Semantics and Verification of Software

Lecture 14: Dataflow Analysis I (Introduction)

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/svsw08/

Winter semester 2008/09

Outline

Extending the Syntax

Syntactic categories:

Category	Domain	Meta variable
Procedure identifiers	$\mathit{PVar} = \{\mathtt{P}, \mathtt{Q}, \ldots\}$	P
Procedure declarations	PDec	p
Variable declarations	VDec	v
Commands (statements)	Cmd	c

Context-free grammar:

$$\begin{aligned}
p &::= \mathtt{proc}\ P\ \mathtt{is}\ c\mathtt{;}p \mid \varepsilon \in \mathit{PDec} \\
v &::= \mathtt{var}\ x\mathtt{;}v \mid \varepsilon \in \mathit{VDec} \\
c &::= \mathtt{skip} \mid x\ \mathtt{:=}\ a \mid c_1\mathtt{;}c_2 \mid \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 \mid \mathtt{while}\ b\ \mathtt{do}\ c \mid \\
&\qquad \mathtt{call}\ P \mid \mathtt{begin}\ v\ p\ c\ \mathtt{end} \in \mathit{Cmd}
\end{aligned}$$

Execution Relation I

Definition (Execution relation)

For $c \in Cmd$, $\sigma, \sigma' \in Sto$, $\rho \in VEnv$, and $\pi \in PEnv$, the execution relation $(\rho, \pi) \vdash \langle c, \sigma \rangle \to \sigma'$ is defined by the following rules:

$$(\text{skip})\ \frac{}{(\rho, \pi) \vdash \langle \texttt{skip}, \sigma \rangle \to \sigma}$$

$$(\text{asgn})\ \frac{\langle a, \sigma \circ \rho \rangle \to z}{(\rho, \pi) \vdash \langle x\ \texttt{:=}\ a, \sigma \rangle \to \sigma[\rho(x) \mapsto z]}$$

$$(\text{seq})\ \frac{(\rho, \pi) \vdash \langle c_1, \sigma \rangle \to \sigma' \quad (\rho, \pi) \vdash \langle c_2, \sigma' \rangle \to \sigma''}{(\rho, \pi) \vdash \langle c_1 \texttt{;} c_2, \sigma \rangle \to \sigma''}$$

$$(\text{if-t})\ \frac{\langle b, \sigma \circ \rho \rangle \to \mathsf{true} \quad (\rho, \pi) \vdash \langle c_1, \sigma \rangle \to \sigma'}{(\rho, \pi) \vdash \langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2, \sigma \rangle \to \sigma'}$$

$$(\text{if-f})\ \frac{\langle b, \sigma \circ \rho \rangle \to \mathsf{false} \quad (\rho, \pi) \vdash \langle c_2, \sigma \rangle \to \sigma'}{(\rho, \pi) \vdash \langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2, \sigma \rangle \to \sigma'}$$

Execution Relation II

Definition (Execution relation; continued)

$$\text{(wh-f)}\,\frac{\langle b, \sigma \circ \rho\rangle \to \mathsf{false}}{(\rho,\pi) \vdash \langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \to \sigma}$$

$$\text{(wh-t)}\,\frac{\langle b, \sigma \circ \rho\rangle \to \mathsf{true}\ (\rho,\pi) \vdash \langle c, \sigma\rangle \to \sigma'\ (\rho,\pi) \vdash \langle \texttt{while}\ b\ \texttt{do}\ c, \sigma'\rangle \to \sigma''}{(\rho,\pi) \vdash \langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \to \sigma''}$$

$$\text{(call)}\,\frac{(\rho', \pi'[P \mapsto (c, \rho', \pi')]) \vdash \langle c, \sigma\rangle \to \sigma'}{(\rho,\pi) \vdash \langle \texttt{call}\ P, \sigma\rangle \to \sigma'} \qquad \text{if } \pi(P) = (c, \rho', \pi')$$

$$\text{(block)}\,\frac{\mathsf{upd}_v[\![v]\!](\rho,\sigma) = (\rho',\sigma') \qquad (\rho', \mathsf{upd}_p[\![p]\!](\rho',\pi)) \vdash \langle c, \sigma'\rangle \to \sigma''}{(\rho,\pi) \vdash \langle \texttt{begin}\ v\ p\ c\ \texttt{end}, \sigma\rangle \to \sigma''}$$

