Slicing for Modern Program Structures: a Theory for Eliminating Irrelevant Loops

Torben Amtoft

Kansas State University

May 2008
Concept: remove program points irrelevant for slicing criterion.

Method: compute dependencies between nodes in CFG.

Applications: include
- compiler optimizations
- debugging
- model checking
- protocol understanding
Theoretical foundation for approach which

- handles arbitrary CFGs, including those that
  - are irreducible (as for state charts)
  - have no unique end node (as for reactive systems)
Theoretical foundation for approach which
- handles arbitrary CFGs, including those that
  - are irreducible (as for state charts)
  - have no unique end node (as for reactive systems)

Ranganath et al [ESOP’05 & TOPLAS’07] were
- first to handle zero (or more than one) end nodes, by generalizing previous notions of control dependency.
- reasoned about termination-preserving slicing:
  - correctness expressed as (weak) bisimulation
Challenge

Theoretical foundation for approach which

- handles arbitrary CFGs, including those that
  - are irreducible (as for state charts)
  - have no unique end node (as for reactive systems)

Ranganath et al [ESOP’05 & TOPLAS’07] were

- first to handle zero (or more than one) end nodes, by generalizing previous notions of control dependency.
- reasoned about termination-preserving slicing:
  - correctness expressed as (weak) bisimulation
  - but slice could be big as must include all nodes that influence guards of potential loops

- produces slices that are manageable
Contributions

- Notion of control dependence that works for arbitrary CFGs, including those with zero end nodes
- generates slices that are likely to be manageable
- correctness criterion: behavior of original program is prefix of behavior of sliced program
- this is expressed using simulation
- correctness proof easily fits conference page limits
Data Dependence

**Source**: final value of $x = 6$

- $w := 7$
- $y := 4$
- $z := 8$
- $x := y + 2$
Data Dependence

Source: final value of $x = 6$

$e$ is data dependent on $b$, written $b^{dd} ightarrow e$, since $y$ is

- used in $e$, defined in $b$, not defined in between
Data Dependence

**Source:** final value of $x = 6$

\[
\begin{align*}
  a & : w := 7 \\
  b & : y := 4 \\
  c & : z := 8 \\
  e & : x := y + 2
\end{align*}
\]

$e$ is data dependent on $b$, written $b \xrightarrow{dd} e$, since $y$ is

- used in $e$, defined in $b$, not defined in between

**Slice:**

\[
\begin{align*}
  a & : \text{skip} \\
  b & : y := 4 \\
  c & : \text{skip} \\
  e & : x := y + 2
\end{align*}
\]
Data Dependence

Source: final value of $x = 6$

$w := 7$ → $y := 4$ → $z := 8$ → $x := y + 2$

$e$ is data dependent on $b$, written $b^{dd} \rightarrow e$, since $y$ is

▶ used in $e$, defined in $b$, not defined in between

Slice: final value of $x = 6$

$w := 7$ → $y := 4$ → $x := y + 2$
Control Dependence

Source: final value of $x = 6$ if $w = 5$, else 2
Control Dependence

**Source:** final value of $x = 6$ if $w = 5$, else $2$

\[
\begin{align*}
\text{a} & \quad w = 5? \\
\text{b} & \quad y := 4 \\
\text{c} & \quad z := 8 \\
\text{e} & \quad x := y + 2
\end{align*}
\]

$b \xrightarrow{dd} e$ and $b$ is control dependent on $a$, $a \xrightarrow{cd} b$, since

- $a$ is not strictly postdominated by $b$: path $[a..e] \not
\supset b$
- in $[a..b]$, all nodes postdominated by $b$ (vacuously)
Slicing for Modern Program Structures: a Theory for Eliminating Irrelevant Loops

Torben Amtoft

Introduction
Standard Definitions
Proposed Solution
Technical Results
Conclusion

Control Dependence

Source: final value of $x = 6$ if $w = 5$, else 2

$$w = 5?\quad \rightarrow \quad y := 4\quad \rightarrow \quad z := 8\quad \rightarrow \quad x := y + 2$$

$b \xleftrightarrow{dd} e$ and $b$ is control dependent on $a$, $a \xleftrightarrow{cd} b$, since

- $a$ is not strictly postdominated by $b$: path $[a..e] \not\subset b$
- in $]a..b[$, all nodes postdominated by $b$ (vacuously)

Slice: final value of $x = 6$ if $w = 5$, else 2

$$w = 5?\quad \rightarrow \quad y := 4\quad \rightarrow \quad \text{skip}\quad \rightarrow \quad x := y + 2$$
Weak Control Dependence

Source: final value of $x = 6$ or $\perp$

\[
\begin{align*}
\text{a} & : w := \ldots \\
\text{b} & : w = 5? \\
\text{c} & : y := 4 \\
\text{e} & : x := y + 2
\end{align*}
\]
Weak Control Dependence

