Compiler Construction

Lecture 24: Code Optimization I (Introduction to Dataflow Analysis)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

noll@cs.rwth-aachen.de

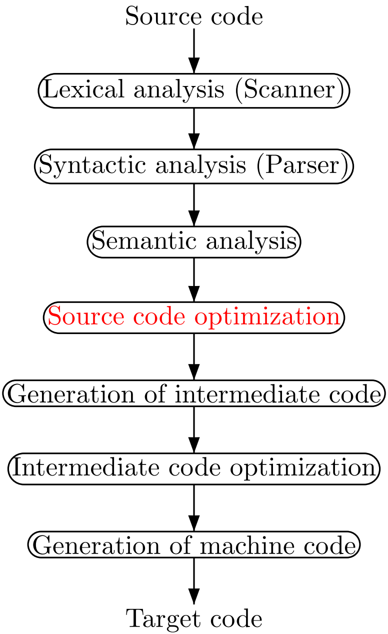
http://www-i2.informatik.rwth-aachen.de/i2/cc09/

Winter semester 2009/10

Outline

1. Code Optimization
2. Preliminaries on Dataflow Analysis
3. Example: Available Expressions Analysis
4. Example: Live Variables Analysis

Conceptual Structure of a Compiler



Code Optimization

Goal: Make generated code faster and/or more compact

Common procedure:

- Gather information about program by performing some kind of analysis
- Exploit information to optimize code

Here: dataflow analysis

$\Longrightarrow$ attach properties to program statements
that hold every time when statement is executed

Outline

Dataflow Analysis: the Approach

- Traditional form of program analysis
- Idea: describe how analysis information flows through program
- Distinctions:

  direction of flow: forward vs. backward analyses

  procedures: interprocedural vs. intraprocedural analyses

  quantification over paths: may (union) vs. must (intersection) analyses

  dependence on statement order: flow-sensitive vs. flow-insensitive analyses

  distinction of procedure calls: context-sensitive vs. context-insensitive analyses

Labelled Programs

- Goal: localization of analysis information
- Dataflow information will be associated with
  - assignments
  - tests in conditionals (`if`) and loops (`while`)

  These constructs will be called blocks (denotation: *Blk*).
- Assume set of labels *Lab* with meta variable $l \in Lab$ (usually $Lab = \mathbb{N}$)

Definition 24.1 (Labelled WHILE programs)

The syntax of labelled WHILE programs is defined by the following context-free grammar:

$$
\begin{aligned}
A &::= z \mid I \mid A_1 \texttt{ + } A_2 \in AExp \\
B &::= A_1 \texttt{ < } A_2 \mid \texttt{not } B \mid B_1 \texttt{ and } B_2 \in BExp \\
C &::= [I \texttt{ := } A]^l \mid C_1 \texttt{;} C_2 \mid \\
&\qquad \texttt{if } [B]^l \texttt{ then } C_1 \texttt{ else } C_2 \mid \texttt{while } [B]^l \texttt{ do } C \in Cmd
\end{aligned}
$$

Here all labels in a statement $C \in Cmd$ are assumed to be distinct.

A WHILE Program with Labels

Example 24.2

```
x := 6;
y := 7;
z := 0;
while x > 0 do
  x := x - 1;
  v := y;
  while v > 0 do
    v := v - 1;
    z := z + 1;
```

Representing Control Flow I

Every (labelled) statement has a single entry (given by the initial label) and generally multiple exits (given by the final labels):

Definition 24.3 (Initial and final labels)

The mapping $\mathrm{init} : \mathit{Cmd} \to \mathit{Lab}$ returns the initial label of a statement:

$$\begin{aligned}
\mathrm{init}([I\ \texttt{:=}\ A]^l) &:= l \\
\mathrm{init}(C_1\,\texttt{;}\,C_2) &:= \mathrm{init}(C_1) \\
\mathrm{init}(\texttt{if}\ [B]^l\ \texttt{then}\ C_1\ \texttt{else}\ C_2) &:= l \\
\mathrm{init}(\texttt{while}\ [b]^l\ \texttt{do}\ C) &:= l
\end{aligned}$$

The mapping $\mathrm{final} : \mathit{Cmd} \to 2^{\mathit{Lab}}$ returns the set of final labels of a statement:

$$\begin{aligned}
\mathrm{final}([I\ \texttt{:=}\ A]^l) &:= \{l\} \\
\mathrm{final}(C_1\,\texttt{;}\,C_2) &:= \mathrm{final}(C_2) \\
\mathrm{final}(\texttt{if}\ [B]^l\ \texttt{then}\ C_1\ \texttt{else}\ C_2) &:= \mathrm{final}(C_1) \cup \mathrm{final}(C_2) \\
\mathrm{final}(\texttt{while}\ [B]^l\ \texttt{do}\ C) &:= \{l\}
\end{aligned}$$

