Compilerkonstruktion

Wintersemester 2015/16

Prof. Dr. R. Parchmann

19. Januar 2016

Analyse der verfügbaren Ausdrücke (Available Expressions)

Ein Ausdruck `b op c` ist an einem Punkt p im Programm verfügbar, wenn auf jedem Weg von Eingangsknoten *entry* zum Punkt p dieser Ausdruck ausgewertet wird und danach keine Zuweisungen zu `b` oder `c` passieren.
Diese Information ist wichtig, um gemeinsame Teilausdrücke über Blockgrenzen hinaus zu erkennen.
Sei U die Menge aller Ausdrücke, die in Drei-Adress-Befehlen des Programms auftreten.

Transferfunktion für einen Drei-Adress-Befehl

Sei s ein Befehl der Form `a := b op c`. Offensichtlich wird an dieser Stelle der Ausdruck `b op c` erzeugt und jeder Ausdruck aus U, der `a` enthält, gelöscht.
Also gilt $\texttt{e\_gen}[s] = \{\texttt{b op c}\}$ und
$\texttt{e\_kill}[s] = \{\text{Ausdrücke aus } U, \text{ die a enthalten}\}$ und die Transferfunktion wäre

$$f_s(x) = (\texttt{e\_gen}[s] \cup x) - \texttt{e\_kill}[s].$$

Analog folgt für andere Formen von Drei-Adress-Befehlen:

Befehlstyp s	`e_gen`$[s]$	`e_kill`$[s]$
`a := b+c`	$\{$`b+c`$\}$	alle Ausdrücke aus U, die `a` enthalten
`a := b[c]`	$\{$`b[c]`$\}$	alle Ausdrücke aus U, die `a` enthalten
`if a relOp b goto ...`	$\{\}$	$\{\}$
`a[b] := c`	$\{\}$	Ausdrücke aus U der Form `a[x]`
usw.		

Transferfunktion für einen einfachen Block

Man erhält

$$f_B(x) = \texttt{e\_gen}[B] \cup (x - \texttt{e\_kill}[B])$$

wobei

$$\texttt{kill}[B] = \texttt{e\_kill}[1] \cup \texttt{e\_kill}[2] \cup \ldots \cup \texttt{e\_kill}[k]$$

und

$$\begin{aligned} \texttt{e\_gen}[B] \quad = \quad & (\texttt{e\_gen}[k] - \texttt{e\_kill}[k]) \cup \\ & (\texttt{gen}[k-1] - \texttt{e\_kill}[k-1] - \texttt{e\_kill}[k]) \cup \\ & \ldots \cup (\texttt{gen}[1] - \texttt{e\_kill}[1] - \cdots - \texttt{e\_kill}[k]) \end{aligned}$$

Algorithmus zur Bestimmung der e_gen- und e_kill-Menge eines Blocks

Eingabe: Ein einfacher Block B
Ausgabe: Die e_gen und e_kill-Menge für den Block B
Verfahren :

1. Setze $\texttt{e\_gen[B]} := \emptyset$ und $\texttt{e\_kill[B]} := \emptyset$.
2. Durchlaufe die Instruktionen des Blocks B vom ersten bis zum letzten Befehl und führe für jeden Befehl s die folgenden Schritte aus:
   - 2.1 $\texttt{e\_gen[B]} := (\texttt{e\_gen[B]} \cup \texttt{e\_gen}[s]) - \texttt{e\_kill}[s]$
   - 2.2 $\texttt{e\_kill[B]} := \texttt{e\_kill[B]} \cup \texttt{e\_kill}[s]$

Beispiel

Annahme: Vor Eintritt in diesen Block ist kein Ausdruck verfügbar.

