Analisi Statica

Introduzione

L'analisi statica mira all'estrazione di proprietà sintattiche o semantiche valide *per ogni esecuzione* di un dato programma P. Distinguiamo principalmente due tipi di analisi: *distributive* ed *non-distributive*. Le prime usualmente studiano **come** avanza il flusso di esecuzione, mentre le altre si concentrano su **cosa** viene calcolato dal programma.

Tali analisi usano come strumento principe il *CFG* (*Control Flow Graph*), ovvero un grafo che rappresenta il flusso di controllo di un programma o di una procedura. È rappresentato mediante un grafo orientato con possibili cicli, avente un unico punto di ingresso e un unico punto di uscita, i cui nodi sono *basic block* mentre gli archi sono le possibili transizioni tra blocchi.
Un basic block è una sequenza di istruzioni del programma priva di costrutti di controllo (tutte le istruzioni che lo compongono sono eseguite sequenzialmente, senza salti o cicli). Per costruirlo quindi si crea un blocco ogni volta che c'è un salto.

QUALCOSA SULL'ASTRAZIONE

Prima distinzione:

❄ Forward

$$\text{FAin}(n) = \begin{cases} \emptyset & n = entry \\ \bigoplus_{m \in \text{Pred}(n)} \text{FAout}(m) & \text{otherwise} \end{cases}$$

$$\text{FAout}(m) = \tau(\text{FAin}(m))$$
$$\tau(\text{FAin}(m)) = \text{gen}(m) \cup (\text{FAout}(m) \setminus \text{kill}(m))$$

❄ Backward

$$\text{BAout}(n) = \begin{cases} \emptyset & n = exit \\ \bigoplus_{m \in \text{Succ}(n)} \text{BAin}(m) & \text{otherwise} \end{cases}$$

$$\text{BAin}(m) = \tau(\text{BAout}(m))$$
$$\tau(\text{BAout}(m)) = \text{gen}(m) \cup (\text{BAin}(m) \setminus \text{kill}(m))$$

Seconda distinzione:

❄ Possible analysis: $\bigoplus = \bigcup$

❄ Definite analysis: $\bigoplus = \bigcap$

Available Expressions

Forward & Definite

Un espressione del tipo $x \leftarrow e$ si dice *available* (disponibile) in un dato punto P di programma se:

- ❄ È definita in un blocco B precedente a P
- ❄ x non è ridefinita tra B e P
- ❄ Le variabili di e non sono ridefinite tra B e P

Per ottimizzare il codice si può sostituire il calcolo di un'espressione disponibile con la deferenziazione della variabile in cui è stata assegnata.

Proprietà dei basic blocks

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

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

$$\text{Gen}_A(n) = \left\{ x \leftarrow e \;\middle|\; \begin{array}{c} x \leftarrow e \in n \\ \wedge \\ x \notin \text{Var}(e) \end{array} \right\}$$

$$\text{Kill}_A(n) = \left\{ x \leftarrow e \;\middle|\; \exists y \leftarrow e' \in n.\ \left(y \in \text{Var}(e) \vee y = x\right) \right\}$$

Semantica

$$\text{Dominio: } \mathbb{P}(\text{Ass})$$

$$[\![;]\!]^\# A = A$$

$$[\![\text{zero}(e)]\!]^\# A = A$$

$$[\![\text{non-zero}(e)]\!]^\# A = A$$

$$[\![M[e_1] \leftarrow e_2]\!]^\# A = A$$

$$[\![x \leftarrow e]\!]^\# A = \begin{cases} A \setminus \text{Occ}(x) \cup \{x \leftarrow e\} & x \notin \text{Var}(e) \\ A \setminus \text{Occ}(x) & \text{otherwise} \end{cases}$$

$$[\![x \leftarrow M[e]]\!]^\# A = A \setminus \text{Occ}(x)$$

$$\text{where } \text{Occ}(x) = \left\{ y \leftarrow e \;\middle|\; \begin{array}{c} x \in \text{Var}(e) \\ \vee \\ y = x \end{array} \right\}$$

Liveness

Backward & Possible

Una variabile è viva in ogni punto di programma tra la sua definizione e il suo ultimo uso. Per calcolarle partiamo dal fondo e saliamo fino ad un uso della variabile; questa variabile resta viva finchè non troviamo la sua definizione, sopra la quale risulta non viva. Se una avariabile è viva vuol dire che il suo valore può essere letto. È un'analisi di tipo possible, poichè è sufficiente che sia viva in un ramo per renderla viva in quel punto di programma.

