Static Program Analysis

Lecture 3: Dataflow Analysis II (Order-Theoretic Foundations)

Winter Semester 2016/17

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-1617/spa/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Labelled Programs

- Goal: localisation of analysis information
- Dataflow information will be associated with
  - `skip` statements
  - assignments
  - tests in conditionals (`if`) and loops (`while`)
- Assume set of labels Lab with meta variable $l \in Lab$ (usually $Lab = \mathbb{N}$)

Definition (Labelled WHILE programs)

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

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

- All labels in $c \in Cmd$ assumed distinct, denoted by Lab_c
- Labelled fragments of c called blocks, denoted by Blk_c

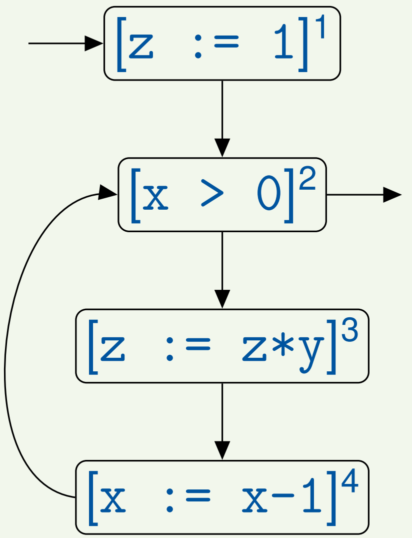
Recap: Dataflow Analysis

Representing Control Flow

Example

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

Visualisation by (control) flow graph:



$$\mathrm{init}(c) = 1$$
$$\mathrm{final}(c) = \{2\}$$
$$\mathrm{flow}(c) = \{(1,2),(2,3),(3,4),(4,2)\}$$

Recap: Dataflow Analysis

Goal of Available Expressions 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 for Common Subexpression Elimination:
  replace subexpression by variable that contains up-to-date value
- Only interesting for non-trivial (i.e., complex) arithmetic expressions

Recap: Dataflow Analysis

The Equation System

- Analysis itself defined by setting up an equation system
- For each $l \in Lab_c$, $\mathrm{AE}_l \subseteq CExp_c$ represents the set of available expressions at the entry of block B^l
- Formally, for $c \in Cmd$ with isolated entry:

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

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

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

- Characterisation of analysis:

  flow-sensitive: results depending on order of assignments

  forward: starts in $\mathrm{init}(c)$ and proceeds downwards

  must: $\bigcap$ in equations for AE_l
- Later: solution not necessarily unique

  $\Longrightarrow$ choose greatest one

Goal of Live Variables 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

The Equation System

- For each $l \in Lab_c$, $\mathsf{LV}_l \subseteq Var_c$ represents the set of live variables at the exit of block B^l
- Formally, for a program $c \in Cmd$ with isolated exits:

$$\mathsf{LV}_l = \begin{cases} Var_c & \text{if } l \in \mathsf{final}(c) \\ \bigcup\{\varphi_{l'}(\mathsf{LV}_{l'}) \mid (l, l') \in \mathsf{flow}(c)\} & \text{otherwise} \end{cases}$$

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

$$\varphi_{l'}(V) := (V \setminus \mathsf{kill}_{\mathsf{LV}}(B^{l'})) \cup \mathsf{gen}_{\mathsf{LV}}(B^{l'})$$

- Characterisation of analysis:

  flow-sensitive: results depending on order of assignments

  backward: starts in $\mathsf{final}(c)$ and proceeds upwards

  may: $\bigcup$ in equations for LV_l

- Later: solution not necessarily unique

  $\Longrightarrow$ choose least one

Similarities Between Analysis Problems

- **Observation:** the analyses presented so far have some similarities

$\Rightarrow$ Look for underlying framework

- **Advantages:**
  - possibility for designing (efficient) generic algorithms for solving dataflow equations
  - enables generic correctness proofs of analyses and algorithms
- **Overall pattern:** for $c \in Cmd$ and $l \in Lab_c$, the analysis information (AI) is described by equations of the form

$$\mathrm{AI}_l = \begin{cases} \iota & \text{if } l \in E \\ \bigsqcup\{\varphi_{l'}(\mathrm{AI}_{l'}) \mid (l', l) \in F\} & \text{otherwise} \end{cases}$$

  where
  - the set of extremal labels, E, is $\{\mathrm{init}(c)\}$ or $\mathrm{final}(c)$
  - ι specifies the extremal analysis information
  - the combination operator, $\bigsqcup$, is $\bigcap$ or $\bigcup$
  - $\varphi_{l'}$ denotes the transfer function of block $B^{l'}$
  - the flow relation F is $\mathrm{flow}(c)$ or $\mathrm{flow}^R(c)$ $(:= \{(l', l) \mid (l, l') \in \mathrm{flow}(c)\})$