Befehl	verfügbare Ausdrücke
	$\emptyset$
`a := b + c`	
	$\{$`b + c`$\}$
`b := a - d`	
	$\{$`a - d`$\}$
`c := b + c`	
	$\{$`a - d`$\}$
`d := a - d`	
	$\emptyset$

Es wäre also $\texttt{e\_gen}[B] = \emptyset$ und es gilt

$$\{\texttt{b + c}, \texttt{a - d}\} \subseteq \texttt{e\_kill}[B].$$

Datenfluss-Gleichungen

Offensichtlich haben wir hier eine Vorwärtsanalyse.

$$\texttt{out}[B] = \texttt{e\_gen}[B] \cup (\texttt{in}[B] - \texttt{e\_kill}[B])$$

$$\texttt{in}[B] = \bigcap_{P \in \text{pred}(B)} \texttt{out}[P]$$

Für den Eingangsblock gilt offensichtlich die Randbedingung $\texttt{out}[\text{entry}] = \emptyset$.

Bemerkung

Ein Ausdruck ist also nur dann am Anfang eines Blocks verfügbar, wenn er am Ende aller Vorgängerblöcke verfügbar ist.

Algorithmus zur Bestimmung verfügbarer Ausdrücke

Eingabe: Ein Flussgraph, für dessen Blöcke die jeweiligen e_gen- und e_kill-Mengen bestimmt worden sind.
Ausgabe: Die out und in-Mengen für jeden Block im Flussgraphen.
Verfahren :
pred(B) sei die Menge der Vorgängerblöcke von B im Flussgraphen.

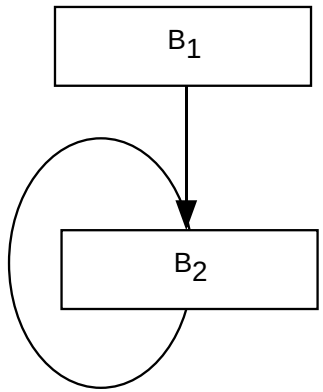
1. Setze out[B] := U, out[entry] = $\emptyset$.
2. Solange sich eine der out-Mengen ändert, führe man die folgenden Schritte für jeden Block B aus:
   - 2.1 in[B] := $\bigcap_{P \in pred(B)}$ out[P]
   - 2.2 out[B] : = e_gen[B] $\cup$ (in[B] - e_kill[B])

Bemerkung

Die Lösung wird wieder iterativ bestimmt. Beim Start der Iteration muss man jedoch anders initialisieren:

- *Man setzt die* $\mathtt{out}$*-Menge für den „entry“-Knoten auf die leere Menge*
- *Für alle anderen Knoten setzt man die* $\mathtt{out}$*-Menge auf die Menge aller Ausdrücke* U*.*

Beispiel



Man kann die zeitliche Entwicklung der in- und out-Werte durch die folgenden Rekursionen beschreiben, dabei seinen $\mathtt{out}[B_2]_j$ bzw. $\mathtt{in}[B_2]_j$ der Wert der out- bzw. in-Menge von $\mathtt{B}_2$ zum Zeitpunkt j.

$$\begin{aligned} \mathtt{in}[B_2]_{j+1} &= \mathtt{out}[B_1]_j \cap \mathtt{out}[B_2]_j \\ \mathtt{out}[B_2]_{j+1} &= \mathtt{e\_gen}[B_2] \cup (\mathtt{in}[B_2]_{j+1} - \mathtt{e\_kill}[B_2]) \end{aligned}$$

Würde man mit $\texttt{out}[B_2]_0 = \emptyset$ starten, dann wäre $\texttt{in}[B_2]_1 = \texttt{out}[B_1]_0 \cap \texttt{out}[B_2]_0 = \emptyset$. Das bedeutet aber, dass ein in B_1 generierter Ausdruck wie `a op b`, der in B_2 nicht auftritt und auch nicht in der Kill-Menge von B_2 vorkommt, nie in der out-Menge von B_2 auftaucht.

Startet man dagegen mit $\texttt{out}[B_2]_0 = U$, erhält man korrekterweise $\texttt{in}[B_2]_1 = \texttt{out}[B_1]_0 \cap \texttt{out}[B_2]_0 = \texttt{out}[B_1]$.

