Progetto di Linguaggi di Programmazione

Anno Accademico 2021/2022

versione del 14 dicembre 2021

Preambolo

Questo preambolo contiene informazioni generali sul progetto.

Gruppi

Il progetto può deve essere fatto da gruppi di 3 - 4 persone.

Progetto

Il termine per la consegna del progetto è il **31 gennaio 2022**.

Il progetto deve essere contenuto in un tre file con estensione `.ml`: il primo deve chiamarsi `functions.ml`, il secondo `rd.ml` ed il terzo `ae.ml`. I tre file devono essere contenuti in una cartella zippata. La cartella deve essere consegnata nell'apposito spazio sulla piattaforma `elearning`.

I files che contengono il progetto **non** devono contenere le prove che le funzioni definite sono corrette.

Punti

I punti sono ripartiti come segue: la prima parte vale 2 punti, le analisi 1,5 ciascuna. Hanno diritto al bonus solo coloro che consegnano il progetto entro il termine. La consegna in ritardo comporta una penalità secondo quanto descritto nella presentazione del corso.

Descrizione

Il linguaggio IMP: Considerate il linguaggio IMP visto a lezione e che qui viene riportato. Il linguaggio ha le seguenti categorie sintattiche:

1. espressioni aritmetiche **Aexp**, indicate con a,
2. espressioni boolenae **Bexp**, indicate con b, e
3. comandi **Com**, indicati con c,

e la sintassi (concreta) è:

$$a \quad ::= \quad n \mid \texttt{x} \mid a_0 \texttt{ + } a_1 \mid a_0 \texttt{ * } a_1 \mid a_0 \texttt{ - } a_1$$

$$b \quad ::= \quad \mathit{true} \mid \mathit{false} \mid b_0 \vee b_1 \mid b_0 \wedge b_1 \mid \neg\, b_0 \mid a_0 \texttt{ = } a_1 \mid a_0 \texttt{ < } a_1 \mid a_0 \texttt{ > } a_1$$

$$c \quad ::= \quad \texttt{skip} \mid \texttt{x := a} \mid c_0 \texttt{ ; } c_1 \mid \texttt{if } b \texttt{ then } c_0 \texttt{ else } c_1 \mid \texttt{while } b \texttt{ do } c$$

In questa sintassi $n \in \mathbb{Z}$ sono i numeri interi, mentre $\mathtt{x} \in \mathit{Var}$ sono le variabili (i nomi) dei programmi scritti in IMP. Assumiamo di avere una nuova categoria sintattica che sono i *comandi* annotati, che chiamiamo **Com*** e che indichiamo con s che è:

$$s \quad ::= \quad [\mathtt{skip}]^l \mid [\mathtt{x := a}]^l \mid s_0 \ ; \ s_1 \mid \mathtt{if}\ [b]^l\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1 \mid \mathtt{while}\ [b]^l\ \mathtt{do}\ s$$

Un comando annotato altro non è che un comando in cui, ad ogni *elemento primitivo* associamo un'etichetta $l \in \mathbf{Lab}$, che assumiamo siano numeri naturali. Gli elementi *primitivi* sono `skip`, l'assegnamento e le condizione booleana nei costrutti di scelta e di iterazione. Non lo sono i comandi che sono *composti* da altri comandi, dato che saranno queste componenti a dire quali sono gli elementi primitivi che li compongono. Se, dato un comando annotato s, denotiamo con $\mathsf{L}(s)$ l'insieme delle etichette associate agli elementi primitivi, otteniamo un insieme che ha la stessa cardinalità degli elementi primitivi presenti nel comando annotato. Questo significa che usiamo etichette distinte per ogni espressione primitiva.