Outline

A Glimpse at the Denotational Semantics

- Similar as before: statements denote storage transformations
- New: dependence on environments
$$\mathfrak{C}[\![.]\!] : \mathit{Cmd} \times \mathit{VEnv} \times \mathit{PEnv} \to (\mathit{Sto} \dashrightarrow \mathit{Sto})$$
- Variable environment obtained as before:
$$\mathit{VEnv} := \{\rho \mid \rho : \mathit{Var} \dashrightarrow \mathit{Loc}\}$$
- Procedures now interpreted as storage transformations:
$$\mathit{PDec} := \{\pi \mid \pi : \mathit{PVar} \dashrightarrow (\mathit{Sto} \dashrightarrow \mathit{Sto})\}$$
- Recursive procedure declarations involve fixpoints:
$$\mathfrak{D}_p[\![.]\!] : \mathit{PDec} \times \mathit{VEnv} \times \mathit{PEnv} \to \mathit{PEnv}$$
$$\mathfrak{D}_p[\![\texttt{proc } P \texttt{ is } c]\!](\rho, \pi) := (\rho, \pi[P \mapsto \mathsf{fix}(\Phi)])$$
where
$$\Phi : (\mathit{Sto} \dashrightarrow \mathit{Sto}) \to (\mathit{Sto} \dashrightarrow \mathit{Sto})$$
$$\Phi(f) := \mathfrak{C}[\![c]\!](\rho, \pi[P \mapsto f])$$

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`)
  - `skip` statements

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

Definition 14.1 (Labelled WHILE programs)

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

$$\begin{aligned} a &::= z \mid x \mid a_1\texttt{+}a_2 \mid a_1\texttt{-}a_2 \mid a_1\texttt{*}a_2 \in \mathit{AExp} \\ b &::= t \mid a_1\texttt{=}a_2 \mid a_1\texttt{>}a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in \mathit{BExp} \\ c &::= [\texttt{skip}]^l \mid [x \texttt{ := } a]^l \mid c_1\texttt{;}c_2 \mid \\ &\qquad \texttt{if } [b]^l \texttt{ then } c_1 \texttt{ else } c_2 \mid \texttt{while } [b]^l \texttt{ do } c \in \mathit{Cmd} \end{aligned}$$

Here all labels in a statement $c \in \mathit{Cmd}$ are assumed to be distinct.

