Překladače (9. přednáška)

Jan Hubička

Katedra aplikované matematiky
Univerzita Karlova
Praha

11. května 2020

Value numbering

Value numbering je analýza, která rozdělí výrazy do tříd ekvivalence. Dva výrazy je vstejné třídě ekvivalence mají vždy stejnou hodnotu.

Local value numbering

Killdall: A Unified Approach to Global Program Optimization (1973)

Unification based: Alpern, Wegman, Zadeck

Alpern, Wegman, Zadeck: Detecting Equality of Variables in Programs (1988)

RPO algorithm

```
for all SSA names i
    VN[i] ← ⊤
repeat
    done ← TRUE
    for all blocks b in reverse postorder
        for all definitions x in b
            temp ← lookup(x.op, VN[x[1]], VN[x[2]], x)
            if VN[x] ≠ temp
                done ← FALSE
                VN[x] ← temp
    Remove all entries from the hash table
until done
```

RPO algorithm

Algoritmus počítá sekvenci ekvivalencí $\simeq_1, \simeq_2, \ldots$ na výrazech.

Pozorování

Každé iterace zjemní dělení: $x \simeq_i y \implies x \simeq_{i-1} y$

RPO algorithm

Algoritmus počítá sekvenci ekvivalencí $\simeq_1, \simeq_2, \ldots$ na výrazech.

Pozorování

Každé iterace zjemní dělení: $x \simeq_i y \implies x \simeq_{i-1} y$

Pozorování

Algoritmus je konečný a vydá maximální fixpoint.

RPO algorithm

Algoritmus počítá sekvenci ekvivalencí $\simeq_1, \simeq_2, \ldots$ na výrazech.

Pozorování

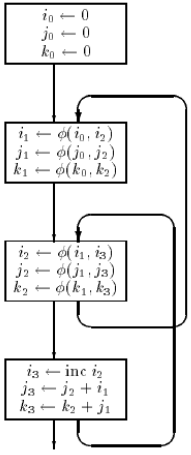
Každé iterace zjemní dělení: $x \simeq_i y \implies x \simeq_{i-1} y$

Pozorování

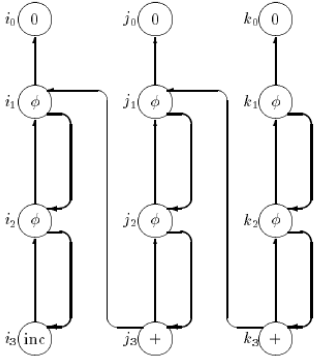
Algoritmus je konečný a vydá maximální fixpoint.

Věta

Algoritmus konverguje po $D(SSA) + 2$ průchodech.



Control-flow graph



SSA graph

SCC GVN

Cooper, Simpson: SCC-based value numbering

- Dvě tabulky: optimistická a konečná
- Pomocí Tarjanova algoritmu hledáme KSS v SSA grafu
- Při opouštění komponenty spustíme RPO agoritmus na dané komponentě pomocí optimistické tabulky. Výsledky překopírujeme do konečné.

Value inference

Predicate inference

Φ predikace

GVN PRE

Thomas John Van Drunen: Partial Redundancy Elimination for Global Value Numbering (2004)

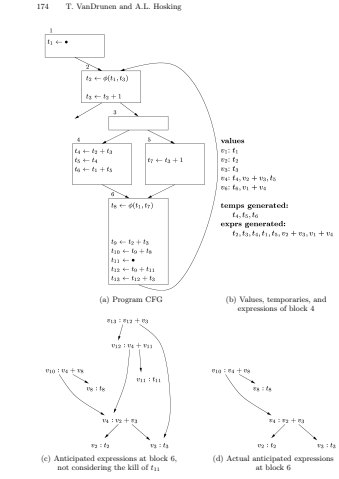


Fig. 3. Running example

GVN PRE

$$\mathsf{AVAIL\_IN}[b] = \mathsf{AVAIL\_OUT}[\mathsf{dom}(b)]$$

$$\mathsf{AVAIL\_OUT}[b] = \mathsf{canon}(\mathsf{AVAIL\_IN}[b] \cup \mathsf{PHI\_GEN}(b) \cup \mathsf{TMP\_GEN}(b))$$

GVN PRE

$$\mathsf{AVAIL\_IN}[b] = \mathsf{AVAIL\_OUT}[\mathsf{dom}(b)]$$

$$\mathsf{AVAIL\_OUT}[b] = \mathsf{canon}(\mathsf{AVAIL\_IN}[b] \cup \mathsf{PHI\_GEN}(b) \cup \mathsf{TMP\_GEN}(b))$$

$$\mathsf{ANTIC\_OUT}[b] = \begin{cases} \{e | e \in \mathsf{ANTIC\_IN}[\mathsf{succ}_0(b)] \wedge \\ \quad \forall\ b' \in \mathsf{succ}(b), \exists e' \in \mathsf{ANTIC\_IN}[b'] | \\ \quad \mathsf{lookup}(e) = \mathsf{lookup}(e')\} & \text{if } |\mathsf{succ}(b)| > 1 \\ \\ \mathsf{phi\_translate}(A[\mathsf{succ}(b)], b, \mathsf{succ}(b)) & \text{if } |\mathsf{succ}(b)| = 1 \end{cases}$$

$$\mathsf{ANTIC\_IN}[b] = \mathsf{clean}(\mathsf{canon}_e(\mathsf{ANTIC\_OUT}[b] \cup \mathsf{EXP\_GEN}[b] - \mathsf{TMP\_GEN}(b)))$$

GVN PRE – algoritmus

Insert: pro bloky s více než jedním předchůdcem prohledni všechny anticipované výrazy. Pokud jsou dostupné jen v některých předchůdcích, přidej výpočty na hrany, kde nejsou.

GVN PRE – algoritmus

Insert: pro bloky s více než jedním předchůdcem prohledni všechny anticipované výrazy. Pokud jsou dostupné jen v některých předchůdcích, přidej výpočty na hrany, kde nejsou.
Delete: Smaž všechny výpočty které jsou v bodě výpočtu dostupné.