Překladače (7. přednáška)

Jan Hubička

Katedra aplikované matematiky
Univerzita Karlova
Praha

27. dubna 2020

Dominance

Definice (Prosser 1959)

Basic blok *a* dominuje basic blok *b* pokud na každé cestě z Entry do *b* je *a*.

Pozorování (tranzitivita)

Pokud *a* dominuje *b* and *b* dominuje *c* potom *a* dominuje *c*.

Důsledek: dominance je částečné uspořádání.

Pozorování

Pokud *a* i *b* dominují *c*, potom *a* dominuje *b* nebo *b* dominuje *a*

Důsledek: dominance tvoří strom zakořeněný v Entry.
Důsledek: dominance má kompaktní stromovou reprezentaci pomocí preorderu a postorderu.

Single static assignment

Definitice (SSA)

Program je v SSA formě pokud každá proměnná je přiřazena právě jednou.

Pokud nechceme používat nedefinované proměnné, každá definice musí dominovat použití.

Oprace ϕ

Operace ϕ se používá jen na začátku basic bloku. Má tolik parametrů kolik je vstupních hran CFG. Hodnota operace ϕ je parametr odpovídající hraně po které program do basic bloku přišel.

Efektivní konstrukce SSA

Konstrukce SSA (Cytron, Ferrante, Rosen, Wegman, and Zadeck)

Převod do SSA formy:

1. Rozmístění ϕ operací
2. Přejmenování proměnných na SSA jména

Kam musíme vložit ϕ?

Join sets

Pro danou množinu $\mathcal{S}$ basic bloků join set $J(\mathcal{S})$ je množina všech basic bloků b takových že existují dvě cesty se začátky v S konvergující v b.

Iterated join set

$$J_1 = J(\mathcal{S})$$

$$J_{i+1} = J(\mathcal{S} \cup J_i)$$

$J^+(\mathcal{S})$ je sjednocení posloupnosti $J_1, J_2, \ldots$.

Kam musíme vložit ϕ?

Join sets

Pro danou množinu $\mathcal{S}$ basic bloků join set $J(\mathcal{S})$ je množina všech basic bloků b takových že existují dvě cesty se začátky v S konvergující v b.

Iterated join set

$$J_1 = J(\mathcal{S})$$

$$J_{i+1} = J(\mathcal{S} \cup J_i)$$

$J^+(\mathcal{S})$ je sjednocení posloupnosti $J_1, J_2, \ldots$.

definice (Dominance frontier)

$$\mathrm{DF}(a) = \{b|_{\exists p \in \mathrm{Preds}(b)} : a \text{ dom } p \;\&\; \neg a \text{ sdom } b)\}$$

Iterated dominance frontier

$$\mathrm{DF}_1 = \mathrm{DF}(\mathcal{S})$$

$$\mathrm{DF}_{i+1} = \mathrm{DF}(\mathcal{S} \cup \mathrm{DF}_i)$$

$\mathrm{DF}^+(\mathcal{S})$ je sjednocení posloupnosti $\mathrm{DF}_1, \mathrm{DF}_2, \ldots$.

Kam musíme vložit ϕ?

Join sets

Pro danou množinu $\mathcal{S}$ basic bloků join set $J(\mathcal{S})$ je množina všech basic bloků b takových že existují dvě cesty se začátky v S konvergující v b.

Iterated join set

$$J_1 = J(\mathcal{S})$$

$$J_{i+1} = J(\mathcal{S} \cup J_i)$$

$J^+(\mathcal{S})$ je sjednocení posloupnosti $J_1, J_2, \ldots$.

definice (Dominance frontier)

$$\mathrm{DF}(a) = \{b|_{\exists p \in \mathrm{Preds}(b)} : a \text{ dom } p \ \& \ \neg a \text{ sdom } b)\}$$

Iterated dominance frontier

$$\mathrm{DF}_1 = \mathrm{DF}(\mathcal{S})$$

$$\mathrm{DF}_{i+1} = \mathrm{DF}(\mathcal{S} \cup \mathrm{DF}_i)$$

$\mathrm{DF}^+(\mathcal{S})$ je sjednocení posloupnosti $\mathrm{DF}_1, \mathrm{DF}_2, \ldots$.