Representing Control Flow II

Definition 24.4 (Flow relation)

Given a statement $C \in Cmd$, the (control) flow relation $\text{flow}(C) \subseteq Lab \times Lab$ is defined by

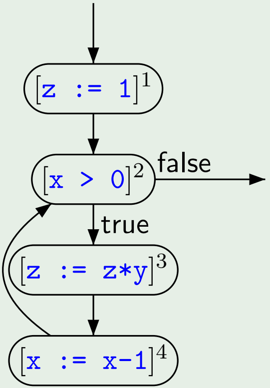
$$
\begin{aligned}
\text{flow}([I \texttt{ := } A]^l) &:= \emptyset \\
\text{flow}(C_1 \texttt{;} C_2) &:= \text{flow}(C_1) \cup \text{flow}(C_2) \cup \\
&\quad\; \{(l, \text{init}(C_2)) \mid l \in \text{final}(C_1)\} \\
\text{flow}(\texttt{if } [B]^l \texttt{ then } C_1 \texttt{ else } C_2) &:= \text{flow}(C_1) \cup \text{flow}(C_2) \cup \\
&\quad\; \{(l, \text{init}(C_1)), (l, \text{init}(C_2))\} \\
\text{flow}(\texttt{while } [B]^l \texttt{ do } C) &:= \text{flow}(C) \cup \{(l, \text{init}(C))\} \cup \\
&\quad\; \{(l', l) \mid l' \in \text{final}(C)\}
\end{aligned}
$$

Example 24.5

$C = $ `[z := 1]`^{1};
    `while [x > 0]`^{2} `do`
        `[z := z*y]`^{3};
        `[x := x-1]`^{4}

$\text{init}(C) = 1$
$\text{final}(C) = \{2\}$
$\text{flow}(C) = \{(1,2),(2,3),(3,4),(4,2)\}$

Visualization by flow graph:



Representing Control Flow IV

- To simplify the presentation we will often assume that the program $C \in Cmd$ under consideration has an isolated entry, meaning that

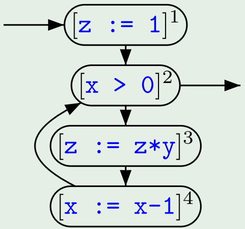
$$\{l \in Lab \mid (l, \text{init}(C)) \in \text{flow}(C)\} = \emptyset$$

(which is the case when C does not start with a `while` loop)

- Similarly: $C \in Cmd$ has isolated exits if

$$\{l' \in Lab \mid (l, l') \in \text{flow}(C) \text{ for some } l \in \text{final}(C)\} = \emptyset$$

Example 24.6



has an isolated entry but not isolated exits

Outline

Goal of the Analysis

Available Expressions Analysis

The goal of Available Expressions Analysis is to determine, for each program point, which (complex) expressions *must* have been computed, and not later modified, on all paths to the program point.

- can be used to avoid recomputations of expressions
- only interesting for non-trivial (i.e., complex) arithmetic expressions

Example 24.7 (Available Expressions Analysis)

```
[x := a+b]^1;
[y := a*b]^2;
while [y > a+b]^3 do
  [a := a+1]^4;
  [x := a+b]^5
```

- a+b available at label 3
- a+b not available at label 5
- possible optimization:
  `while [y > x]^3 do`

Formalizing Available Expressions Analysis I

- Given $C \in Cmd$, $Lab_C/Blk_C/AExp_C$ denote the sets of all labels/blocks/complex arithmetic expressions occurring in C, respectively
- An expression A is killed in a block β if any of the variables in A is modified in β
- Formally: $\mathrm{kill}_{AE} : Blk_C \to 2^{AExp_C}$ is defined by
  $$\mathrm{kill}_{AE}([I := A]^l) := \{A' \in AExp_C \mid I \in \mathrm{FV}(A')\}$$
  $$\mathrm{kill}_{AE}([B]^l) := \emptyset$$
- An expression A is generated in a block β if it is evaluated in and none of its variables are modified by β
- Formally: $\mathrm{gen}_{AE} : Blk_C \to 2^{AExp_C}$ is defined by
  $$\mathrm{gen}_{AE}([I := A]^l) := \{A \mid I \notin \mathrm{FV}(A)\}$$
  $$\mathrm{gen}_{AE}([B]^l) := AExp_B$$

Formalizing Available Expressions Analysis II

Example 24.8 ($\mathrm{kill}_{AE}/\mathrm{gen}_{AE}$ functions)

```
C = [x := a+b]^1;
    [y := a*b]^2;
    while [y > a+b]^3 do
      [a := a+1]^4;
      [x := a+b]^5
```

