Překladače (6. přednáška)

Jan Hubička

Katedra aplikované matematiky
Univerzita Karlova
Praha

20. dubna 2020

Dominance

Definice (Prosser 1959)

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

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*.

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.

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

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.

Dominance jako dataflow

Dominance dataflow

$$\mathrm{Dom}(\mathrm{Entry}) = \{\mathrm{Entry}\}$$

$$\mathrm{Dom}(b) = \left(\bigcup_{p \in \mathrm{preds}(b)} \mathrm{Dom}(p) \right) \cup \{b\}$$

Dominance jako dataflow

Dominance dataflow

$$\mathrm{Dom}(\mathrm{Entry}) = \{\mathrm{Entry}\}$$

$$\mathrm{Dom}(b) = \left(\bigcup_{p \in \mathrm{preds}(b)} \mathrm{Dom}(p) \right) \cup \{b\}$$

Počet průchodů v reverse postorder: $d(\mathrm{CFG}) + 3$.

Lepší datové struktury (Cooper, Harvey, Kennedy 2001)

Pozorování

Pro $b \neq$Entry platí

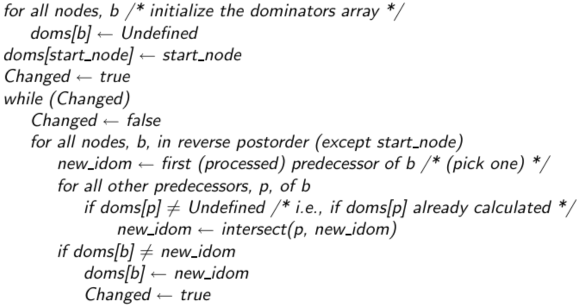
$$\mathrm{Dom}(b) = \{b\} \cup \mathrm{IDom}(b) \cup \mathrm{IDom}(\mathrm{IDom}(b)) \cup \cdots \cup \{\mathrm{Entry}\}$$

Lepší datové struktury (Cooper, Harvey, Kennedy 2001)

Pozorování

Pro $b \neq$Entry platí

$$\mathrm{Dom}(b) = \{b\} \cup \mathrm{IDom}(b) \cup \mathrm{IDom}(\mathrm{IDom}(b)) \cup \cdots \cup \{\mathrm{Entry}\}$$



```
for all nodes, b /* initialize the dominators array */
    doms[b] ← Undefined
doms[start_node] ← start_node
Changed ← true
while (Changed)
    Changed ← false
    for all nodes, b, in reverse postorder (except start_node)
        new_idom ← first (processed) predecessor of b /* (pick one) */
        for all other predecessors, p, of b
            if doms[p] ≠ Undefined /* i.e., if doms[p] already calculated */
                new_idom ← intersect(p, new_idom)
        if doms[b] ≠ new_idom
            doms[b] ← new_idom
            Changed ← true
```

Lepší datové struktury (Cooper, Harvey, Kennedy 2001)

Pozorování

Pro $b \neq$Entry platí

$$\mathrm{Dom}(b) = \{b\} \cup \mathrm{IDom}(b) \cup \mathrm{IDom}(\mathrm{IDom}(b)) \cup \cdots \cup \{\mathrm{Entry}\}$$

```
for all nodes, b /* initialize the dominators array */
    doms[b] ← Undefined
doms[start_node] ← start_node
Changed ← true
while (Changed)
    Changed ← false
    for all nodes, b, in reverse postorder (except start_node)
        new_idom ← first (processed) predecessor of b /* (pick one) */
        for all other predecessors, p, of b
            if doms[p] ≠ Undefined /* i.e., if doms[p] already calculated */
                new_idom ← intersect(p, new_idom)
        if doms[b] ≠ new_idom
            doms[b] ← new_idom
            Changed ← true
```