Věta (Cytron, Ferrante, Rosen, Wegman, and Zadeck, 1991)

Množina bloků, které potřebují ϕ dané proměnné v je $\mathrm{DF}(\mathcal{S})$ kde $\mathcal{S}$ jsou všeschny bloky nastavující v a Entry.

Lemma

každá cesta $p : X \to Z$ obsahuje vrchol $Y \in \mathrm{DF}^+(X)$ domunující Z.

Lemma

každá cesta $p : X \to Z$ obsahuje vrchol $Y \in \mathrm{DF}^+(X)$ domunující Z.

Lemma

Pokud dvě cesty $p : X \to Z$ a $q : Y \to Z$ pro $X \neq Y$ konvergují v basic blocku Z, potom $Z \in DF^+(X) \cup DF^+(Y)$.

Lemma

každá cesta $p : X \to Z$ obsahuje vrchol $Y \in \mathrm{DF}^+(X)$ domunující Z.

Lemma

Pokud dvě cesty $p : X \to Z$ a $q : Y \to Z$ pro $X \neq Y$ konvergují v basic blocku Z, potom $Z \in DF^+(X) \cup DF^+(Y)$.

Důsledek: $J(\mathcal{S}) \subseteq \mathrm{DF}^+(\mathcal{S})$

Lemma

každá cesta $p : X \to Z$ obsahuje vrchol $Y \in \mathrm{DF}^+(X)$ domunující Z.

Lemma

Pokud dvě cesty $p : X \to Z$ a $q : Y \to Z$ pro $X \neq Y$ konvergují v basic blocku Z, potom $Z \in DF^+(X) \cup DF^+(Y)$.

Důsledek: $J(\mathcal{S}) \subseteq \mathrm{DF}^+(\mathcal{S})$

Lemma

Pokud $\mathcal{S}$ obsahuje Entry, potom $\mathrm{DF}(\mathcal{S}) \subseteq J(\mathcal{S})$.

Lemma

každá cesta $p : X \to Z$ obsahuje vrchol $Y \in \mathrm{DF}^+(X)$ domunující Z.

Lemma

Pokud dvě cesty $p : X \to Z$ a $q : Y \to Z$ pro $X \neq Y$ konvergují v basic blocku Z, potom $Z \in DF^+(X) \cup DF^+(Y)$.

Důsledek: $J(\mathcal{S}) \subseteq \mathrm{DF}^+(\mathcal{S})$

Lemma

Pokud $\mathcal{S}$ obsahuje Entry, potom $\mathrm{DF}(\mathcal{S}) \subseteq J(\mathcal{S})$.

Věta (Cytron, Ferrante, Rosen, Wegman, and Zadeck, 1991)

Množina bloků, které potřebují ϕ dané proměnné v je $\mathrm{DF}(\mathcal{S})$ kde $\mathcal{S}$ jsou všeschny bloky nastavující v a Entry.

Lemma

každá cesta $p : X \to Z$ obsahuje vrchol $Y \in \mathrm{DF}^+(X)$ domunující Z.

Lemma

Pokud dvě cesty $p : X \to Z$ a $q : Y \to Z$ pro $X \neq Y$ konvergují v basic blocku Z, potom $Z \in DF^+(X) \cup DF^+(Y)$.

Důsledek: $J(\mathcal{S}) \subseteq \mathrm{DF}^+(\mathcal{S})$

Lemma

Pokud $\mathcal{S}$ obsahuje Entry, potom $\mathrm{DF}(\mathcal{S}) \subseteq J(\mathcal{S})$.

Věta (Cytron, Ferrante, Rosen, Wegman, and Zadeck, 1991)

Množina bloků, které potřebují ϕ dané proměnné v je $\mathrm{DF}(\mathcal{S})$ kde $\mathcal{S}$ jsou všeschny bloky nastavující v a Entry.

Časová a paměťová složitost

Výsledkem je minimální SSA forma. Ještě menší SSA forma: prunned SSA.

Časová a paměťová složitost

Výsledkem je minimální SSA forma. Ještě menší SSA forma: prunned SSA.

Věta

Pokud control flow obsahuje pouze podmínky a smyčky, potom DF(B) obsahuje max. 2 bloky

Časová a paměťová složitost

Výsledkem je minimální SSA forma. Ještě menší SSA forma: prunned SSA.