- $AExp_C = \{$a+b, a*b, a+1$\}$
-

Lab_C	$\mathrm{kill}_{AE}(\beta^l)$	$\mathrm{gen}_{AE}(\beta^l)$
1	$\emptyset$	{a+b}
2	$\emptyset$	{a*b}
3	$\emptyset$	{a+b}
4	{a+b, a*b, a+1}	$\emptyset$
5	$\emptyset$	{a+b}

The Equation System I

- Analysis itself defined by setting up an equation system
- For each $l \in Lab_C$, $AE_l \subseteq AExp_C$ represents the set of available expressions at the entry of block β^l
- Formally, for $C \in Cmd$ with isolated entry:

$$AE_l = \begin{cases} \emptyset & \text{if } l = \text{init}(C) \\ \bigcap\{\varphi_{l'}(AE_{l'}) \mid (l', l) \in \text{flow}(C)\} & \text{otherwise} \end{cases}$$

  where $\varphi_{l'} : 2^{AExp_C} \to 2^{AExp_C}$ denotes the transfer function of block $\beta^{l'}$, given by

$$\varphi_{l'}(A) := (A \setminus \text{kill}_{AE}(\beta^{l'})) \cup \text{gen}_{AE}(\beta^{l'})$$

- Characterization of analysis:

  forward: starts in $\text{init}(C)$ and proceeds downwards

  must: $\bigcap$ in equation for AE_l

  flow-sensitive: results depending on order of assignments

- Later: solution not necessarily unique

  $\Longrightarrow$ choose greatest one

The Equation System II

Reminder:

$$AE_l = \begin{cases} \emptyset & \text{if } l = \text{init}(C) \\ \bigcap\{\varphi_{l'}(AE_{l'}) \mid (l', l) \in \text{flow}(C)\} & \text{otherwise} \end{cases}$$

$$\varphi_{l'}(E) = (E \setminus \text{kill}_{AE}(\beta^{l'})) \cup \text{gen}_{AE}(\beta^{l'})$$

Example 24.9 (AE equation system)

$C =$
```
[x := a+b]^1;
[y := a*b]^2;
while [y > a+b]^3 do
  [a := a+1]^4;
  [x := a+b]^5
```

$l \in Lab_C$	$\text{kill}_{AE}(\beta^l)$	$\text{gen}_{AE}(\beta^l)$
1	$\emptyset$	{a+b}
2	$\emptyset$	{a*b}
3	$\emptyset$	{a+b}
4	{a+b, a*b, a+1}	$\emptyset$
5	$\emptyset$	{a+b}

Equations:

$$\begin{aligned} AE_1 &= \emptyset \\ AE_2 &= \varphi_1(AE_1) = AE_1 \cup \{\texttt{a+b}\} \\ AE_3 &= \varphi_2(AE_2) \cap \varphi_5(AE_5) \\ &= (AE_2 \cup \{\texttt{a*b}\}) \cap (AE_5 \cup \{\texttt{a+b}\}) \\ AE_4 &= \varphi_3(AE_3) = AE_3 \cup \{\texttt{a+b}\} \\ AE_5 &= \varphi_4(AE_4) = AE_4 \setminus \{\texttt{a+b}, \texttt{a*b}, \texttt{a+1}\} \end{aligned}$$

Solution:

$$\begin{aligned} AE_1 &= \emptyset \\ AE_2 &= \{\texttt{a+b}\} \\ AE_3 &= \{\texttt{a+b}\} \\ AE_4 &= \{\texttt{a+b}\} \\ AE_5 &= \emptyset \end{aligned}$$

Outline

Goal of the Analysis

Live Variables Analysis

The goal of Live Variables Analysis is to determine, for each program point, which variables *may* be live at the exit from the point.

- A variable is called live at the exit from a block if there exists a path from the block to a use of the variable that does not re-define the variable
- All variables considered to be live at the end of the program (alternative: restriction to output variables)
- Can be used for Dead Code Elimination:
  remove assignments to non-live variables

An Example

Example 24.10 (Live Variables Analysis)

```
[x := 2]¹;
[y := 4]²;
[x := 1]³;
if [y > 0]⁴ then
  [z := x]⁵
else
  [z := y*y]⁶;
[x := z]⁷
```

- x not live at exit from label 1
- y live at exit from 2
- x live at exit from 3
- z live at exits from 5 and 6
- possible optimization: remove [x := 2]¹