Un esempio è il comando annotato (che spesso chiameremo programma) per calcolare il fattoriale di un numero che assumiamo sia associato alla variabile `x`:

$$[\mathtt{y := x}]^1 \ ; \ [\mathtt{z := 1}]^2 \ ; \ \mathtt{while}\ [\mathtt{y} > 1]^3\ \mathtt{do}\ ([\mathtt{z := z * x}]^4 \ ; \ [\mathtt{y := y - 1}]^5) \ ; \ [\mathtt{y := 0}]^6$$

In questo programma le etichette sono i numeri da 1 a 6.

Su un comando annotato è possibile fare delle analisi, che sono parte del progetto.

Reaching definitions: Questa analisi consente di stabilire per ogni punto del programma, che variabili coinvolte in assegnamenti `x := a` (che sono elementi primitivi ed hanno quindi un'etichetta) *possono raggiungere* quel certo punto senza essere state *riscritte*.

Se guardiamo il programma che calcola il fattoriale abbiamo che la variabile `x` non raggiunge alcuna etichetta (non viene mai assegnata), la `y` può raggiungere l'etichetta 6, e la `z` la posizione 4. Osserviamo che non è detto che la `z` raggiunga la posizione 4, se ad esempio calcolassimo il fattoriale di 1 la `z` raggiungerebbe solo la posizione 2.

La soluzione di questa analisi consiste nel trovare, per ogni elichetta $\ell \in \mathbf{Lab}$, una coppia di insiemi $\mathsf{RD}_{en}(\ell), \mathsf{RD}_{ex}(\ell) \subseteq \mathit{Var} \times (\mathbf{Lab} \cup \{?\})$ che contengono le variabili del programma ed il punto in cui sono state *assegnate*. Nel caso del programma per il fattoriale abbiamo:

ℓ	RD_{en}	RD_{ex}
1	$\{(\mathtt{x},?),(\mathtt{y},?),(\mathtt{z},?)\}$	$\{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{z},?)\}$
2	$\{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{z},?)\}$	$\{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{z},2)\}$
3	$\{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{y},5),(\mathtt{z},2),(\mathtt{z},4)\}$	$\{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{y},5),(\mathtt{z},2),(\mathtt{z},4)\}$
4	$\{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{y},5),(\mathtt{z},2),(\mathtt{z},4)\}$	$\{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{y},5),(\mathtt{z},4)\}$
5	$\{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{y},5),(\mathtt{z},4)\}$	$\{(\mathtt{x},?),(\mathtt{y},5),(\mathtt{z},4)\}$
6	$\{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{y},5),(\mathtt{z},2),(\mathtt{z},4)\}$	$\{(\mathtt{x},?),(\mathtt{y},6),(\mathtt{z},2),(\mathtt{z},4)\}$

Prendiamo la prima riga, in entrata ogni variabile ha come "etichetta" il '?', mentre in uscita abbiamo che la `y` è stata assegnata in quel punto. Se guardiamo la terza riga (la condizione booleana del `while`) abbiamo che in entrata le variabili sono o la `y` modificata al punto 1 o quella al punto 5, mentre la `z` è o quella modificata al punto 2 o quella modificata al punto 4, così come in uscita, dato che la condizione booleana non modifica alcun assegnamento. La quarta riga ha come RD_{en} l'RD_{ex} del punto precedente ma in uscita ha che l'insieme non conterrà più la coppia $(\mathtt{z}, 2)$ dato la `z` è stata modificata.

Vedremo in seguito come questa va calcolata.

Available expressions: Questa analisi consente di stabilire, per ogni punto del programma, quali espressioni devono essere state calcolate, e non modificate successivamente, su tutti i possibili cammini che portano a quel punto del programma. Un cammino nel programma altro non è che la sequenza, con i cicli, dei comandi eseguiti in quel programma. Guardiamo il programma

$$[\texttt{x := a + b}]^1 \ ; \ [\texttt{y := a * b}]^2 \ ; \ \texttt{while}\ [\texttt{y} > \texttt{a + b}]^3\ \texttt{do}\ ([\texttt{a := a + 1}]^4 \ ; \ [\texttt{x := a + b}]^5)$$

Si vede che l'espressione `a + b` è *available* ogni volta che l'esecuzione raggiunge il test del loop (il `while`), quindi non è necessario ricalcolarla, è stata calcolata in precedenza, o al punto 1 o al punto 5.