Characterisation of Analyses

Direction of information flow

- Forward:
  - $F = \text{flow}(c)$
  - AI_l refers to entry of B^l
  - c has isolated entry
- Backward:
  - $F = \text{flow}^R(c)$
  - AI_l refers to exit of B^l
  - c has isolated exits

Quantification over paths

- May:
  - $\bigsqcup = \bigcup$
  - property satisfied by some path
  - interested in least solution (later)
- Must:
  - $\bigsqcup = \bigcap$
  - property satisfied by all paths
  - interested in greatest solution (later)

Heading for a Dataflow Analysis Framework

Roadmap

Goal: solve dataflow equation system by fixpoint iteration

1. Characterise solution of equation system as fixpoint of a transformation
2. Introduce partial order for comparing analysis results
3. Establish least upper bound as combination operator
4. Ensure monotonicity of transfer functions
5. Guarantee termination of fixpoint iteration by ascending chain condition
6. Optimise fixpoint iteration by worklist algorithm

Heading for a Dataflow Analysis Framework

Motivation

- **Wanted:** solution of (dataflow) equation system
- **Problem:** recursive dependencies between dataflow variables
- **Idea:** characterise solution as fixpoint of transformation:

$$(\mathsf{AI}_l = \tau_l)_{l \in Lab_c} \iff \Phi((\mathsf{AI}_l)_{l \in Lab_c}) = (\mathsf{AI}_l)_{l \in Lab_c}$$

  where $\Phi\left((\mathsf{AI}_l)_{l \in Lab_c}\right) := (\tau_l)_{l \in Lab_c}$
- **Approach:** approximate fixpoint by iteration

Order-Theoretic Foundations: The Domain

Partial Orders

The domain of analysis information usually forms a partial order where the ordering relation compares the "precision" of information.

Definition 3.1 (Partial order)

A partial order (PO) $(D, \sqsubseteq)$ consists of a set D, called domain, and of a relation $\sqsubseteq \subseteq D \times D$ such that, for every $d_1, d_2, d_3 \in D$,

reflexivity: $d_1 \sqsubseteq d_1$
transitivity: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_3 \implies d_1 \sqsubseteq d_3$
antisymmetry: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_1 \implies d_1 = d_2$

It is called total if, in addition, always $d_1 \sqsubseteq d_2$ or $d_2 \sqsubseteq d_1$.

Example 3.2

1. $(\mathbb{N}, \leq)$ is a total partial order
2. $(\mathbb{N}, <)$ is not a partial order (since not reflexive)
3. (Live Variables) $(2^{Var_c}, \subseteq)$ is a (non-total) partial order
4. (Available Expressions) $(2^{CExp_c}, \supseteq)$ is a (non-total) partial order

Upper Bounds

In the dataflow equation system, analysis information from several predecessors is combined by taking the least upper bound.

Definition 3.3 ((Least) upper bound)

Let $(D, \sqsubseteq)$ be a partial order and $S \subseteq D$.

1. An element $d \in D$ is called an upper bound of S if $s \sqsubseteq d$ for every $s \in S$ (notation: $S \sqsubseteq d$).
2. An upper bound d of S is called least upper bound (LUB) or supremum of S if $d \sqsubseteq d'$ for every upper bound d' of S (notation: $d = \bigsqcup S$).

Example 3.4

1. $S \subseteq \mathbb{N}$ has a LUB in $(\mathbb{N}, \leq)$ iff it is finite
2. (Live Variables) $(D, \sqsubseteq) = (2^{Var_c}, \subseteq)$. Given $V_1, \ldots, V_n \subseteq Var_c$,
$$\bigsqcup\{V_1, \ldots, V_n\} = \bigcup\{V_1, \ldots, V_n\}$$
3. (Available Expressions) $(D, \sqsubseteq) = (2^{CExp_c}, \supseteq)$. Given $A_1, \ldots, A_n \subseteq CExp_c$,
$$\bigsqcup\{A_1, \ldots, A_n\} = \bigcap\{A_1, \ldots, A_n\}$$

Order-Theoretic Foundations: The Domain

Complete Lattices

Since $\{\varphi_{l'}(\mathsf{AI}_{l'}) \mid (l', l) \in F\}$ can contain arbitrary elements, the existence of least upper bounds must be ensured for arbitrary subsets.

Definition 3.5 (Complete lattice)

A complete lattice is a partial order $(D, \sqsubseteq)$ such that all subsets of D have least upper bounds. In this case,

$$\perp := \bigsqcup \emptyset$$

denotes the least element of D.