Formalizing Live Variables Analysis I

- A variable on the left-hand side of an assignment is killed by the assignment; tests do not kill
- Formally: $\mathrm{kill}_{LV} : Blk_C \to 2^{Var_C}$ is defined by
  $$\mathrm{kill}_{LV}([I \texttt{ := } A]^l) := \{I\}$$
  $$\mathrm{kill}_{LV}([B]^l) := \emptyset$$
- Every reading access generates a live variable
- Formally: $\mathrm{gen}_{LV} : Blk_C \to 2^{Var_C}$ is defined by
  $$\mathrm{gen}_{LV}([I \texttt{ := } A]^l) := \mathrm{FV}(A)$$
  $$\mathrm{gen}_{LV}([B]^l) := \mathrm{FV}(B)$$

Example 24.11 ($\text{kill}_{LV}/\text{gen}_{LV}$ functions)

$c =$ [x := 2][1];
[y := 4][2];
[x := 1][3];
if [y > 0][4] then
[z := x][5]
else
[z := y*y][6];
[x := z][7]

- $Var_c = \{x, y, z\}$
-

$l \in Lab_c$	$\text{kill}_{LV}(\beta^l)$	$\text{gen}_{LV}(\beta^l)$
1	{x}	$\emptyset$
2	{y}	$\emptyset$
3	{x}	$\emptyset$
4	$\emptyset$	{y}
5	{z}	{x}
6	{z}	{y}
7	{x}	{z}

The Equation System I

- For each $l \in Lab_C$, $LV_l \subseteq Var_c$ represents the set of live variables at the exit of block β^l
- Formally, for a program $C \in Cmd$ with isolated exits:
$$LV_l = \begin{cases} Var_C & \text{if } l \in \text{final}(C) \\ \bigcup\{\varphi_{l'}(LV_{l'}) \mid (l, l') \in \text{flow}(C)\} & \text{otherwise} \end{cases}$$
where $\varphi_{l'} : 2^{Var_C} \to 2^{Var_C}$ denotes the transfer function of block $\beta^{l'}$, given by
$$\varphi_{l'}(V) := (V \setminus \text{kill}_{LV}(\beta^{l'})) \cup \text{gen}_{LV}(\beta^{l'})$$
- Characterization of analysis:
  - backward: starts in final(C) and proceeds upwards
  - may: $\bigcup$ in equation for LV_l
  - flow-sensitive: results depending on order of assignments
- Later: solution not necessarily unique
  $\Longrightarrow$ choose least one

The Equation System II

Reminder:

$$LV_l = \begin{cases} Var_C & \text{if } l \in \mathrm{final}(C) \\ \bigcup\{\varphi_{l'}(LV_{l'}) \mid (l,l') \in \mathrm{flow}(C)\} & \text{otherwise} \end{cases}$$

$$\varphi_{l'}(V) = (V \setminus \mathrm{kill}_{LV}(\beta^{l'})) \cup \mathrm{gen}_{LV}(\beta^{l'})$$

Example 24.12 (LV equation system)

$C =$
```
[x := 2]^1;[y := 4]^2;
[x := 1]^3;
if [y > 0]^4 then
  [z := x]^5
else
  [z := y*y]^6;
[x := z]^7
```

$l \in Lab_c$	$\mathrm{kill}_{LV}(\beta^l)$	$\mathrm{gen}_{LV}(\beta^l)$
1	{x}	$\emptyset$
2	{y}	$\emptyset$
3	{x}	$\emptyset$
4	$\emptyset$	{y}
5	{z}	{x}
6	{z}	{y}
7	{x}	{z}

$$\begin{aligned}
LV_1 &= \varphi_2(LV_2) = LV_2 \setminus \{y\} \\
LV_2 &= \varphi_3(LV_3) = LV_3 \setminus \{x\} \\
LV_3 &= \varphi_4(LV_4) = LV_4 \cup \{y\} \\
LV_4 &= \varphi_5(LV_5) \cup \varphi_6(LV_6) \\
&= ((LV_5 \setminus \{z\}) \cup \{x\}) \cup \\
&\quad ((LV_6 \setminus \{z\}) \cup \{y\}) \\
LV_5 &= \varphi_7(LV_7) = (LV_7 \setminus \{x\}) \cup \{z\} \\
LV_6 &= \varphi_7(LV_7) = (LV_7 \setminus \{x\}) \cup \{z\} \\
LV_7 &= \{x, y, z\}
\end{aligned}$$

Solution:

$$\begin{aligned}
LV_1 &= \emptyset \\
LV_2 &= \{y\} \\
LV_3 &= \{x, y\} \\
LV_4 &= \{x, y\} \\
LV_5 &= \{y, z\} \\
LV_6 &= \{y, z\} \\
LV_7 &= \{x, y, z\}
\end{aligned}$$