Anche in questo caso l'analisi restuisce, per ogni punto del programma, quali sono le espressioni disponibili in entrata e quelle in uscita. Abbiamo quindi AE_{en} ed AE_{ex} per ogni punto del programma, che sono sottoinsiemi di **Aexp**. Se consideriamo il programma precedente abbiamo:

ℓ	AE_{en}	AE_{ex}
1	$\emptyset$	{a + b}
2	{a + b}	{a + b, a * b}
3	{a + b}	{a + b}
4	{a + b}	$\emptyset$
5	$\emptyset$	{a + b}

Per il primo punto in entrata non abbiamo alcuna espressione, mentre in uscita abbiam l'espressione `a + b` che è anche in entrata al punto successivo, dove in uscita si aggiunge l'espressione `a * b`. Nella valutazione dell'espressione booleana - il punto 3 - abbiamo che sia in entrata che in uscita abbiamo solo l'espressione `a + b` che è quella usata nella condizione medesima. In uscita dal punto 4 invece abbiamo l'insieme vuoto dato che le espressioni contengono la variabile `a` che è stata aggiornata.

Come compiere le analisi: Per compiere le analisi dobbiamo definire un certo numero di operazioni su programmi ed etichette, che sono comuni a tutte le varie analisi. La prima è la funzione *init* : **Com*** $\rightarrow$ **Lab** così definita:

$$\begin{aligned}
\mathit{init}([\texttt{x := a}]^\ell) &= \ell \\
\mathit{init}([\texttt{skip}]^\ell) &= \ell \\
\mathit{init}([s_1 \ ; \ s_2]^\ell) &= \mathit{init}(s_1) \\
\mathit{init}(\texttt{while}\ [b]^\ell\ \texttt{do}\ s) &= \ell \\
\mathit{init}(\texttt{if}\ [b]^\ell\ \texttt{then}\ s_1\ \texttt{else}\ s_2) &= \ell
\end{aligned}$$

mentre la seconda è la funzione *final* : **Com*** $\rightarrow 2^{\mathbf{Lab}}$

$$\begin{aligned}
\mathit{final}([\texttt{x := a}]^\ell) &= \{\ell\} \\
\mathit{final}([\texttt{skip}]^\ell) &= \{\ell\} \\
\mathit{final}([s_1 \ ; \ s_2]^\ell) &= \mathit{final}(s_2) \\
\mathit{final}(\texttt{while}\ [b]^\ell\ \texttt{do}\ s) &= \{\ell\} \\
\mathit{final}(\texttt{if}\ [b]^\ell\ \texttt{then}\ s_1\ \texttt{else}\ s_2) &= \mathit{final}(s_1) \cup \mathit{final}(s_2)
\end{aligned}$$

Queste due funzioni restituiscono rispettivamente il punto d'ingresso ed i punti d'uscita di ogni costrutto. Per semplicità chiamiamo gli elementi primitivi **Blocks**, che indicheremo con B, e definiamo

una funzione $blocks : \mathbf{Com*} \to 2^{\mathbf{Blocks}}$ in questo modo:

