

Analiza fluxului de date

4 mai 2004

- O sistematizare a analizelor de flux de date
- Analiza interprocedurale
- Aplicație: property simulation

Analiza programelor. Curs 2

Marius Minea

Proprietăți analizate (dataflow facts)

Concret: analizăm diverse proprietăți, de ex.

- valoarea unei variabile într-un punct de program
- sau *intervalul* de valori pentru o variabilă
- sau mulțimi de variabile (live), expresii (available, very busy), definiții posibile pentru o valoare (reaching definitions), etc.

Abstract: o mulțime D de valori pentru o proprietate (*dataflow facts*)
 Restricție: D e o mulțime *finită*

Analiza programelor. Curs 2

Marius Minea

Analiza fluxului de date

3

Mulțimi parțial ordonate

Concret:

– am asociat cu punctele de program *mulțimi* de valori pentru proprietatea analizată

– am recalculat iterativ mulțimile respective, prin operații de *reuniune* sau *intersecție*; obținând tot mai multe (sau mai puține) valori

Care sunt premisele esențiale pentru a efectua calculele în acest fel ?

Abstract: O *mulțime parțial ordonată* (L, \sqsubseteq) e o mulțime echipată cu o *relație de ordonare parțială* $\sqsubseteq \subseteq L \times L$, adică o relație:

- reflexivă, $x \sqsubseteq x$ pentru orice $x \in L$
- tranzitivă, $x \sqsubseteq y \wedge y \sqsubseteq z \Rightarrow x \sqsubseteq z$, pentru orice $x, y, z \in L$
- antisimetrică: $x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y$, pentru orice $x, y \in L$

Exemplu: mulțimea părților $(\mathcal{P}(D), \subseteq)$ sau $(\mathcal{P}(D), \supseteq)$

Analiza programelor. Curs 2

Marius Minea

Analiza fluxului de date

4

Latici

Lattice (completă) = o mulțime parțial ordonată în care orice submulțime finită are un cel mai mic majorant (least upper bound) și cel mai mare minorant (greatest upper bound).

l_0 e majorant al lui $Y \subseteq L$ dacă $\forall l \in Y$ avem $l \sqsubseteq l_0$

l_0 e minorant al lui $Y \subseteq L$ dacă $\forall l \in Y$ avem $l_0 \sqsubseteq l$

Notăm: $\sqcup Y$: cel mai mic majorant al mulțimii $Y \subseteq L$

$\sqcap Y$: cel mai mare minorant al mulțimii $Y \subseteq L$

și $\perp = \sqcup \emptyset = \sqcap L$ $\top = \sqcap \emptyset = \sqcup L$

Definim atunci operațiile

meet : $x \sqcap y = \sqcap \{x, y\}$

join : $x \sqcup y = \sqcup \{x, y\}$

(în cazul mulțimii părților: intersecție, reuniune)

Analiza programelor. Curs 2

Marius Minea

Analiza fluxului de date

5

Latici (cont.)

Operațiile \sqcap (*meet*) și \sqcup (*join*) au proprietăți:

- sunt comutative
- sunt asociative
- $x \sqcap \perp = \perp$ și $x \sqcup \top = \top$, pentru orice x .

Lattice *distributivă*: în care operatorii \sqcap și \sqcup sunt reciproc distributivi:

$x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$

$x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$

Analiza programelor. Curs 2

Marius Minea

Analiza fluxului de date

6

Funcții de transfer

Concret: instrucțiunile determină modificări ale stării programului. Valoarea unei variabile după o instrucțiune e o funcție a valorii de la începutul instrucțiunii.

Abstract: Fiecare instrucțiune s are asociată o funcție de transfer $F(s) : L \rightarrow L$ care determină modul în care valoarea proprietății la începutul instrucțiunii e modificată de instrucțiune:

$Prop_{out}(s) = F(s)(Prop_{in}(s))$ (pentru analize înainte), sau invers (pentru analize înapoi)

Restricție: punem condiția ca funcțiile de transfer să fie *monotone*:

$$x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$$

(dacă știm mai multe despre argument, atunci și despre rezultat)

Caz particular: *bitvector frameworks*: latticea e o mulțime de părți $\mathcal{P}(D)$, funcții de transfer monotone și de forma:

$$F(s)(v) = (v \setminus kill(s)) \sqcup gen(s)$$

(v = dataflow fact, $gen/kill(s)$ = informația generată/eliminată în s)

Analiza programelor. Curs 2

Marius Minea

Ecuatii de flux de date

Exemplu: pentru analize înainte:

$$Prop_{out}(s) = F(s)(Prop_{in}(s))$$

$$Prop_{in}(s) = \prod_{s' \in pred(s)} Prop_{out}(s')$$

unde prin \prod am reprezentat efectul combinării informațiilor (*meet*) pe mai multe căi (ar putea fi \cap sau \cup)

Inițial, e cunoscută valoarea $Prop_{out}(entry)$.