**Source:** final value of \( x = 6 \) or \( \perp \)

\[
\begin{align*}
\textbf{a} & \quad w := \ldots \\
\textbf{b} & \quad w = 5? \\
\textbf{c} & \quad y := 4 \\
\textbf{e} & \quad x := y + 2
\end{align*}
\]

\( c \xrightarrow{dd} e \) but \( b \not\xrightarrow{cd} c \), since \( c \) strictly postdominates \( b \)

**Slice:** final value of \( x = 6 \)

\[
\begin{align*}
\textbf{a} & \quad \text{skip} \\
\textbf{b} & \quad \text{true?} \\
\textbf{c} & \quad y := 4 \\
\textbf{e} & \quad x := y + 2
\end{align*}
\]
Weak Control Dependence

**Source:** final value of $x = 6$ or ⊥

\[
\begin{align*}
\text{a:} & \quad w := \ldots \\
\text{b:} & \quad w = 5? \\
\text{c:} & \quad y := 4 \\
\text{e:} & \quad x := y + 2
\end{align*}
\]

$c \xrightarrow{dd} e$ but $b \xrightarrow{cd} c$, since $c$ strictly postdominates $b$

But $c$ is **weakly control dependent** on $b$, $b \xrightarrow{wcd} c$ (though no $[b..e]$ bypasses $c$, $b$ may cause indefinite delay of $c$)

**Slice:** final value of $x = 6$ or ⊥
Weak Order Dependence

**Goal**: a dependence relation that
- does not mention “end node”
- allows to slice away irrelevant loops
Weak Order Dependence

**Goal:** a dependence relation that

- does not mention “end node”
- allows to slice away irrelevant loops

```
x := x + 1
```

Observable behavior: \(1,2,3,\ldots\) if \(w > 7\), else \(0,1,2,\ldots\)
Weak Order Dependence

**Goal:** a dependence relation that
- does not mention “end node”
- allows to slice away irrelevant loops

Observable behavior: 1,2,3,... if \( w > 7 \), else 0,1,2,...
Thus \( a \) should be in slice set, but hard to see how \( a \rightarrow b \) or \( a \rightarrow e \) since all paths from \( a \) hit \( b \) and \( e \).
Weak Order Dependence

**Goal:** A dependence relation that
- does not mention “end node”
- allows to slice away irrelevant loops

Observable behavior: 1, 2, 3, ... if \( w > 7 \), else 0, 1, 2, ...

Ternary relation: \( e \) & \( b \) weakly order dependent on \( a \), \( a \overset{wod}{\rightarrow} b, e \):
- Path \([a..b] \not
\ne\) and path \([a..e] \not
\ne\)
- \( a \) has successor \( x \) such that either
  - \( b \) is reachable from \( x \) and all \([x..e]\) contain \( b \), or
  - \( e \) is reachable from \( x \) and all \([x..b]\) contain \( e \).
Example: WOD for Conditionals

Source: observable behavior of the form 1,1,2,2,2,3,4,4,…
Example: WOD for Conditionals

**Source:** observable behavior of the form 
1,1,2,2,2,3,4,4,…

$$b \xrightarrow{dd} e \text{ and } c \xrightarrow{dd} a \xrightarrow{wod} b, e$$
Example: WOD for Conditionals

Source: observable behavior of the form
1,1,2,2,2,3,4,4,…

\[ w > 7 \]

\[ a \quad \text{F} \]
\[ b \quad \text{T} \]
\[ c \]
\[ e \quad \text{out} \quad x \]

\[ w := \ldots \]

\[ x := x + 1 \]

\[ b \overset{dd}{\rightarrow} e \quad \text{and} \quad c \overset{dd}{\rightarrow} a \overset{\text{wod}}{\rightarrow} b, e \quad \text{since} \]

\[
\begin{align*}
&\quad \Rightarrow \text{path} \ [a..b] \not\ni e \quad \text{and} \quad \text{path} \ [a..e] \not\ni b \\
&\quad \Rightarrow a \text{ has successor } b \text{ with } [b..b] \text{ and all } [b..e] \text{ contain } b
\end{align*}
\]
Example: WOD for Relevant Loops

Source: observable behavior 6,7,8,9,10,...

\[ w := \ldots \]
\[ x := x + 1 \]
\[ x > 5? \]
\[ \text{out } x \]
Example: WOD for Relevant Loops

Source: observable behavior 6,7,8,9,10,...

Slice: observable behavior if guard **wrongly** sliced away:
- 1,2,3,4,5,... if replaced by *true*
- ε if replaced by *false*
Example: WOD for Relevant Loops

Source: observable behavior 6,7,8,9,10,…

c \xrightarrow{\text{wod}} b, e \text{ since}

- path [c..b] \not\ni e and path [c..e] \not\ni b
- c has successor b with [b..b] and all [b..e] contain b
Example: WOD for Irrelevant Loops

Source: observable behavior 1,2,3,4,... or prefix thereof

\[w := \ldots\]
\[w > 7?\]
\[x := x + 1\]
\[\text{out } x\]
Example: WOD for Irrelevant Loops

**Source:** observable behavior 1,2,3,4,... or *prefix* thereof

New guard chosen to get *closer* to next observable. **Slice:** observable behavior 1,2,3,4,...
Example: WOD for Irrelevant Loops

**Source**: observable behavior 1,2,3,4,... or prefix thereof

- \( a \) : \( w := \ldots \)
- \( b \) : \( w > 7? \)
- \( c \) : \( x := x + 1 \)
- \( e \) : \( \text{out} x \)

\( b \xrightarrow{\text{wod}} c, e \) since

- no path \([b..e]\) \( \not\ni c \)

New guard chosen to get closer to next observable.