Tabelle mit den wichtigsten Parametern für die Datenfluss-Analyse

	verfügbare Definitionen	Lebendigkeit	verfügbare Ausdrücke
Domain	Menge von Definitionen	Menge von Variablen	Menge von Ausdrücken
Richtung	Vorwärts	Rückwärts	Vorwärts
Transfer-Fkt.	$\texttt{gen}[B] \cup (x - \texttt{kill}[B])$	$\texttt{use}[B] \cup (x - \texttt{def}[B])$	$\texttt{e\_gen}[B] \cup (x - \texttt{e\_kill}[B])$
Gleichungen	$\texttt{out[B]} = f_B(\texttt{in[B]})$	$\texttt{in[B]} = f_B(\texttt{out[B]})$	$\texttt{out[B]} = f_B(\texttt{in[B]})$
	$\texttt{in[B]} = \cup_{B' \in \text{pred}[B]} \texttt{out[B']}$	$\texttt{out[B]} = \cup_{B' \in \text{succ}[B]} \texttt{in[B']}$	$\texttt{in[B]} = \cap_{B' \in \text{pred}[B]} \texttt{out[B']}$
Initialisierung	$\texttt{in}[B] = \texttt{out}[B] = \emptyset$	$\texttt{in}[B] = \texttt{out}[B] = \emptyset$	$\texttt{in}[B] = \texttt{out}[B] = N$
Randbedingung	$\texttt{out}[\text{entry}] = \emptyset$	$\texttt{in}[\text{exit}] = \emptyset$	$\texttt{out}[\text{entry}] = \emptyset$

Analyse der Ablaufstruktur

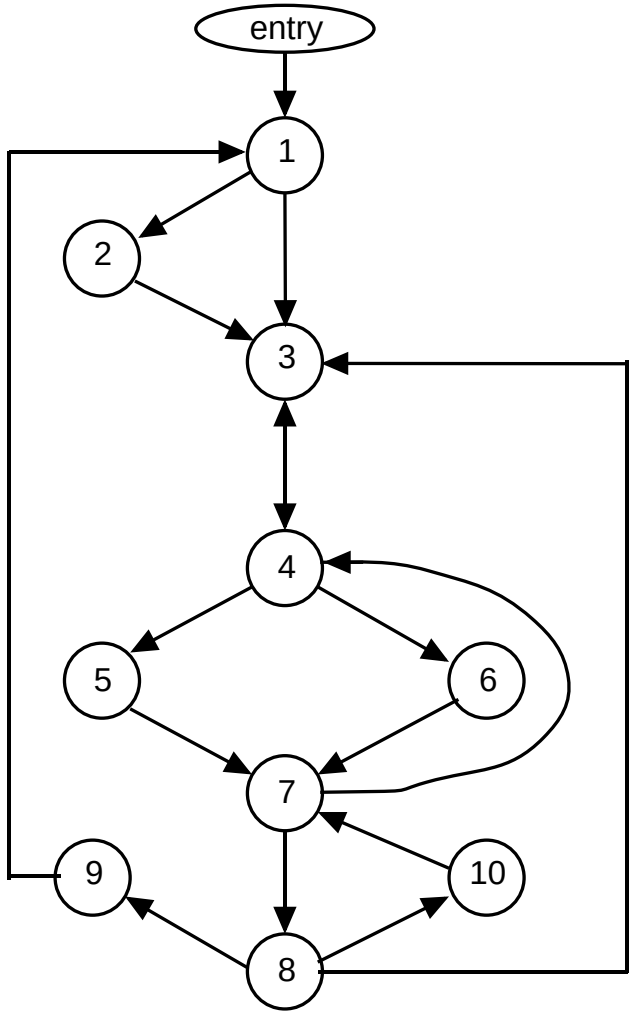
Definition

Gegeben sei ein erweiterter Flussgraph. Block $\mathtt{B_d}$ dominiert Block $\mathtt{B}$, geschrieben $\mathtt{B_d} \succeq \mathtt{B}$, falls $\mathtt{B_d}$ auf jedem Weg vom Knoten entry nach $\mathtt{B}$ liegt. Die Relation $\succeq$ ist offensichtlich eine Ordnung auf der Menge der Blöcke.