A WHILE Program with Labels

Example 14.2

```
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 14.3 (Initial and final labels)

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

$$\begin{aligned}
\mathsf{init}([\texttt{skip}]^l) &:= l \\
\mathsf{init}([x \texttt{ := } a]^l) &:= l \\
\mathsf{init}(c_1\,;c_2) &:= \mathsf{init}(c_1) \\
\mathsf{init}(\texttt{if } [b]^l \texttt{ then } c_1 \texttt{ else } c_2) &:= l \\
\mathsf{init}(\texttt{while } [b]^l \texttt{ do } c) &:= l
\end{aligned}$$

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

$$\begin{aligned}
\mathsf{final}([\texttt{skip}]^l) &:= \{l\} \\
\mathsf{final}([x \texttt{ := } a]^l) &:= \{l\} \\
\mathsf{final}(c_1\,;c_2) &:= \mathsf{final}(c_2) \\
\mathsf{final}(\texttt{if } [b]^l \texttt{ then } c_1 \texttt{ else } c_2) &:= \mathsf{final}(c_1) \cup \mathsf{final}(c_2) \\
\mathsf{final}(\texttt{while } [b]^l \texttt{ do } c) &:= \{l\}
\end{aligned}$$

Representing Control Flow II

Definition 14.4 (Flow relation)

Given a statement $c \in Cmd$, the (control) flow relation $\mathsf{flow}(c) \subseteq L \times L$ is defined by

$$\begin{aligned}
\mathsf{flow}([\texttt{skip}]^l) &:= \emptyset \\
\mathsf{flow}([x \texttt{ := } a]^l) &:= \emptyset \\
\mathsf{flow}(c_1\,\texttt{;}\,c_2) &:= \mathsf{flow}(c_1) \cup \mathsf{flow}(c_2) \cup \\
&\quad\ \{(l, \mathsf{init}(c_2)) \mid l \in \mathsf{final}(c_1)\} \\
\mathsf{flow}(\texttt{if }[b]^l\texttt{ then }c_1\texttt{ else }c_2) &:= \mathsf{flow}(c_1) \cup \mathsf{flow}(c_2) \cup \\
&\quad\ \{(l, \mathsf{init}(c_1)), (l, \mathsf{init}(c_2))\} \\
\mathsf{flow}(\texttt{while }[b]^l\texttt{ do }c) &:= \mathsf{flow}(c) \cup \{(l, \mathsf{init}(c))\} \cup \\
&\quad\ \{(l', l) \mid l' \in \mathsf{final}(c)\}
\end{aligned}$$

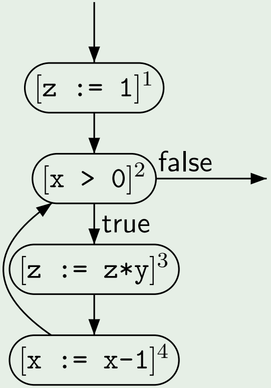
Representing Control Flow III

Example 14.5

Visualization by flow graph:

$$
\begin{aligned}
c = {} & [\texttt{z := 1}]^1; \\
& \quad \texttt{while } [\texttt{x > 0}]^2 \texttt{ do} \\
& \qquad [\texttt{z := z*y}]^3; \\
& \qquad [\texttt{x := x-1}]^4
\end{aligned}
$$

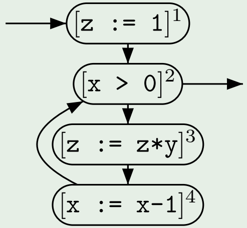
$$
\begin{aligned}
\mathsf{init}(c) &= 1 \\
\mathsf{final}(c) &= \{2\} \\
\mathsf{flow}(c) &= \{(1,2),(2,3),(3,4),(4,2)\}
\end{aligned}
$$



Representing Control Flow IV

- To simplify the presentation we will often assume that the program $c \in \mathit{Cmd}$ under consideration has an isolated entry, meaning that
$$\{l \in L \mid (l, \mathsf{init}(c)) \in \mathsf{flow}(c)\} = \emptyset$$
(which is the case when c does not start with a `while` loop)
- Similarly: $c \in \mathit{Cmd}$ has isolated exits if
$$\{l' \in L \mid (l, l') \in \mathsf{flow}(c) \text{ for some } l \in \mathsf{final}(c)\} = \emptyset$$

Example 14.6



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 14.7 (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 $c \in Cmd$, $L_c/Block_c/AExp_c$ denote the sets of all labels/blocks/complex arithmetic expressions occurring in c, respectively
- An expression a is killed in a block B if any of the variables in a is modified in B
- Formally: $\mathsf{kill}_{\mathsf{AE}} : Block_c \to 2^{AExp_c}$ is defined by

$$\begin{aligned}\mathsf{kill}_{\mathsf{AE}}([\mathtt{skip}]^l) &:= \emptyset\\ \mathsf{kill}_{\mathsf{AE}}([x \ \mathtt{:=}\ a]^l) &:= \{a' \in AExp_c \mid x \in FV(a')\}\\ \mathsf{kill}_{\mathsf{AE}}([b]^l) &:= \emptyset\end{aligned}$$

- An expression a is generated in a block B if it is evaluated in and none of its variables are modified by B
- Formally: $\mathsf{gen}_{\mathsf{AE}} : Block_c \to 2^{AExp_c}$ is defined by

$$\begin{aligned}\mathsf{gen}_{\mathsf{AE}}([\mathtt{skip}]^l) &:= \emptyset\\ \mathsf{gen}_{\mathsf{AE}}([x \ \mathtt{:=}\ a]^l) &:= \{a \mid x \notin FV(a)\}\\ \mathsf{gen}_{\mathsf{AE}}([b]^l) &:= AExp_b\end{aligned}$$

Formalizing Available Expressions Analysis II

Example 14.8 ($\mathsf{kill}_{\mathsf{AE}}/\mathsf{gen}_{\mathsf{AE}}$ functions)

$c = [\texttt{x := a+b}]^1;$
$\quad [\texttt{y := a*b}]^2;$
$\quad \texttt{while } [\texttt{y > a+b}]^3 \texttt{ do}$
$\qquad [\texttt{a := a+1}]^4;$
$\qquad [\texttt{x := a+b}]^5$

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

L_c	$\mathsf{kill}_{\mathsf{AE}}(B^l)$	$\mathsf{gen}_{\mathsf{AE}}(B^l)$
1	$\emptyset$	$\{\texttt{a+b}\}$
2	$\emptyset$	$\{\texttt{a*b}\}$
3	$\emptyset$	$\{\texttt{a+b}\}$
4	$\{\texttt{a+b}, \texttt{a*b}, \texttt{a+1}\}$	$\emptyset$
5	$\emptyset$	$\{\texttt{a+b}\}$

The Equation System I

- Analysis itself defined by setting up an equation system
- For each $l \in L_c$, $\mathsf{AE}_l \subseteq AExp_c$ represents the set of available expressions at the entry of block B^l
- Formally, for $c \in Cmd$ with isolated entry:
$$\mathsf{AE}_l = \begin{cases} \emptyset & \text{if } l = \mathsf{init}(c) \\ \bigcap\{\varphi_{l'}(\mathsf{AE}_{l'}) \mid (l', l) \in \mathsf{flow}(c)\} & \text{otherwise} \end{cases}$$
where $\varphi_{l'} : 2^{AExp_c} \to 2^{AExp_c}$ denotes the transfer function of block $B^{l'}$, given by
$$\varphi_{l'}(A) := (A \setminus \mathsf{kill}_{\mathsf{AE}}(B^{l'})) \cup \mathsf{gen}_{\mathsf{AE}}(B^{l'})$$
- Characterization of analysis:

forward:	starts in $\mathsf{init}(c)$ 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:

$$\mathsf{AE}_l = \begin{cases} \emptyset & \text{if } l = \mathsf{init}(c) \\ \bigcap\{\varphi_{l'}(\mathsf{AE}_{l'}) \mid (l', l) \in \mathsf{flow}(c)\} & \text{otherwise} \end{cases}$$

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

Example 14.9 (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
```