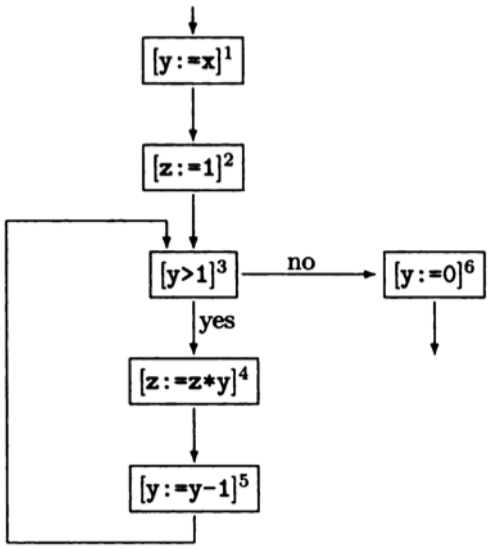
$$\begin{aligned}
blocks([\texttt{x := a}]^\ell) &= \{[\texttt{x := a}]^\ell\} \\
blocks([\texttt{skip}]^\ell) &= \{[\texttt{skip}]^\ell\} \\
blocks([s_1 \ ; \ s_2]^\ell) &= blocks(s_1) \cup blocks(s_2) \\
blocks(\texttt{while } [b]^\ell \texttt{ do } s) &= \{[b]^\ell\} \cup blocks(s) \\
blocks(\texttt{if } [b]^\ell \texttt{ then } s_1 \texttt{ else } s_2) &= \{[b]^\ell\} \cup blocks(s_1) \cup blocks(s_2)
\end{aligned}$$

Questa funzione restituisce gli elementi primitivi di un programma, e su questa possiamo definire la funzione $labels : \mathbf{Com*} \to 2^{\mathbf{Lab}}$ come $labels(s) = \{\ell \mid [B]^\ell \in blocks(s)\}$.

Da ultimo definiamo una funzione *flow* che associa ad un programma i possibili flussi delle esecuzioni:

$$\begin{aligned}
flow([\texttt{x := a}]^\ell) &= \emptyset \\
flow([\texttt{skip}]^\ell) &= \emptyset \\
flow([s_1 \ ; \ s_2]^\ell) &= flow(s_1) \cup flow(s_2) \cup \{(\ell, init(s_2)) \mid \ell \in final(s_1)\} \\
flow(\texttt{while } [b]^\ell \texttt{ do } s) &= flow(s) \cup \{(\ell, init(s))\} \cup \{(\ell', \ell) \mid \ell' \in final(s)\} \\
flow(\texttt{if } [b]^\ell \texttt{ then } s_1 \texttt{ else } s_2) &= flow(s_1) \cup flow(s_2) \cup \{(\ell, init(s_1)), (\ell, init(s_2))\}
\end{aligned}$$

Con l'ausilio di queste funzioni possiamo disegnare il *flow* graph di un programma s che ha come nodi l'insieme $\{\ell \mid [B]^\ell \in blocks(s)\}$ e come archi gli elementi di $flow(s)$. Di seguito il flow graph del programma per il fattoriale che è servito come esempio dell'analisi reaching definitions.



Oltre a queste funzioni generali, introduciamo delle funzioni specifiche alle due analisi che ci interessano. Per l'analisi reaching definitions, abbiamo $kill_{\mathsf{RD}} : \mathbf{Blocks} \to 2^{Var \times \mathbf{Lab}}$ e $gen_{\mathsf{RD}} : \mathbf{Blocks} \to 2^{Var \times \mathbf{Lab}}$ mentre per l'analisi available expressions abbiamo $kill_{\mathsf{AE}} : \mathbf{Blocks} \to 2^{\mathbf{Aexp}}$ e $gen_{\mathsf{AE}} : \mathbf{Blocks} \to 2^{\mathbf{Aexp}}$.

Le funzioni sono definite come segue (a partire da $s \in \mathbf{Com*}$)

$$\begin{aligned}
kill_{\mathsf{RD}}([\texttt{x := a}]^\ell) &= \{(\texttt{x}, ?)\} \cup \{(\texttt{x}, \ell) \mid [\texttt{x := a}']^\ell \in blocks(s)\} \\
kill_{\mathsf{RD}}([\texttt{skip}]^\ell) &= \emptyset \\
kill_{\mathsf{RD}}([b]^\ell) &= \emptyset
\end{aligned}$$

$$\begin{aligned}
gen_{\mathsf{RD}}([\texttt{x := a}]^\ell) &= \{(\texttt{x}, \ell)\} \\
gen_{\mathsf{RD}}([\texttt{skip}]^\ell) &= \emptyset \\
gen_{\mathsf{RD}}([b]^\ell) &= \emptyset
\end{aligned}$$