Pentru analize înapoi, se schimbă rolul între *in* și *out*, și e cunoscută valoarea lui $Prop_{in}(exit)$.

Terminare: condiția de punct fix

Terminarea analizei e garantată dacă funcția de transfer e monotonă: $x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$, ceea ce implică faptul că proprietățile calculate se modifică în mod monoton.

Def: **Punct fix** pentru o funcție f : o valoare x pt. care $f(x) = x$
Teorema lui Tarski garantează că o funcție monotonă pe o latice completă are un punct fix minimal și un punct fix maximal.

Algoritmul *worklist* calculează punctul fix minimal dat fiind sistemul de funcții de transfer.

Clasificare a analizelor

- înainte sau înapoi
- must sau may
- dependente sau independente de fluxul de control (flow (in)sensitive):
 Trebuie luată în considerare ordinea instrucțiunilor în program
 - nu: ce variabile sunt folosite/modificate, funcții apelate, etc.
 - da: proprietăți legate de valorile calculate efective de program
- dependente sau independente de context
 în cazul programelor ce conțin proceduri: e specializată analiza fiecărei proceduri în funcție de locul de apel, sau se poate face un sumar (o analiză comună) ?

Soluția: Algoritm de tip *worklist*

Pentru calculul soluției la sistemul de ecuații de mai sus: algoritmi iterativ care propagă modificările în sensul analizei

```

foreach  $s \in N$  do  $Prop_{in}(s) = \top$  /* no info */
 $Prop_{in}(entry) = init$  /* in funcție de analiza */
 $W = \{entry\}$ 
while  $W \neq \emptyset$ 
  choose  $s \in W$ 
   $W = W \setminus \{s\}$ 
   $Prop_{in}(s) = \prod_{s' \in pred(s)} Prop_{out}(s')$ 
   $Prop_{out}(s) = F(s)(Prop_{in}(s))$ 
  if change then
    forall  $s' \in succ(s)$  do  $W = W \cup \{s'\}$ 
    
```

Meet over all paths

Dorim să calculăm efectul combinat al instrucțiunilor programului: pentru $p = s_1 s_2 \dots s_n$ șir de instrucțiuni definim

$$F(p) = F(s_n) \circ \dots \circ F(s_2) \circ F(s_1)$$

și dorim să calculăm:

$$\prod_{p \in Path(Prog)} F_p(entry)$$

Dar algoritmul iterativ combină efectele la fiecare punct de întâlnire înainte de a calcula mai departe. Funcțiile f fiind monotone, avem:

$$f(x \sqcup y) \sqsupseteq f(x) \sqcup f(y)$$

deci analiza *pierde din precizie*

Pentru funcțiile de transfer *distributive* avem chiar: $f(x) \cup f(y) = f(x \cup y)$

Se demonstrează că în acest caz algoritmul iterativ (care generează o soluție de punct fix) e echivalent cu calculul soluției prin combinarea valorilor pe toate căile posibile (*meet over all paths*).

⇒ combinarea diverselor căi de execuție nu pierde informație

Cele 4 exemple date (live variables, etc.) sunt distributive.

Analize interprocedurale

- Modelarea programelor cu proceduri: în graful de flux de control, un
 - muchie de la locul de apel la începutul procedurii
 - muchie de la sfârșitul procedurii la instrucțiunea de după apel

În felul acesta, se obțin toate căile, dar și căi nefezabile
 ⇒ se restrânge analiza asupra căilor *realizabile*, în care apelurile și reîntoarcerile din funcții sunt împerecheate corect

Transformarea în problemă de parcurgere de grafuri

[Reps, Horwitz, Sagiv '95]

– pentru un subset de probleme de analiză de flux de date
(interprocedural, finite, distributive subset problem)

Property simulation

[Das, Lerner, Seigle '02 - Microsoft + U. Washington]

- o analiză statică (*property simulation*) scalabilă la $n \cdot 100$ kloc
- exemplu: absența de erori în > 600 apeluri de sistem pentru 15 pointeri de fișiere în codul `gcc` (140 kloc)
- prin analiză hibridă între o analiză standard de flux de data (imprecisă) și analiză dependentă de cale (path-sensitive, prea costisitoare)
- păstrând corelarea dintre starea proprietății analizate (ex. `uninit`, `open`, `closed` pentru fișier) și variabilele relevante din program)

```
if (dump)
    f = fopen(dumpFil, "w");
if (p) x = 0; else x = 1;
if (dump) fclose(f);
```