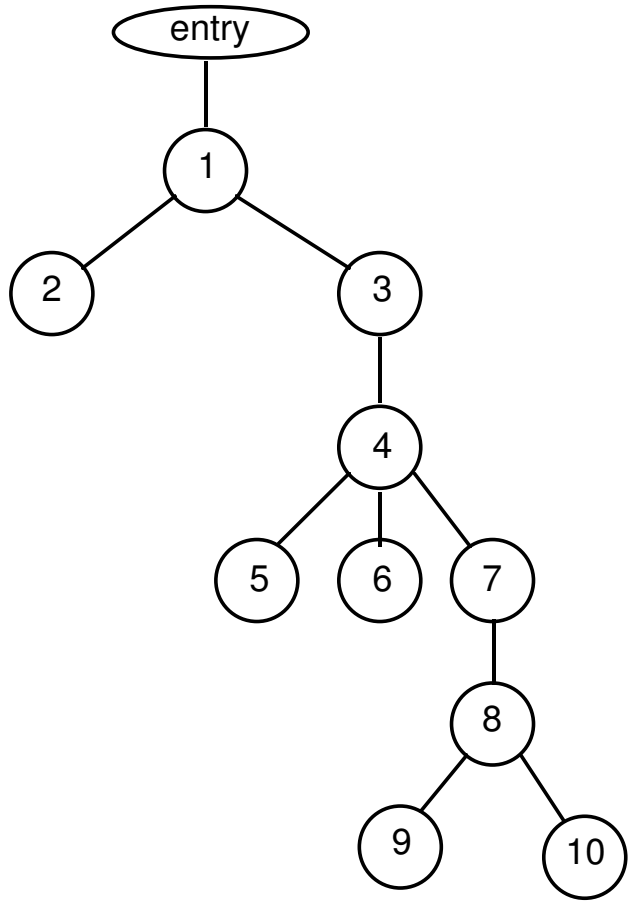
Block $\mathtt{B_1}$ ist direkter Dominator-Block vom Block $\mathtt{B_2}$, geschrieben $\mathtt{B_1} \succ \mathtt{B_2}$, falls für alle Blöcke $\mathtt{B}$ gilt: Aus $\mathtt{B} \succeq \mathtt{B_2}$ und $\mathtt{B} \neq \mathtt{B_2}$ folgt $\mathtt{B} \succeq \mathtt{B_1}$

Jeder Block außer entry hat einen eindeutigen direkten Dominator-Block.

Beispiel



Der Dominator-Baum gibt für jeden Block den direkten Dominator-Block als Vorgängerknoten an.



Algorithmus zur Berechnung der $\succeq$-Relation

Eingabe:
Ein Flussgraph mit der Knotenmenge N und einem Startknoten entry $\in N$.
Ausgabe:
Die $\succeq$-Relation. Für jeden Knoten B aus N gibt $D(B) = \texttt{out}[B]$ die Menge der Dominatoren von B an. Es ist $a \in D(b)$ genau dann wenn $a \succeq b$.
Verfahren:
Man löse die folgende Datenfluss-Analyse:

	Dominatoren
Domain	Potenzmenge von N
Richtung	Vorwärts
Transfer-Fkt.	$f_B(x) = x \cup \{B\}$
Gleichungen	$\texttt{out[B]} = f_B(\texttt{in[B]})$
	$\texttt{in[B]} = \cap_{P \in \text{pred}[B]} \texttt{out[P]})$
Randbedingung	$\texttt{out[entry]} = \{\texttt{entry}\}$
Initialisierung	$\texttt{out[B]} = N$

