi la seguente:

$$AvailIn(n) = \bigcap_{m \in Pred(n)} (DEExpr(m) \cup (AvailIn(m) \cap Kill(m))$$

Una volta inizializzati gli insiemi $DEExpr$ e $Kill$, il compilatore esegue uno pseudocodice simile a questo per calcolare $AvailIn$ su ogni blocco:

```
for i <- 0 to N-1
  AvailIn(i) <- [AllExpr]

changed = true
while (changed)
  changed <- false
  for i <- 0 to N-1
    recompute AVailIn(i)
    if AvailIn(i) changed then
      changed <- true
```