Questa analisi risulta utile per cercare codice morto, variabili non inizializzate o registri liberi.

Proprietà dei basic blocks

$$\begin{aligned}
\mathrm{LiveOut}(n) &= \begin{cases} \emptyset & n = exit \\ \bigcup_{m \in \mathrm{Succ}(n)} \mathrm{LiveIn}(m) & \end{cases} \\
\mathrm{LiveIn}(m) &= \mathrm{Gen}(m) \cup (\mathrm{LiveOut}(m) \setminus \mathrm{Kill}(m)) \\
\mathrm{Gen}_L(n) &= \left\{ x \;\middle|\; \exists e \in n.\, x \in \mathrm{Var}(e) \right\} \\
\mathrm{Kill}_L(n) &= \left\{ x \;\middle|\; \exists x \leftarrow e \in n \right\}
\end{aligned}$$

Semantica

$$\begin{aligned}
\text{Dominio: } \mathbb{P}(\mathrm{Var}) \\
[\![;]\!]^{\#} L &= L \\
[\![\mathrm{zero}(e)]\!]^{\#} L &= L \cup \mathrm{Var}(e) \\
[\![\text{non-zero}(e)]\!]^{\#} L &= L \cup \mathrm{Var}(e) \\
[\![M[e_1] \leftarrow e_2]\!]^{\#} L &= L \cup \mathrm{Var}(e_1) \cup \mathrm{Var}(e_2) \\
[\![x \leftarrow e]\!]^{\#} L &= (L \setminus \{x\}) \cup \mathrm{Var}(e) \\
[\![x \leftarrow M[e]]\!]^{\#} L &= (L \setminus \{x\}) \cup \mathrm{Var}(e)
\end{aligned}$$

True Liveness

Backward & Possible

$$\text{TLiveOut}(n) = \begin{cases} \emptyset & n = exit \\ \bigcup_{m \in \text{Succ}(n)} \text{TLiveIn}(m) & \end{cases}$$

$$\text{TLiveIn}(m) = \text{Gen}(m) \cup (\text{TLiveOut}(m) \setminus \text{Kill}(m))$$

$$\text{Gen}_{TL}(n) = \left\{ x \;\middle|\; x \in TL \wedge \exists e \in n.\, x \in \text{Var}(e) \right\}$$

$$\text{Kill}_{TL}(n) = \left\{ x \;\middle|\; \exists x \leftarrow e \in n \right\}$$

Semantica

$$\text{Dominio: } \mathbb{P}(\text{Var})$$

$$[\![;]\!]^{\#} TL = TL$$

$$[\![\text{zero}(e)]\!]^{\#} TL = TL \cup \text{Var}(e)$$

$$[\![\text{non-zero}(e)]\!]^{\#} TL = TL \cup \text{Var}(e)$$

$$[\![M[e_1] \leftarrow e_2]\!]^{\#} TL = TL \cup \text{Var}(e_1) \cup \text{Var}(e_2)$$

$$[\![x \leftarrow e]\!]^{\#} TL = \begin{cases} (TL \setminus \{x\}) \cup \text{Var}(e) & x \in TL \\ (TL \setminus \{x\}) & \text{otherwise} \end{cases}$$

$$[\![x \leftarrow M[e]]\!]^{\#} TL = \begin{cases} (TL \setminus \{x\}) \cup \text{Var}(e) & x \in TL \\ (TL \setminus \{x\}) & \text{otherwise} \end{cases}$$

Very Busy Expressions

Backward & Definite

$$\mathrm{VBOut}(n) = \begin{cases} \emptyset & n = exit \\ \bigcap_{m \in \mathrm{Succ}(n)} \mathrm{VBIn}(m) & \end{cases}$$

$$\mathrm{VBIn}(m) = \mathrm{Gen}(m) \cup (\mathrm{VBOut}(m) \setminus \mathrm{Kill}(m))$$

$$\mathrm{Gen}_{VB}(n) = \left\{ x \leftarrow e \in n \;\middle|\; x \notin \mathrm{Var}(e) \right\}$$

$$\mathrm{Kill}_{VB}(n) = \left\{ x \leftarrow e \;\middle|\; \exists y \leftarrow e' \in n.\ \begin{array}{rl} x \in \mathrm{Var}(e') & \vee \\ y \in \mathrm{Var}(e) & \vee \\ x = y & \end{array} \right\}$$

