Points-to analysis

Alias analysis

Points-to analysis

- Dynamická analýza (pouze myšlenkový experiment)
  - V daném okamžiku při běhu programu existuje množina živých proměnných a dynamicky alokovaných objektů
  - Součástí těchto proměnných a objektů jsou odkazy na jiné objekty (případně na jejich části nebo na proměnné)
  - Výstupem dynamické analýzy pro daný okamžik je orientovaný graf odkazů mezi proměnnými a objekty
    - Zkoumání tohoto grafu je principem garbage collection
  - Za běhu programu se situace mění
    - Přibývají/ubývají proměnné i dynamicky alokované objekty
    - Přesměrovávají se odkazy
  - Kompletním výsledkem dynamické analýzy by byly
    - Množina všech proměnných a objektů, které během běhu existovaly
      - Proměnné/objekty existující na témže místě považovány za různé
    - Orientovaný graf zaznamenávající odkazy mezi nimi
      - Doplněno „časovými razítky“ upřesňujícími trvání odkazu
  - *Taková analýza je reálně neproveditelná*

Points-to analysis

- Statická analýza
  - Potřeba integrovat výsledky pro všechny možné běhy programu
    - Tj. pro různá vstupní data etc.
  - Nutnost zmenšit výsledek analýzy na zvládnutelnou velikost
    - Nároky na výpočet i zpracování kompletního výsledku by převýšily užitečný běh programu
- Zmenšení objemu dat
  - Proměnné a objekty se sdruží do zvládnutelného počtu skupin
    - Faktorizace vrcholů dynamického grafu
  - Hrany statického points-to grafu mají existenční semantiku
    - Odkaz mezi skupinami znamená existenci odkazu mezi jejich prvky
  - Statická analýza počítá horní odhad dynamické analýzy
  - Časová razítka nahrazena pojmem kontext
    - Pozice v kódu, někdy včetně volajících procedur (stack trace)
    - Nerozlišuje iterace ve smyčkách

Varianty

- Kontextová analýza (flow-sensitive)
  - Points-to informace je relativní vůči pozici v programu
    - Semantika: Na pozici P v programu výraz E může označovat paměťové místo ze skupiny M
  - Pozice může a nemusí zahrnovat kontext, odkud byla procedura volána (call stack)
    - Úplná kontextová senzitivita je obvykle nerealizovatelná (různých kontextů by bylo příliš mnoho, v rekurzivních programech nekonečně)
    - Uvažuje se obvykle pouze začátek a/nebo konec zásobníku

- Bezkontextová analýza (flow-insensitive)
  - Points-to informace je společná pro celý program
    - Semantika: Existuje pozice v programu, kde výraz E může označovat paměťové místo ze skupiny M

Points-to analysis

- Dělení do skupin
  - Proměnné
    - Každá proměnná má obvykle svou skupinu
    - Ztotožňují se instance téže lokální proměnné v opakovaných/rekurzivních voláních procedury
  - Dynamicky alokované objekty
    - Skupiny definovány podle místa alokace objektu
    - Každý výskyt alokačního příkazu (new) definuje skupinu
      - Ve většině jazyků alokační příkaz staticky určuje typ objektu
    - Ztotožňují se opakovaná provedení téhož příkazu
      - V generickém kódu bývá nutné uvažovat kontext
    - Analyzátor musí chápat triky v implementaci např. u kontejnerů
      - Knihovny kontejnerů apod. musejí být pro analýzu anotovány

- **Překladač počítá horní odhad**
  - Sémantika: "může" = "překladač nebyl schopen vyloučit"
  - Všechny metody analýzy jsou iterativní (rekurzivní)
    - Zvolená přesnost reprezentace dramaticky ovlivňuje výsledek
    - Analýza s kontextem:

```
p = & x;        // p -> x
* p = & y;      // p -> x, x -> y
p = & u;        // p -> u, x -> y
* p = & v;      // p -> u, x -> y, u -> v
```

    - Bez kontextu:

```
p -> {x, u}
x -> {y, v}
u -> {y, v}
```

Points-to analysis

Andersenova metoda

- Bez kontextu

- $\boldsymbol{\sigma}(\boldsymbol{p})$ = množina (skupin) paměťových míst, na která může odkazovat proměnná (paměťové místo/skupina) $\boldsymbol{p}$
  - Tj. odchozí hrany vrcholu *p* v points-to grafu
- Na základě kódu se sestaví soustava množinových nerovnic (nebo pravidel v Datalogu):