Beispiel

Für den Flussgraphen aus dem Beispiel ergibt sich:

$$
\begin{aligned}
D(1) &:= \{1\} \\
D(2) &:= \{2\} \cup D(1) = \{1,2\} \\
D(3) &:= \{3\} \cup (D(1) \cap D(2) \cap D(8)) = \{1,3\} \\
D(4) &:= \{4\} \cup (D(3) \cap D(7)) = \{1,3,4\} \\
D(5) &:= \{5\} \cup D(4) = \{1,3,4,5\} \\
D(6) &:= \{6\} \cup D(4) = \{1,3,4,6\} \\
D(7) &:= \{7\} \cup (D(5) \cap D(6) \cap D(10)) = \{1,3,4,7\} \\
D(8) &:= \{8\} \cup D(7) = \{1,3,4,7,8\} \\
D(9) &:= \{9\} \cup D(8) = \{1,3,4,7,8,9\} \\
D(10) &:= \{10\} \cup D(8) = \{1,3,4,7,8,10\}
\end{aligned}
$$

Ein zweiter Durchlauf durch die Schleife bringt bei diesem Beispiel bereits keine weitere Veränderung.

Natürliche Schleifen

Mit dieser Information ist nun möglich, Schleifen im Programmablauf zu erkennen.
Eigenschaften einer Schleife:

1. Eine Schleife hat einen einzigen Eintrittspunkt oder Kopf (Header). Dieser Knoten dominiert alle anderen Knoten in der Schleife.
2. Es muss mindestens einen Knoten in der Schleife geben, vom dem aus eine Kante zum Eintrittspunkt zurückführt.

Also muss man Kanten $a \to b$ im Flussgraphen suchen, für die $b \succeq a$ gilt, d.h. der Zielknoten b dominiert den Ausgangsknoten a.
Die Kanten nennt man Rückwärtskanten (Back-Edges).
In unserem Beispiel wären dies die Kanten $7 \to 4$, $10 \to 7$, $4 \to 3$, $8 \to 3$ und $9 \to 1$.

Definition

Die natürliche Schleife einer Rückwärtskante $n \to d$, $n \neq d$, besteht aus d und den Knoten, über die man n erreichen kann ohne über d zu gehen. Ist $n = d$, so besteht die natürliche Schleife nur aus Block d.
d ist der Eintrittspunkt der Schleife.

Bemerkung

Natürliche Schleifen haben die Eigenschaft, dass zwei Schleifen entweder disjunkt sind oder, sofern beide Schleifen nicht den gleichen Kopf besitzen, die eine in der anderen enthalten ist. Für den Fall, dass zwei Schleifen den gleichen Kopf haben und keine in der anderen enthalten ist, ist es sinnvoll zu sein, beide zusammen als eine Schleife aufzufassen.

Algorithmus zur Konstruktion einer natürlichen Schleife

Eingabe:
Ein Flussgraph und eine Rückwärtskante $n \to d$
Ausgabe:
Eine Menge `loop`, die alle Knoten enthält, die in der natürlichen Schleife von $n \to d$ enthalten sind.
Verfahren :
folgender Algorithmus liefert das Ergebnis:

```
stack := leer;
loop := {d};
if n = d return;
insert (n);
while stack ≠ leer do begin
    m := pop(stack);
    for jedem Vorgänger p von m do
        insert(p);
    end;

procedure insert(m)
if m ∉ loop then begin
    loop := loop ∪ {m}
    push (m, stack)
end
```

Beispiel

Man betrachte wieder den Flussgraphen aus dem Beispiel.

- Zur Kante $10 \to 7$ gehört die Schleife 7,8,10.
- Zur Kante $7 \to 4$ gehört die Schleife 4,5,6,7,8,10.
- Die Kanten $4 \to 3$ und $8 \to 3$ definieren die Schleife 3,4,5,6,7,8,10 und
- die Kante $9 \to 1$ definiert eine Schleife, die alle Knoten von 1 bis 10 enthält.
- Die Knoten 4,5,6,7 bilden keine Schleife, da in diese Knotenmenge sowohl über 4 als auch über 7 eingetreten werden kann und damit die Kopf-Bedingung verletzt ist.