Semantica

$$\text{Dominio: } \mathbb{P}(\mathrm{Ass})$$

$$[\![;]\!]^{\#} VB = VB$$

$$[\![\mathrm{zero}(e)]\!]^{\#} VB = VB \setminus \mathrm{Ass}(e)$$

$$[\![\text{non-zero}(e)]\!]^{\#} VB = VB \setminus \mathrm{Ass}(e)$$

$$[\![M[e_1] \leftarrow e_2]\!]^{\#} VB = VB \setminus (\mathrm{Ass}(e_1) \cup \mathrm{Ass}(e_2))$$

$$[\![x \leftarrow e]\!]^{\#} VB = \begin{cases} VB \setminus (\mathrm{Occ}(x) \cup \mathrm{Ass}(e)) \cup \{x \leftarrow e\} & \text{if } x \notin \mathrm{Var}(e) \\ VB \setminus (\mathrm{Occ}(x) \cup \mathrm{Ass}(e)) & \text{otherwise} \end{cases}$$

$$[\![x \leftarrow M[e]]\!]^{\#} VB = VB \setminus (\mathrm{Occ}(x) \cup \mathrm{Ass}(e))$$

$$\text{where } \mathrm{Occ}(x) = \left\{ y \leftarrow e \;\middle|\; \begin{array}{c} x \in \mathrm{Var}(e) \\ \vee \\ y = x \end{array} \right\}$$

$$\text{and } \mathrm{Ass}(e) = \left\{ y \leftarrow e' \in Ass \;\middle|\; y \in \mathrm{Var}(e) \wedge e \neq M[e] \right\}$$

Reaching Definitions

Forward & Possible

Le reaching definitions ci dicono, per ogni punto di programma, le variabili che sono disponibili e in che punto di programma sono state definite. Per ogni punto quindi si ha un insieme di coppie. A seconda del cammino percorso le definizioni disponibili potrebbero essere diverse, per questo ad una variabile possono corrispondere più punti di programma, rendendo così l'analisi di tipo possible.

Proprietà dei basic blocks

$$\mathrm{RDIn}(n) = \begin{cases} \emptyset & n = entry \\ \bigcup_{m \in \mathrm{Pred}(n)} \mathrm{RDOut}(m) & \end{cases}$$

$$\mathrm{RDOut}(m) = \mathrm{Gen}(m) \cup (\mathrm{RDIn}(m) \setminus \mathrm{Kill}(m))$$

$$\mathrm{Gen}_{RD}(n) = \left\{ (x,n) \;\middle|\; \exists x \leftarrow e \in n \right\}$$

$$\mathrm{Kill}_{RD}(n) = \left\{ (x,n') \;\middle|\; \exists x \leftarrow e \in n \wedge \exists (x,n') \in RD(n) \right\}$$

Semantica

$$\text{Dominio: } \mathbb{P}(\mathrm{Var} \times \mathrm{ProgPoints})$$

$$[\![;]\!]^{\#} RD = RD$$

$$[\![\mathrm{zero}(e)]\!]^{\#} RD = RD$$

$$[\![\text{non-zero}(e)]\!]^{\#} RD = RD$$

$$[\![M[e_1] \leftarrow e_2]\!]^{\#} RD = RD$$

$$[\![x \leftarrow e]\!]^{\#} RD = (RD \setminus \mathrm{Def}(x)) \cup \left\{ (x,u) \;\middle|\; K = (u, x \leftarrow e, v) \right\}$$

$$[\![x \leftarrow M[e]]\!]^{\#} RD = (RD \setminus \mathrm{Def}(x)) \cup \left\{ (x,u) \;\middle|\; K = (u, x \leftarrow e, v) \right\}$$

$$\text{where } \mathrm{Def}(x) = \left\{ (x,n) \;\middle|\; n \text{ punto di programma} \right\}$$

$$\text{and } K = \begin{array}{l} \text{Arco che stiamo analizzando, rappresentato da una} \\ \text{tupla di: [nodo di partenza, istruzione, nodo di arrivo]} \end{array}$$

Copy Propagation

Forward & Definite

L'analisi di copy propagation ci dice, per ogni punto di programma, l'insieme di coppie di variabili che contengono lo stesso valore. È un'analisi di tipo forward e definite, che può essere utilizzata nelle ottimizzazione per evitare copie inutili. L'unica istruzione che genera una nuova coppia è $x \leftarrow y$, mentre ogni altra modifica di una di queste variabili uccide tutte le coppie in cui è presente.