$l \in L_c$	$\mathsf{kill}_{\mathsf{AE}}(B^l)$	$\mathsf{gen}_{\mathsf{AE}}(B^l)$
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:

$$\begin{aligned}
\mathsf{AE}_1 &= \emptyset \\
\mathsf{AE}_2 &= \varphi_1(\mathsf{AE}_1) = \mathsf{AE}_1 \cup \{\texttt{a+b}\} \\
\mathsf{AE}_3 &= \varphi_2(\mathsf{AE}_2) \cap \varphi_5(\mathsf{AE}_5) \\
&= (\mathsf{AE}_2 \cup \{\texttt{a*b}\}) \cap (\mathsf{AE}_5 \cup \{\texttt{a+b}\}) \\
\mathsf{AE}_4 &= \varphi_3(\mathsf{AE}_3) = \mathsf{AE}_3 \cup \{\texttt{a+b}\} \\
\mathsf{AE}_5 &= \varphi_4(\mathsf{AE}_4) = \mathsf{AE}_4 \setminus \{\texttt{a+b}, \texttt{a*b}, \texttt{a+1}\}
\end{aligned}$$

Solution:

$$\begin{aligned}
\mathsf{AE}_1 &= \emptyset \\
\mathsf{AE}_2 &= \{\texttt{a+b}\} \\
\mathsf{AE}_3 &= \{\texttt{a+b}\} \\
\mathsf{AE}_4 &= \{\texttt{a+b}\} \\
\mathsf{AE}_5 &= \emptyset
\end{aligned}$$