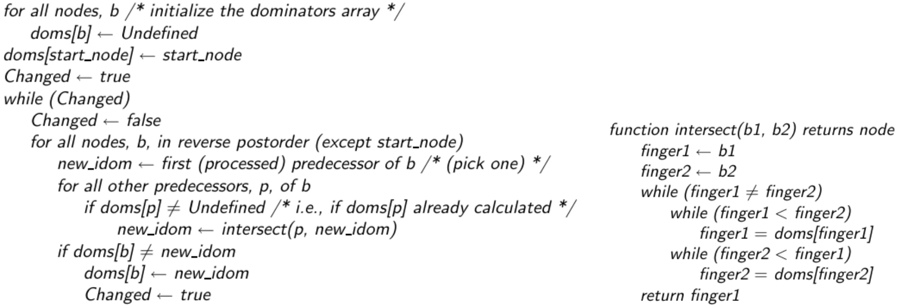
```
function intersect(b1, b2) returns node
    finger1 ← b1
    finger2 ← b2
    while (finger1 ≠ finger2)
        while (finger1 < finger2)
            finger1 = doms[finger1]
        while (finger2 < finger1)
            finger2 = doms[finger2]
    return finger1
```

Lepší datové struktury (Cooper, Harvey, Kennedy 2001)

Pozorování

Pro $b \neq$Entry platí

$$\text{Dom}(b) = \{b\} \cup \text{IDom}(b) \cup \text{IDom}(\text{IDom}(b)) \cup \cdots \cup \{\text{Entry}\}$$



```
for all nodes, b /* initialize the dominators array */
    doms[b] ← Undefined
doms[start_node] ← start_node
Changed ← true
while (Changed)
    Changed ← false
    for all nodes, b, in reverse postorder (except start_node)
        new_idom ← first (processed) predecessor of b /* (pick one) */
        for all other predecessors, p, of b
            if doms[p] ≠ Undefined /* i.e., if doms[p] already calculated */
                new_idom ← intersect(p, new_idom)
        if doms[b] ≠ new_idom
            doms[b] ← new_idom
            Changed ← true
```

```
function intersect(b1, b2) returns node
    finger1 ← b1
    finger2 ← b2
    while (finger1 ≠ finger2)
        while (finger1 < finger2)
            finger1 = doms[finger1]
        while (finger2 < finger1)
            finger2 = doms[finger2]
    return finger1
```

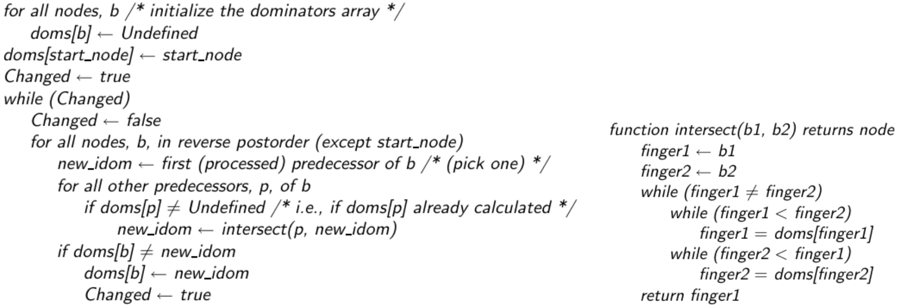
Časová složitost na iteraci: $O(N + E \times D)$ kde D je největší množina.

Lepší datové struktury (Cooper, Harvey, Kennedy 2001)

Pozorování

Pro $b \neq$Entry platí

$$\mathrm{Dom}(b) = \{b\} \cup \mathrm{IDom}(b) \cup \mathrm{IDom}(\mathrm{IDom}(b)) \cup \cdots \cup \{\mathrm{Entry}\}$$



```
for all nodes, b /* initialize the dominators array */
    doms[b] ← Undefined
doms[start_node] ← start_node
Changed ← true
while (Changed)
    Changed ← false
    for all nodes, b, in reverse postorder (except start_node)
        new_idom ← first (processed) predecessor of b /* (pick one) */
        for all other predecessors, p, of b
            if doms[p] ≠ Undefined /* i.e., if doms[p] already calculated */
                new_idom ← intersect(p, new_idom)
        if doms[b] ≠ new_idom
            doms[b] ← new_idom
            Changed ← true
```

```
function intersect(b1, b2) returns node
    finger1 ← b1
    finger2 ← b2
    while (finger1 ≠ finger2)
        while (finger1 < finger2)
            finger1 = doms[finger1]
        while (finger2 < finger1)
            finger2 = doms[finger2]
    return finger1
```