Proprietà dei basic blocks

$$\begin{aligned}
\text{CopyIn}(n) &= \begin{cases} \emptyset & n = entry \\ \bigcap_{m \in \text{Pred}(n)} \text{CopyOut}(m) & \end{cases} \\
\text{CopyOut}(m) &= \text{Gen}(m) \cup (\text{CopyIn}(m) \setminus \text{Kill}(m)) \\
\text{Gen}_C(n) &= \left\{ (x\ y) \;\middle|\; \exists x \leftarrow y \in n \right\} \\
\text{Kill}_C(n) &= \left\{ (x\ y) \;\middle|\; \exists x \leftarrow e \in n \vee \exists y \leftarrow e \in n \right\}
\end{aligned}$$

Semantica

$$\begin{aligned}
\text{Dominio: } & \mathbb{P}(\text{Var} \times \text{Var}) \\
[\![;]\!]^\# C &= C \\
[\![\text{zero}(e)]\!]^\# C &= C \\
[\![\text{non-zero}(e)]\!]^\# C &= C \\
[\![M[e_1] \leftarrow e_2]\!]^\# C &= C \\
[\![x \leftarrow e]\!]^\# C &= C \setminus \text{Copie}(x) \quad e \notin Var \\
[\![x \leftarrow M[e]]\!]^\# C &= C \setminus \text{Copie}(x) \\
[\![x \leftarrow y]\!]^\# C &= (C \setminus \text{Copie}(x)) \cup (x\ y) \cup \left\{ (x\ z) \;\middle|\; (z\ y) \in C \right\}
\end{aligned}$$

$$\text{where } \text{Copie}(x) = \left\{ (x\ y) \;\middle|\; (x\ y) \in C \right\}$$

Intervals

Non-Distributive, Forward & Possible

Questa analisi è solo semantica, in quanto ci dice cosa calcola un programma. Per ogni punto viene associato ad ogni variabile un intervallo che comprende i valori che questa può assumere. È perciò necessario definire nella semantica le operazioni tra intervalli per ogni possibile istruzione, incluse operazioni aritmetiche e di confronto.

In alcuni casi però i cicli possono essere ripetuti molte volte, per cui si utilizza il **widening**: se dopo due iterazioni un estremo di un intervallo continua a crescere lo si approssima con l'infinito. Una volta raggiunto il punto fisso è possibile applicare il narrowing per restringere l'intervallo.

Semantica

$$\text{Dominio: } \mathbb{I} = \{[l,u] \mid l \in \mathbb{Z} \cup \{-\infty\},\ u \in \mathbb{Z} \cup \{+\infty\},\ l \leq u\}$$

$$[\![;]\!]^{\#} I = I$$

$$[\![\text{zero}(e)]\!]^{\#} I = \begin{cases} \bot & [0,0] \not\sqsubseteq [\![e]\!]^{\#} I \\ I \sqcap [\![e]\!]^{\#} I & \textit{otherwise} \end{cases}$$

$$[\![\text{non-zero}(e)]\!]^{\#} I = \begin{cases} \bot & [0,0] = [\![e]\!]^{\#} I \\ I \sqcap [\![e]\!]^{\#} I & \textit{otherwise} \end{cases}$$

$$[\![M[e_1] \leftarrow e_2]\!]^{\#} I = I$$

$$[\![x \leftarrow e]\!]^{\#} I = I[x \mapsto [\![e]\!]^{\#} I]$$

$$[\![x \leftarrow M[e]]\!]^{\#} I = I[x \mapsto \top]$$

Widening (∇)

$$\bot \nabla I = I \nabla \bot = I$$

$$(I_1 \nabla I_2)(x) = I_1(x) \nabla I_2(x) \text{ where } [l_1,\ u_1] \nabla [l_2,\ u_2] = [l,\ u] \text{ such that}$$

$$l = \begin{cases} l_1 & \text{if } l_1 \leq l_2 \\ -\infty & \text{otherwise} \end{cases} \qquad u = \begin{cases} u_1 & \text{if } u_1 \geq u_2 \\ +\infty & \text{otherwise} \end{cases}$$

Segni

Non-Distributive, Forward & Possible

Quella dei segni è un'analisi simile agli intervalli, ma semplificata, in cui le variabili possono assumere, nel dominio astratto, solo i valori positivo o negativo, con lo 0 incluso. Anche in questo caso dobbiamo definire le come le operazioni aritmetiche modificano il dominio.