Queste funzioni consentono di calcolare RD_{en} e RD_{ex} come segue:

$$\begin{array}{rcl} \mathsf{RD}_{en}(\ell) & = & \begin{cases} \{(\mathtt{x}, ?) \mid \mathtt{x} \in FV(s)\} & \textit{se} \quad \ell \in \textit{init}(s) \\ \bigcup\{\mathsf{RD}_{ex}(\ell') \mid (\ell', \ell) \in \textit{flow}(s)\} & \textit{altrimenti} \end{cases} \\ \mathsf{RD}_{ex}(\ell) & = & (\mathsf{RD}_{en}(\ell) \setminus \textit{kill}_{\mathsf{RD}}(B^\ell)) \cup \textit{gen}_{\mathsf{RD}}(B^\ell) \quad \textit{dove} \quad B^\ell \in \textit{blocks}(s) \end{array}$$

Figura 1: Come sono definiti gli insiemi dell'analisi *reaching definitions*

Consideriamo come calcolare questi insiemi, che indichiamo con $\overline{\mathsf{RD}}$. L'operatore $\mathcal{F}_{\mathsf{RD}}$ che calcola le reaching definitions sarà tale che per la soluzione del problema $\overline{\mathsf{RD}}$ vale che $\mathcal{F}_{\mathsf{RD}}(\overline{\mathsf{RD}}) = \overline{\mathsf{RD}}$.

Per trovare tale soluzione usiamo il fatto che $2^{Var \times \mathbf{Lab}}$ è ordinato parzialmente per inclusione, e possiamo adattare la precedente definizione (in Figura 1) incrementalmente usando il seguente adattamento:

$$\begin{array}{rcl} \mathsf{RD}^0_{en}(\ell) & = & \emptyset \\ \mathsf{RD}^n_{en}(\ell) & = & \begin{cases} \{(\mathtt{x}, ?) \mid \mathtt{x} \in FV(s)\} & \textit{se} \quad \ell \in \textit{init}(s) \\ \bigcup\{\mathsf{RD}^{n-1}_{ex}(\ell') \mid (\ell', \ell) \in \textit{flow}(s)\} & \textit{altrimenti} \end{cases} \\ \mathsf{RD}^0_{ex}(\ell) & = & \emptyset \\ \mathsf{RD}^n_{ex}(\ell) & = & (\mathsf{RD}^{n-1}_{en}(\ell) \setminus \textit{kill}_{\mathsf{RD}}(B^\ell)) \cup \textit{gen}_{\mathsf{RD}}(B^\ell) \quad \textit{dove} \quad B^\ell \in \textit{blocks}(s) \end{array}$$

Figura 2: Calcolo incrementale degli insiemi dell'analisi *reaching definitions*

Mi fermo quando arrivo ad una iterazione dove per ogni $\ell \in \mathbf{Lab}$ abbiamo che $\mathsf{RD}^n_{en}(\ell) = \mathsf{RD}^{n-1}_{en}(\ell)$ e $\mathsf{RD}^n_{ex}(\ell) = \mathsf{RD}^{n-1}_{ex}(\ell)$. Nel caso del programma fattoriale $\overline{\mathsf{RD}}$ è la ennupla

$$(\mathsf{RD}_{en}(1), \mathsf{RD}_{en}(2), \mathsf{RD}_{en}(3), \mathsf{RD}_{en}(4), \mathsf{RD}_{en}(5), \mathsf{RD}_{en}(6), \mathsf{RD}_{ex}(1), \mathsf{RD}_{ex}(2), \mathsf{RD}_{ex}(3), \mathsf{RD}_{ex}(4), \mathsf{RD}_{ex}(5), \mathsf{RD}_{ex}(6))$$

che abbiamo visto prima.