Časová složitost na iteraci: $O(N + E \times D)$ kde D je největší množina.
Lengauer-Tarjan: praktický algoritmus založený na union find běžící v čase $O(E \log N)$.

Single static assignment

Definitice (SSA)

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

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

První přiblížení ke vkládání ϕ: hranice dominance

definice (Dominance frontier)

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

První přiblížení ke vkládání ϕ: hranice dominance

definice (Dominance frontier)

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

Rozdělime $\mathrm{DF}(a)$ na $\mathrm{DF}_{\mathrm{local}}(a)$ a $\mathrm{DF}_{\mathrm{up}}(a)$:

$$\mathrm{DF}(a) = \mathrm{DF}_{\mathrm{local}}(a) \cup \bigcup_{s \in \mathrm{sons}(a)} \mathrm{DF}_{\mathrm{up}}(s)$$

První přiblížení ke vkládání ϕ: hranice dominance

definice (Dominance frontier)

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

Rozdělime $\mathrm{DF}(a)$ na $\mathrm{DF}_{\mathrm{local}}(a)$ a $\mathrm{DF}_{\mathrm{up}}(a)$:

$$\mathrm{DF}(a) = \mathrm{DF}_{\mathrm{local}}(a) \cup \bigcup_{s \in \mathrm{sons}(a)} \mathrm{DF}_{\mathrm{up}}(s)$$

$$\mathrm{DF}_{\mathrm{local}}(a) = \{b \in \mathrm{succs}(a) | \neg a \text{ dom } b\}$$

První přiblížení ke vkládání ϕ: hranice dominance

definice (Dominance frontier)

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

Rozdělíme $\mathrm{DF}(a)$ na $\mathrm{DF}_{\mathrm{local}}(a)$ a $\mathrm{DF}_{\mathrm{up}}(a)$:

$$\mathrm{DF}(a) = \mathrm{DF}_{\mathrm{local}}(a) \cup \bigcup_{s \in \mathrm{sons}(a)} \mathrm{DF}_{\mathrm{up}}(s)$$

$$\mathrm{DF}_{\mathrm{local}}(a) = \{b \in \mathrm{succs}(a) | \neg a \text{ dom } b\}$$

$$\mathrm{DF}_{\mathrm{up}}(c) = \{d \in \mathrm{DF}(c) | \neg \mathrm{idom}(c) \text{ dom } d\}$$

Kam musíme vložit ϕ?

Cesty $p : x_0, x_1, \ldots x_j$ a $q : y_0, y_1, \ldots y_k$ konvergují v bloku z pokud

$$x_0 \neq y_0$$

$$z_j = y_k = z$$

$$(x_{j'} = y_{k'} \implies (j' = j \text{ nebo } k' = k)$$

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.

Kam musíme vložit ϕ?

Cesty $p : x_0, x_1, \ldots x_j$ a $q : y_0, y_1, \ldots y_k$ konvergují v bloku z pokud

$$x_0 \neq y_0$$

$$z_j = y_k = z$$

$$(x_{j'} = y_{k'} \implies (j' = j \text{ nebo } k' = k)$$

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 sjednoceni posloupnosti $J_1, J_2, \ldots$.

Kam musíme vložit ϕ?

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 sjednoceni posloupnosti $\mathrm{DF}_1, \mathrm{DF}_2, \ldots$.

Věta

$J^+(\mathcal{S}) = \mathrm{DF}^+(\mathcal{S})$