`p = & x;`	$\{x\} \subseteq \boldsymbol{\sigma}(\boldsymbol{p})$	`E('p','x').`
`p = q;`	$\boldsymbol{\sigma}(\boldsymbol{q}) \subseteq \boldsymbol{\sigma}(\boldsymbol{p})$	`E('p',a) :- E('q',a).`
`p = * q;`	$(\forall \boldsymbol{b} \in \boldsymbol{\sigma}(\boldsymbol{q}))\ \boldsymbol{\sigma}(\boldsymbol{b}) \subseteq \boldsymbol{\sigma}(\boldsymbol{p})$	`E('p',a) :- E('q',b), E(b,a).`
`* p = q;`	$(\forall \boldsymbol{b} \in \boldsymbol{\sigma}(\boldsymbol{p}))\ \boldsymbol{\sigma}(\boldsymbol{q}) \subseteq \boldsymbol{\sigma}(\boldsymbol{b})$	`E(b,a) :- E('p',b), E('q',a).`

- Soustava se vyřeší metodou nejmenšího pevného bodu

Příklad - Andersenova metoda

- Zdrojový kód a odpovídající pravidla Datalogu:

```
p = & x;        // E('p','x').
* p = & y;      // E(b,'y') :- E('p',b).
p = & u;        // E('p','u').
* p = & v;      // E(b,'v') :- E('p',b).
```

- Výsledek (minimální model výše uvedených pravidel Datalogu):

```
E('p','x')
E('p','u')
E('x','y')
E('x','v')
E('u','y')
E('u','v')
```

Points-to analysis

- Realita je složitější:
  - Ukazatelé ukazují na objekty, objekty obsahují položky, položky mohou být ukazatelé
    - Dále komplikováno dědičností
  - Originální metoda odpovídá sloučení položek
    - "Objekt odkazuje" znamená "některá z položek odkazuje"
    - Statický ekvivalent postupu garbage collectoru
    - Nerozlišování položek vede k příliš pesimistickému odhadu
  - Vylepšená metoda zahrnuje jména položek:
    - $\boldsymbol{\sigma}(\boldsymbol{p}, \boldsymbol{m})$ = množina míst, na která může odkazovat položka m objektu p
    - V Datalogové verzi zapisováno

      `E('p','.m','a')`
    - Odpovídající situace v jazyce C:

      `p.m == &a`
    - Stále neumíme přesně modelovat ukazatele na části objektů:

      `q == &b.m` `E('q','','b')`

Realističtější příklad - Andersenova metoda

- Zdrojový kód a odpovídající pravidla Datalogu:

```
void f(Node *p, Node *q) {
  q->n = p->n; // E(x,'.n',z) :- E('q',x), E('p',y), E(y,'.n',z).
  p->n = q;    // E(x,'.n',z) :- E('p',x), E('q',z).
}
Node r;
r.n = NULL;                        // E('r','.n',0).
f(&r,malloc(sizeof(Node)));        // E('p','r'). E('q','malloc1').
```

- Výsledek (minimální model výše uvedených pravidel Datalogu):

```
E('p','r').
E('q','malloc1').
E('r','.n',0).
E('r','.n','malloc1').
E('malloc1','.n',0).
E('malloc1','.n','malloc1').
```

- Poslední hrana je důsledkem bezkontextovosti
  - metoda nerozlišuje stavy před a po volání f
  - odpovídá opakovanému volání f

Points-to analysis

- **Steensgaardova metoda (1996)**
  - Bez kontextu
  - Méně přesná, ale rychlejší než Andersenova metoda

- $\tau(p)$ = "typ" proměnné (paměťového místa/skupiny) p
  - definován rekurzivně, na základě typu, na který může odkazovat
  - $\tau(\tau(p))$ - "typ" výrazu *p, atd.
- Začínáme s předpokladem, že $\tau(p)$ jsou navzájem různé
- Přiřazení nalezená v kódu způsobují ztotožnění typů
- Ztotožnění typů se propagují směrem k odkazovaným typům

Realističtější příklad - Steensgaardova metoda

- Zdrojový kód a odpovídající Steensgaardovy ekvivalence:

```
void f(Node *p, Node *q) {
  q->n = p->n; // T(T(q),'.n') == T(T(p),'.n')
  p->n = q;    // T(T(p),'.n') == T(q)
}
Node r;
r.n = NULL;                        // T(r,'.n') == T(0)
f(&r,malloc(sizeof(Node)));   // T(T(p)) == T(r), T(T(q)) == T(m1)
```

- Pravidlo pro manipulaci s položkami:

```
T(x) == T(y) => T(x,'.n') == T(x,'.n')
```

- Výsledkem jsou tyto třídy ekvivalence:

```
{ T(r), T(T(p)) }
```

  - odpovídá objektu r

```
{ T(m1), T(T(q)), T(T(T(p),'.n')), T(T(r,'.n')), ... }
```

  - odpovídá všem dynamicky alokovaným objektům

```
{ T(0), T(q), T(r,'.n'), T(m1,'.n'), T(T(p),'.n'), T(T(q),'.n'), ... }
```

  - odpovídá všem ukazatelům, vyjma p

```
{ T(p) }
```