Per le funzioni che riguardano le available expressions abbiamo bisogno di ricordare quali sono le free variable di una espressione aritmetica ($FV(n) = \emptyset, FV(\mathtt{x}) = \{\mathtt{x}\}$ e $FV(a_1 \mathtt{op} a_2) = FV(a_1) \cup FV(a_2)$ con $\mathtt{op} = \{\mathtt{+}, \mathtt{*}, \mathtt{-}\}$), mentre se scriviamo $\mathbf{Aexp}(\mathtt{a})$ indichiamo le espressioni aritmetiche che contengono $\mathtt{a}$ come sottoespressione, e con $\mathbf{Aexp}(b)$ le espressioni aritmetiche contenute nell'espressione booleana b.

$$\begin{array}{rcl} \textit{kill}_{\mathsf{AE}}([\mathtt{x := a}]^\ell) & = & \{\mathtt{a}' \in \mathbf{Aexp} \mid \mathtt{x} \in FV(\mathtt{a}')\} \\ \textit{kill}_{\mathsf{AE}}([\mathtt{skip}]^\ell) & = & \emptyset \\ \textit{kill}_{\mathsf{AE}}([b]^\ell) & = & \emptyset \\ \\ \textit{gen}_{\mathsf{AE}}([\mathtt{x := a}]^\ell) & = & \{\mathtt{a}' \in \mathbf{Aexp}(\mathtt{a}) \mid \mathtt{x} \notin FV(\mathtt{a}')\} \\ \textit{gen}_{\mathsf{AE}}([\mathtt{skip}]^\ell) & = & \emptyset \\ \textit{gen}_{\mathsf{AE}}([b]^\ell) & = & \mathbf{Aexp}(b) \end{array}$$

Come prima queste funzioni consentono di calcolare AE_{en} e AE_{ex} come segue:

$$
\begin{array}{rcl}
\mathsf{AE}_{en}(\ell) & = & \begin{cases} \emptyset & se \quad \ell \in init(s) \\ \bigcap\{\mathsf{AE}_{ex}(\ell') \mid (\ell', \ell) \in flow(s)\} & altrimenti \end{cases} \\
\mathsf{AE}_{ex}(\ell) & = & (\mathsf{AE}_{en}(\ell) \setminus kill_{\mathsf{AE}}(B^\ell)) \cup gen_{\mathsf{AE}}(B^\ell) \quad dove \quad B^\ell \in blocks(s)
\end{array}
$$

Figura 3: Come sono definiti gli insiemi dell'analisi *available expressions*

Anche in questo caso calcoliamo incrementalmente, solo che ora $2^{\mathbf{Aexp}}$ ha l'ordine inverso rispetto all'inclusione su insiemi, che indichiamo con $\supseteq$, e l'operatore $\mathcal{F}_{\mathsf{AE}}$ userà incrementalmente questa riscrittura della definizione in Figura 4:

$$
\begin{array}{rcl}
\mathsf{AE}^0_{en}(\ell) & = & \mathbf{Aexp} \\
\mathsf{AE}^n_{en}(\ell) & = & \begin{cases} \emptyset & se \quad \ell \in init(s) \\ \bigcap\{\mathsf{AE}^{n-1}_{ex}(\ell') \mid (\ell', \ell) \in flow(s)\} & altrimenti \end{cases} \\
\mathsf{AE}^0_{ex}(\ell) & = & \mathbf{Aexp} \\
\mathsf{AE}^n_{ex}(\ell) & = & (\mathsf{AE}^{n-1}_{en}(\ell) \setminus kill_{\mathsf{AE}}(B^\ell)) \cup gen_{\mathsf{AE}}(B^\ell) \quad dove \quad B^\ell \in blocks(s)
\end{array}
$$

Figura 4: Come calcolare incrementalmente gli insiemi dell'analisi *available expressions*

Anche in questo caso le iterazioni terminano quando si arriva ad un punto fisso, che in questo caso è il massimo.

