Static Program Analysis

Lecture 3: Dataflow Analysis II (Algebraic Foundations)

Summer Semester 2018

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ss-18/spa/

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Recap: Dataflow Analysis

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:

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

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

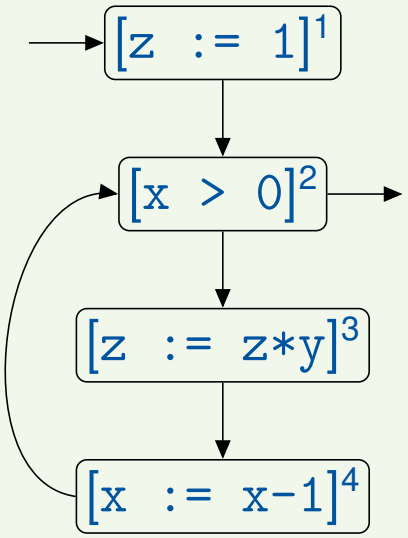
Representing Control Flow

Example

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

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

Visualisation by (control) flow graph:



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

Example (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
end
```

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

Formalising Available Expressions Analysis

- Given $a \in AExp$, $b \in BExp$, $c \in Cmd$,
  - Var_a/Var_b/Var_c denotes the set of all variables occurring in a/b/c
  - $CExp_b$/$CExp_c$ denote the sets of all complex arithmetic expressions occurring in b/c
- An expression a is killed in a block B if any of the variables in a is modified in B
- Formally: $\text{kill}_{\text{AE}} : Blk_c \to 2^{CExp_c}$ is defined by

$$\begin{aligned} \text{kill}_{\text{AE}}([\texttt{skip}]^l) &:= \emptyset \\ \text{kill}_{\text{AE}}([x := a]^l) &:= \{a' \in CExp_c \mid x \in Var_{a'}\} \\ \text{kill}_{\text{AE}}([b]^l) &:= \emptyset \end{aligned}$$

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

$$\begin{aligned} \text{gen}_{\text{AE}}([\texttt{skip}]^l) &:= \emptyset \\ \text{gen}_{\text{AE}}([x := a]^l) &:= \{a \mid x \notin Var_a\} \\ \text{gen}_{\text{AE}}([b]^l) &:= CExp_b \end{aligned}$$

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

Formalising Live Variables Analysis

- A variable on the left-hand side of an assignment is killed by the assignment; tests and `skip` do not kill
- Formally: $\text{kill}_{\text{LV}} : \mathit{Blk}_c \to 2^{\mathit{Var}_c}$ is defined by

$$\begin{aligned} \text{kill}_{\text{LV}}([\texttt{skip}]^l) &:= \emptyset \\ \text{kill}_{\text{LV}}([x := a]^l) &:= \{x\} \\ \text{kill}_{\text{LV}}([b]^l) &:= \emptyset \end{aligned}$$

- Every reading access generates a live variable
- Formally: $\text{gen}_{\text{LV}} : \mathit{Blk}_c \to 2^{\mathit{Var}_c}$ is defined by

$$\begin{aligned} \text{gen}_{\text{LV}}([\texttt{skip}]^l) &:= \emptyset \\ \text{gen}_{\text{LV}}([x := a]^l) &:= \mathit{Var}_a \\ \text{gen}_{\text{LV}}([b]^l) &:= \mathit{Var}_b \end{aligned}$$

The Equation System

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

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

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

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

- Characterisation of analysis:

  flow-sensitive: results depending on order of assignments

  backward: starts in $\mathrm{final}(c)$ 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

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

where

- the set of extremal labels, E, is $\{\mathsf{init}(c)\}$ or $\mathsf{final}(c)$
- 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 $\mathsf{flow}(c)$ or $\mathsf{flow}^R(c)$ $(:= \{(l', l) \mid (l, l') \in \mathsf{flow}(c)\})$

Recap: Dataflow Analysis

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

The Fixpoint Approach

- **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

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 (interpretation: $\bigsqcup S$ is the most precise information that approximates all elements of S)

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 $A_1, \ldots, A_n \subseteq CExp_c$,
$$\bigsqcup\{A_1, \ldots, A_n\} = \bigcap\{A_1, \ldots, A_n\}$$

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,

$$\bot := \bigsqcup \emptyset$$

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 $\bot = \emptyset$
3. (Available Expressions) $(D, \sqsubseteq) = (2^{CExp_c}, \supseteq)$ is a complete lattice with $\bot = CExp_c$