**Slice**: observable behavior 1,2,3,4,...

- \( a \) : \( \text{skip} \)
- \( b \) : \( \text{true?} \)
- \( c \) : \( x := x + 1 \)
- \( e \) : \( \text{out} x \)
Unique Next Observable

- For the technical development, we shall assume the set of observables is closed under $dd \rightarrow$ and $wod \rightarrow$.
- Key advantage of being closed under $wod \rightarrow$: at most one next observable.
For the technical development, we shall assume the set of observables is closed under $\text{dd} \to$ and $\text{wod} \to$.

Key advantage of being closed under $\text{wod} \to$: at most one next observable.
Weak Simulation

- Slicing Correctness expressed using weak simulation
- A weak simulation relates source states to slice states
- A program state is an node and a store

\[
\sigma_1 \xrightarrow{\phi} \sigma_2
\]

\[
\sigma''_1 \xrightarrow{\sigma_1'} \sigma_1
\]
Weak Simulation

- Slicing Correctness expressed using weak simulation
- A weak simulation relates source states to slice states
- A program state is an node and a store

\[ \sigma_1 \xrightarrow{\phi} \sigma_2 \]

\[ \sigma_1'' \xrightarrow{\text{same node}} \sigma_1 \]

\[ \sigma_2'' \xrightarrow{\phi} \sigma_2 \]
A variable $x$ is relevant at $n$ if

- $x$ is used at an observable node $q$
- there exists a path $[n..q]$ along which $x$ is not defined
A variable $x$ is **relevant** at $n$ if

- $x$ is **used** at an **observable** node $q$
- there exists a path $[n..q]$ along which $x$ is **not defined**

**Facts:**

- this definition is **independent** of whether one considers the source or the slice
- if two nodes have the **same next observable**, they have the **same relevant variables**
Defining a Concrete Weak Simulation

We define \((n_1, \sigma_1) \leq R (n_2, \sigma_2)\) if

- \(n_1\) and \(n_2\) have the same next observable
- \(\sigma_1\) and \(\sigma_2\) agree on the relevant variables
Defining a Concrete Weak Simulation

We define \( (n_1, \sigma_1) \ R (n_2, \sigma_2) \) if

- \( n_1 \) and \( n_2 \) have the same next observable
- \( \sigma_1 \) and \( \sigma_2 \) agree on the relevant variables

Example:
Correctness

Theorem

- $R$ is a weak simulation.

Thus any observable action by the source can be matched by the slice, but **not** necessarily vice versa.
Conservative Extension

For CFGs with a unique end node
the closures of $\text{wod}$ and $\text{cd}$ coincide
For CFGs with a unique end node which is part of slicing criterion the closures of $\rightarrow$ and $\cdot \rightarrow$ coincide.
Conservative Extension

For CFGs with a unique end node which is part of slicing criterion the closures of $\text{wod}$ and $\text{cd}$ coincide.
Related Work

Ball & Horwitz, 1993
- assumes unique end node
- correctness expressed using pointwise history
- one history is allowed to be prefix of other

Hatcliff et al, HOSC 2000
- assumes unique end node
- used for model checking, implying
  - sequencing of observables must be preserved
  - liveness properties must be preserved
- detailed correctness proof, assuming termination of both source and slice

Hatcliff et al, SAS 1999
- considers multi-threading
- proposes bisimulation as correctness property
- does not work out the details
Related Work, II

Ranganath et al, ESOP’05 & TOPLAS’07

- considers arbitrary CFGs
- proposes a variety of dependencies
- correctness is
  - only worked out for termination-sensitive dependency
  - expressed using bisimulation
Assessment

Foundation for slicing which

- handles CFGs that arise in modeling reactive systems
- produces manageable slices
- allows crisp correctness proof
Assessment

Foundation for slicing which
- handles CFGs that arise in modeling reactive systems
- produces manageable slices
- allows crisp correctness proof

Open question: Is (bi)simulation a strictly stronger correctness criterion than “pointwise” (bi)simulation?
If so, we might modify $\xrightarrow{\text{wod}}$ to allow for independent assignments to be swapped.
Future Work

- Algorithm computing $\xrightarrow{wod}$
  (cf. Ranganath et al [TOPLAS’07])
- Interprocedural analysis
- Multi-threading
- Connect to information flow analysis
  (Hammer & Krinke & Snelting)