Progetto

Il progetto si compone di tre parti, che vengono descritte di seguito.

Prima parte: In questa parte si devono implementare le funzioni

- `fv` : **Com** $\to 2^{Var}$. Questa funzione restituisce gli identificatori presenti nel programma.
- `annotate` : **Com** $\to$ **Com*** . Questa funzione riceve in ingresso un programma e restituisce un programma annotato
- `init` : **Com*** $\to$ **Lab**. Questa funzione implementa la *init* descritta prima
- `final` : **Com*** $\to 2^{\mathbf{Lab}}$. Questa funzione implementa la *final* descritta prima
- `blocks` : **Com*** $\to 2^{\mathbf{Blocks}}$. Questa funzione implementa la *blocks* descritta prima
- `label` : **Com*** $\to 2^{\mathbf{Lab}}$. Questa funzione implementa la *label* descritta prima
- `flow` : **Com*** $\to 2^{\mathbf{Lab}\times\mathbf{Lab}}$. Questa funzione implementa la *flow* descritta prima
- `check` : **Com*** $\to$ `bool` verifica che le etichette assegnate nel programma siano uniche

Gli insiemi sono rappresentati come list del tipo opportuno (dovete anche implementare l'inserzione in un insieme, l'estrazione da un insieme e l'appartenenza). L'insieme vuoto è la lista vuota. I tipi li trovate nella cartella allegata.

Seconda parte: In questa parte si deve realizzre l'analisi reaching definition, che consiste nel calcolare, dato un programma c, gli insiemi $\mathsf{RD}_x(\ell)$ dove $x \in \{en, ex\}$ per ogni $\ell \in label(ann(c))$. La funzione ann è quella che annota il programma c.

L'analisi deve calcolare la ennupla di insiemi secondo lo schema suggerito, e le ennuple sono in questo caso rappresentate come una coppia di liste, una contenente gli insiemi RD_{en}, mentre l'altra contiene gli insiemi RD_{ex}. l'etichetta ℓ, che è un intero, è l'indice dell'elemento nella lista.

Quindi, dato il tipo $Var \times$ **lab** `list`, l'analisi RD è una funzione $\mathsf{RD}functor : \mathtt{RDlist} \times \mathtt{RDlist} \to \mathtt{RDlist} \times \mathtt{RDlist}$ dove le due liste sono la *entry* e la *exit* list.

Riassumendo, dovete scrivere una funzione `rdanalisi` che riceve in input un comando e restituisce il comando annotato e le due liste di insiemi.

Terza parte: In questa parte si deve realizzare l'analisi available expressions, che consiste nel calcolare, dato un programma c, gli insiemi $\mathsf{AE}_x(\ell)$ dove $x \in \{en, ex\}$ per ogni $\ell \in label(ann(c))$. La funzione ann è quella che annota il programma c.

L'analisi deve calcolare la ennupla di insiemi secondo lo schema suggerito, e le ennuple sono in questo caso rappresentate come una coppia di liste, una contenente gli insiemi AE_{en}, mentre l'altra contiene gli insiemi AE_{ex}. l'etichetta ℓ, che è un intero, è l'indice dell'elemento nella lista.

Tale analisi ha bisogno di una ulteriore funzione, la `expressions` : $\mathbf{Com} \to 2^{\mathbf{Aexp}}$ che per ogni programma c restituisce le espressioni coinvolte (quelle che sono il *lato destro* di un assegnamento.).

Come per il secondo punto, l'analisi sarà $\mathsf{AE}functor : \mathtt{AElist} \times \mathtt{AElist} \to \mathtt{AElist} \times \mathtt{AElist}$ dove le due liste sono la *entry* e la *exit* list.

Anche in questo caso dovete scrivere una funzione `aeanalisi` che riceve in input un comando e restituisce il comando annotato e le due liste di insiemi.

Riferimenti

Flemming Nielson, Hanne Riis Nielson, Chris Hankin, **Principles of Program Analysis**, Springer, 2005