Semantica

$$\text{Dominio: } \mathbb{S} = \{\top, \mathbb{Z}_0^+, \mathbb{Z}^+, \mathbb{Z}_0^-, \mathbb{Z}^-, 0, \neq 0, \bot\}$$

$$[\![;]\!]^\# S = S$$

$$[\![\text{zero}(e)]\!]^\# S = \begin{cases} \bot & [0,0] \not\sqsubseteq [\![e]\!]^\# S \\ S' & \textit{otherwise} \end{cases}$$

$$[\![\text{non-zero}(e)]\!]^\# S = \begin{cases} \bot & [0,0] = [\![e]\!]^\# S \\ S' & \textit{otherwise} \end{cases}$$

$$[\![M[e_1] \leftarrow e_2]\!]^\# S = S$$

$$[\![x \leftarrow e]\!]^\# S = S \oplus \{x \mapsto [\![e]\!]^\# S\}$$

$$[\![x \leftarrow M[e]]\!]^\# S = S \oplus \{x \mapsto \top\}$$

$$\text{where } S' = \textit{The sign evaluation of } [\![e]\!]^\# S$$

Esercizi

Available Expressions

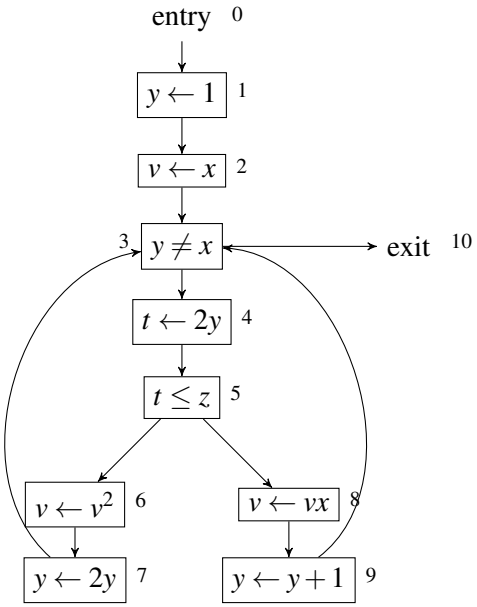
Codice

```
y ← 1;
v ← x;
while y ≠z
    t ← 2y;
    if t ≤z then
        v ← v^2;
        y ← 2y;
    else
        v ← vx;
        y ← y+1;
```

$\text{Ass} = \{y \leftarrow 1,\ v \leftarrow x,\ t \leftarrow 2y\}$

	Gen	Kill
1	$y \leftarrow 1$	$t \leftarrow 2y$
2	$v \leftarrow x$	$\emptyset$
3	$\emptyset$	$\emptyset$
4	$t \leftarrow 2y$	$\emptyset$
5	$\emptyset$	$\emptyset$
6	$\emptyset$	$v \leftarrow x$
7	$\emptyset$	$t \leftarrow 2y,\ y \leftarrow 1$
8	$\emptyset$	$v \leftarrow x$
9	$\emptyset$	$t \leftarrow 2y,\ y \leftarrow 1$
10	$\emptyset$	$\emptyset$

CFG - MOP



AvailIn	$\emptyset$	1	2
1	$\emptyset$	$\emptyset$	$\emptyset$
2	T	$y \leftarrow 1$	$y \leftarrow 1$
3	T	$v \leftarrow x,\ y \leftarrow 1$	$\emptyset$
4	T	$v \leftarrow x,\ y \leftarrow 1$	$\emptyset$
5	T	$t \leftarrow 2y,\ v \leftarrow x, y \leftarrow 1$	$t \leftarrow 2y$
6	T	$t \leftarrow 2y,\ v \leftarrow x, y \leftarrow 1$	$t \leftarrow 2y$
7	T	$t \leftarrow 2y,\ y \leftarrow 1$	$t \leftarrow 2y$
8	T	$t \leftarrow 2y,\ v \leftarrow x, y \leftarrow 1$	$t \leftarrow 2y$
9	T	$t \leftarrow 2y,\ y \leftarrow 1$	$t \leftarrow 2y$
10	T	$v \leftarrow x,\ y \leftarrow 1$	$\emptyset$

CFG - Semantics