Example 3.6

1. $(\mathbb{N}, \leq)$ is not a complete lattice as, e.g., $\mathbb{N}$ does not have a LUB
2. (Live Variables) $(D, \sqsubseteq) = (2^{Var_c}, \subseteq)$ is a complete lattice with $\perp = \emptyset$
3. (Available Expressions) $(D, \sqsubseteq) = (2^{CExp_c}, \supseteq)$ is a complete lattice with $\perp = CExp_c$

Order-Theoretic Foundations: The Domain

Duality in Complete Lattices

- Dual concept of least upper bound: greatest lower bound
- **Definitions:**
  - An element $d \in D$ is called a lower bound of $S \subseteq D$ if $d \sqsubseteq s$ for every $s \in S$ (notation: $d \sqsubseteq S$).
  - A lower bound d is called greatest lower bound (GLB) or infimum of S if $d' \sqsubseteq d$ for every lower bound d' of S (notation: $d = \bigsqcap S$).
- **Examples:**
  - (Live Variables) $(D, \sqsubseteq) = (2^{Var_c}, \subseteq)$, $\bigsqcap\{V_1, \ldots, V_n\} = \bigcap\{V_1, \ldots, V_n\}$
  - (Available Expressions) $(D, \sqsubseteq) = (2^{CExp_c}, \supseteq)$, $\bigsqcap\{A_1, \ldots, A_n\} = \bigcup\{A_1, \ldots, A_n\}$
- **Lemma:** the following are equivalent:
  - $(D, \sqsubseteq)$ is a complete lattice (i.e., every subset of D has a least upper bound)
  - Every subset of D has a greatest lower bound
- **Corollary:** every complete lattice has a greatest element $\top := \bigsqcap \emptyset$

Chains

Chains are generated by the approximation of the analysis information in the fixpoint iteration.

Definition 3.7 (Chain)

Let $(D, \sqsubseteq)$ be a partial order.

- A subset $S \subseteq D$ is called a chain in D if, for every $d_1, d_2 \in S$,
$$d_1 \sqsubseteq d_2 \text{ or } d_2 \sqsubseteq d_1$$
(that is, S is a totally ordered subset of D).
- $(D, \sqsubseteq)$ has finite height if all chains are finite. In this case, its height is
$$\max\{|S| \mid S \text{ chain in } D\} - 1.$$

Example 3.8

1. Every $S \subseteq \mathbb{N}$ is a chain in $(\mathbb{N}, \leq)$ (which is of infinite height)
2. $\{\emptyset, \{0\}, \{0, 1\}, \{0, 1, 2\}, \ldots\}$ is a chain in $(2^{\mathbb{N}}, \subseteq)$
3. $\{\{0\}, \{1\}, \{2\}\}$ is not a chain in $(2^{\mathbb{N}}, \subseteq)$

The Ascending Chain Condition I

Termination of fixpoint iteration is guaranteed by the following condition.

Definition 3.9 (Ascending Chain Condition)

- A sequence $(d_i)_{i \in \mathbb{N}}$ is called an ascending chain in D if $d_i \sqsubseteq d_{i+1}$ for each $i \in \mathbb{N}$.
- A partial order $(D, \sqsubseteq)$ satisfies the Ascending Chain Condition (ACC) if each ascending chain $d_0 \sqsubseteq d_1 \sqsubseteq \ldots$ eventually stabilises, i.e., there exists $n \in \mathbb{N}$ such that $d_n = d_{n+1} = \ldots$

Notes:

- The finite height property implies ACC, but not vice versa (as there might be non-stabilising descending chains – see next slide)
- The complete lattice and ACC properties are orthogonal (see next slide)

The Ascending Chain Condition II

Example 3.10

1. $(\mathbb{N}, \leq)$ does not satisfy ACC and is of infinite height (and not a complete lattice)
2. $(\mathbb{Z}_{\leq 0}, \leq)$ satisfies ACC but is of infinite height (and not a complete lattice)
3. $(\mathbb{Z} \cup \{-\infty, +\infty\}, \leq)$ (where $-\infty \leq z \leq +\infty$ for all $z \in \mathbb{Z}$) is a complete lattice but does not satisfy ACC
4. $(\{\emptyset, \{0\}, \{1\}\}, \subseteq)$ satisfies ACC but is not a complete lattice
5. (Live Variables) $(2^{Var_c}, \subseteq)$ is a complete lattice satisfying ACC and is of finite height (since Var_c [unlike Var] is finite)
6. (Available Expressions) $(2^{CExp_c}, \supseteq)$ is a complete lattice satisfying ACC and is of finite height (since $CExp_c$ [unlike $AExp$] is finite)

Domain requirements for dataflow analysis

$(D, \sqsubseteq)$ must be a complete lattice satisfying ACC