Věta

Pokud control flow obsahuje pouze podmínky a smyčky, potom DF(B) obsahuje max. 2 bloky

Matthias Braun, Sebastian Buchwald, Sebastian Hack, Roland Leißa,Christoph Mallon, and Andreas Zwinkau: Simple and Efficient Construction of Static SingleAssignment Form.

Jednoduchý algoritmus založený na value numbering, který postaví SSA formu rovnou při expanzi AST do mezijazyka a ušetří tím dost paměti. Postaví prunned SSA pro reducibilní CFG.

Destrukce SSA

1.
2. Vkládání kopií

Destrukce SSA

1. Coalescing: výpočet liveness, sjednocování pomocí union-find.
2. Vkládání kopií

1. Globální propagace konstant, rozsahů hodnot...

1. Globální propagace konstant, rozsahů hodnot...
2. Globální eliminace společných podvýrazů

1. Globální propagace konstant, rozsahů hodnot...
2. Globální eliminace společných podvýrazů
3. Mazání mrtvého kódu

1. Globální propagace konstant, rozsahů hodnot...
2. Globální eliminace společných podvýrazů
3. Mazání mrtvého kódu

Optimalizace snadno udržují SSA formu. Výsledkem ale mohou být redundantní PHI.

SCCP: Sparse conditional constant propagation: Wegman, Zadeck, 1991

Propagační engine

1. SSAwl $\leftarrow \emptyset$, CFGwl $\leftarrow \emptyset$.
2. Pro každý basic blok B: exec(B) $=$ false
3. Dokud SSAwl $\cup$ CFGwl $\neq \emptyset$
4. Dokud CFGwl $\neq \emptyset$
5. Vyzvedni B z CFGwl, exec(B) $\leftarrow$ true
6. Vyhodnoť každé ϕ a každý výraz v B
7. Pokud B ma jen jednoho potomka, přidej ho do CFGwl.
8. Dokud SSAwl $\neq \emptyset$
9. Vyzvedni s z SSA*wl* a vyhodnoť definici s do t.
10. Pokud $t \neq$ value(n)
11. Přidej do SSAwl všechny použití p hodnoty s takové, že exec(bb(p)) $=$ true

SCCP: Sparse conditional constant propagation: Wegman, Zadeck, 1991

Propagační engine

1. SSAwl $\leftarrow \emptyset$, CFGwl $\leftarrow \emptyset$.
2. Pro každý basic blok B: exec(B) $=$ false
3. Dokud SSAwl $\cup$ CFGwl $\neq \emptyset$
4. Dokud CFGwl $\neq \emptyset$
5. Vyzvedni B z CFGwl, exec(B) $\leftarrow$ true
6. Vyhodnoť každé ϕ a každý výraz v B
7. Pokud B ma jen jednoho potomka, přidej ho do CFGwl.
8. Dokud SSAwl $\neq \emptyset$
9. Vyzvedni s z SSA*wl* a vyhodnoť definici s do t.
10. Pokud $t \neq$ value(n)
11. Přidej do SSAwl všechny použití p hodnoty s takové, že exec(bb(p)) $=$ true

EvalPhi(s)

Uvažuje jen ty parametry jejiž bloky mají nastavený exec.

SCCP: Sparse conditional constant propagation: Wegman, Zadeck, 1991

Propagační engine

1. SSAwl $\leftarrow \emptyset$, CFGwl $\leftarrow \emptyset$.
2. Pro každý basic blok B: exec(B) $=$ false
3. Dokud SSAwl $\cup$ CFGwl $\neq \emptyset$
4. Dokud CFGwl $\neq \emptyset$
5. Vyzvedni B z CFGwl, exec(B) $\leftarrow$ true
6. Vyhodnoť každé ϕ a každý výraz v B
7. Pokud B ma jen jednoho potomka, přidej ho do CFGwl.
8. Dokud SSAwl $\neq \emptyset$
9. Vyzvedni s z SSA*wl* a vyhodnoť definici s do t.
10. Pokud $t \neq$ value(n)
11. Přidej do SSAwl všechny použití p hodnoty s takové, že exec(bb(p)) $=$ true

EvalPhi(s)

Uvažuje jen ty parametry jejiž bloky mají nastavený exec.

EvalStmt(s)

Po vyhodnocení podmínky na známou konstatu nebo $\perp$ přidá odpovídající bloky do CFGwl.