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 during 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).
- The height of $(D, \sqsubseteq)$ is given by
$$\max\{|S| \mid S \text{ chain in } D\} \qquad \text{(which can be infinite)}$$

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 $(2^{\mathbb{N}}, \subseteq)$ (which is of infinite height)
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 or stabilising chains of unbounded size – see following slide)
- The complete lattice and ACC properties are orthogonal (see following 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. $(\mathbb{N} \times \mathbb{N}, \sqsubseteq)$ with $(m_1, n_1) \sqsubseteq (m_2, n_2)$ iff $m_1 = m_2$ and $n_1 \leq n_2 \leq m_1$ satisfies ACC but is of infinite height (and not a complete lattice)
4. $(\mathbb{Z} \cup \{-\infty, +\infty\}, \leq)$ (where $-\infty \leq z \leq +\infty$ for all $z \in \mathbb{Z}$) is a complete lattice but does not satisfy ACC
5. (Live Variables) $(2^{Var_c}, \subseteq)$ is a complete lattice of finite height satisfying ACC (since Var_c [unlike Var] is finite)
6. (Available Expressions) $(2^{CExp_c}, \supseteq)$ is a complete lattice of finite height satisfying ACC (since $CExp_c$ [unlike $AExp$] is finite)

Domain requirements for dataflow analysis

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

Monotonicity of Functions

Monotonicity of transfer functions excludes “oscillating behaviour” in fixpoint iteration.

Definition 3.11 (Monotonicity)

Let $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ be partial orders, and let $\Phi : D \to D'$. Φ is called monotonic (w.r.t. $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$) if, for every $d_1, d_2 \in D$,

$$d_1 \sqsubseteq d_2 \implies \Phi(d_1) \sqsubseteq' \Phi(d_2).$$

Example 3.12

1. Let $T := \{S \subseteq \mathbb{N} \mid S \text{ finite}\}$. Then $\Phi_1 : T \to \mathbb{N} : S \mapsto \sum_{n \in S} n$ is monotonic w.r.t. $(2^{\mathbb{N}}, \subseteq)$ and $(\mathbb{N}, \leq)$.
2. $\Phi_2 : 2^{\mathbb{N}} \to 2^{\mathbb{N}} : S \mapsto \mathbb{N} \setminus S$ is not monotonic w.r.t. $(2^{\mathbb{N}}, \subseteq)$ (since, e.g., $\emptyset \subseteq \mathbb{N}$ but $\Phi_2(\emptyset) = \mathbb{N} \not\subseteq \Phi_2(\mathbb{N}) = \emptyset$).
3. (Live Variables) $(D, \sqsubseteq) = (D', \sqsubseteq') = (2^{Var_c}, \subseteq)$
   Each transfer function $\varphi_{l'}(V) := (V \setminus \text{kill}_{\text{LV}}(B^{l'})) \cup \text{gen}_{\text{LV}}(B^{l'})$ is obviously monotonic
4. (Available Expressions) $(D, \sqsubseteq) = (D', \sqsubseteq') = (2^{CExp_c}, \supseteq)$ ditto

Fixpoints

Definition 3.13 (Fixpoint)

Let D be some domain, $d \in D$, and $\Phi : D \to D$. If

$$\Phi(d) = d$$

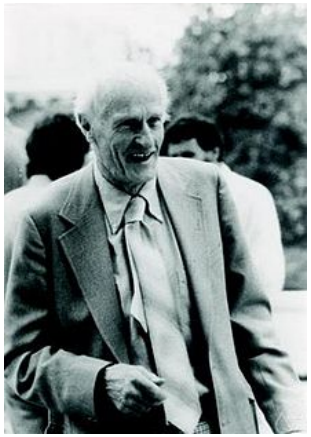
then d is called a fixpoint of Φ.

Example 3.14

The (only) fixpoints of $\Phi : \mathbb{N} \to \mathbb{N} : n \mapsto n^2$ are 0 and 1

Algebraic Foundations: The Function

The Fixpoint Theorem I



Stephen Kleene (1909–1994)

Theorem 3.15 (Fixpoint Theorem by Kleene)

Let $(D, \sqsubseteq)$ *be a complete lattice satisfying ACC and* $\Phi : D \to D$ *monotonic. Then*

$$\mathrm{fix}(\Phi) := \bigsqcup \{ \Phi^k(\bot) \mid k \in \mathbb{N} \}$$

is the least fixpoint of Φ *where* $\Phi^0(d) := d$ *and* $\Phi^{k+1}(d) := \Phi(\Phi^k(d))$.

Function requirements for dataflow analysis

All transfer functions must be a monotonic

The Fixpoint Theorem II

The proof of Theorem 3.15 requires the following lemma.

Lemma 3.16

Let $(D, \sqsubseteq)$ be a complete lattice satisfying ACC, $S \subseteq D$ a chain, and $\Phi : D \to D$ monotonic. Then

$$\Phi(\bigsqcup S) = \bigsqcup \Phi(S)$$

Proof (Lemma 3.16).

on the board □

Proof (Theorem 3.15).

on the board □

Remark: $(\Phi^k(\bot))_{k \in \mathbb{N}}$ is obviously an ascending chain which (by ACC) stabilises at some $k_0 \in \mathbb{N}$ with $\text{fix}(\Phi) = \Phi^{k_0}(\bot)$ (where k_0 bounded by height of $(D, \sqsubseteq)$)