Analiza fluxului de date

4 mai 2004

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

Proprietăţi analizate (dataflow facts)

Concret: analizăm diverse proprietăţi, de ex.
– valoarea unei variabile într-un punct de program
– sau *intervalul* de valori pentru o variabilă
– sau mulţimi de variabile (live), expresii (available, very busy), definiţii posibile pentru o valoare (reaching definitions), etc.

Abstract: o mulţime D de valori pentru o proprietate (*dataflow facts*)
Restricţie: D e o mulţime *finită*

Mulţimi parţial ordonate

Concret:
– am asociat cu punctele de program *mulţimi* de valori pentru proprietatea analizată
– am recalculat iterativ mulţimile respective, prin operaţii de *reuniune* sau *intersecţie*; obţinând tot mai multe (sau mai puţine) valori

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

Abstract: O *mulţime parţial ordonată* $(L, \sqsubseteq)$ e o mulţime echipată cu o *relaţie de ordonare parţială* $\sqsubseteq \subseteq L \times L$, adică o relaţie:
– 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: mulţimea părţilor $(\mathcal{P}(D), \subseteq)$ sau $(\mathcal{P}(D), \supseteq)$

Latici

Latice (completă) = o multime parţial ordonată în care orice submulţime finită are un cel mai mic majorant (least upper bound) şi 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: $\bigsqcup Y$: cel mai mic majorant al mulţimii $Y \subseteq L$
$\bigsqcap Y$: cel mai mare minorant al mulţimii $Y \subseteq L$
şi $\bot = \bigsqcup \emptyset = \bigsqcap L \quad \top = \bigsqcap \emptyset = \bigsqcup L$

Definim atunci operaţiile
meet : $x \sqcap y = \bigsqcap \{x, y\}$
join : $x \sqcup y = \bigsqcup \{x, y\}$
(în cazul mulţimii părţilor: intersecţie, reuniune)

Latici (cont.)

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

– sunt comutative

– sunt asociative

– $x \sqcap \bot = \bot$ și $x \sqcup \top = \top$, pentru orice x.

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

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

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

Funcţii de transfer

Concret: instrucţiunile determină modificări ale stării programului. Valoarea unei variabile după o instrucţiune e o funcţie a valorii de la începutul instrucţiunii.

Abstract: Fiecare instrucţiune s are asociată o funcţie de transfer $F(s) : L \to L$ care determină modul în care valoarea proprietăţii la începutul instrucţiunii e modificată de instrucţiune:
$Prop_{out}(s) = F(s)(Prop_{in}(s))$ (pentru analize înainte),
sau invers (pentru analize înapoi)

Restricţie: punem condiţia ca funcţiile de transfer să fie *monotone*:

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

(dacă ştim mai multe despre argument, atunci şi despre rezultat)

Caz particular: *bitvector frameworks*: laticea e o mulţime de părţi $\mathcal{P}(D)$, funcţii de transfer monotone şi de forma:

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

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

Ecuații de flux de date

Exemplu: pentru analize înainte:

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

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

unde prin $\bigsqcap$ 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 $Prop_{in}(exit)$.

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 functie de analiza */
$W = \{entry\}$
while $W \neq \emptyset$
    choose $s \in W$
    $W = W \setminus \{s\}$
    $Prop_{in}(s) = \bigsqcap_{s' \in pred(s)} Prop_{out}(s')$
    $Prop_{out}(s) = F(s)(Prop_{in}(s))$
    if change **then**
        forall $s' \in succ(s)$ **do** $W = W \cup \{s'\}$

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.

Meet over all paths

Dorim să calculăm efectul combinat al instrucţiunilor programului: pentru $p = s_1 s_2 \ldots s_n$ şir de instrucţiuni definim

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

şi dorim să calculăm:

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

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

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

deci analiza *pierde din precizie*

Pentru funcţiile 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 soluţie de punct fix) e echivalent cu calculul soluţiei prin combinarea valorilor pe toate căile posibile (*meet over all paths*).

$\Rightarrow$ combinarea diverselor căi de execuţie nu pierde informaţie

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

Clasificare a analizelor

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

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 sfârşitul procedurii la instrucţiunea de după apel

În felul acesta, se obţin toate căile, dar şi căi nefezabile
$\Rightarrow$ se restrânge analiza asupra căilor *realizabile*, în care apelurile şi reîntoarcerile din funcţii 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 in $>$ 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);
```