Why3 a dit : gardez le contrôle en toute situation

*Jean-Christophe Léchenet, Nikolai Kosmatov, Pascale Le Gall*

26 janvier 2018
JFLA - Banyuls-sur-Mer
Arbitrary control dependence

Context: static backward program slicing
Definitions of control dependence

Danicic’s algorithm
Description
Illustration
Formalization in Coq

A new optimized algorithm
Presentation
Formalization in Why3
Experiments

Conclusion
Plan

Arbitrary control dependence
   Context: static backward program slicing
   Definitions of control dependence

Danicic’s algorithm
   Description
   Illustration
   Formalization in Coq

A new optimized algorithm
   Presentation
   Formalization in Why3
   Experiments

Conclusion
Plan

Arbitrary control dependence

Context: static backward program slicing
Definitions of control dependence

Danicic’s algorithm
Description
Illustration
Formalization in Coq

A new optimized algorithm
Presentation
Formalization in Why3
Experiments

Conclusion
Definition

**Static backward slicing** (introduced by Weiser in 1981)

- simplifies a given program $p$ but preserves the behavior w.r.t. a point of interest $C$ (**slicing criterion**, typically a statement)
- removes irrelevant statements that do not impact $C$
- produces a simplified program $q$ (**slice**)
Example: divisibility test

euclidean division of $a$ by $b$

1 : quo = 0;
2 : r = a;
3 : while (b <= r) {
   4 : quo = quo + 1;
   5 : r = r - b;
}

is the remainder equal to 0 ?

6 : if (r != 0) {
   7 : res = 0;
} else {
   8 : res = 1;
}

Original program $p$

Slice $q$ w.r.t. line 8

Jean-Christophe Léchenet  Why3 a dit : gardez le contrôle en toute situation  26 janvier 2018  6 / 30
Example: divisibility test

1: quo = 0;
2: r = a;
3: while (b <= r) {
   4:   quo = quo + 1;
   5:   r = r - b;
}
6: if (r != 0) {
   7:   res = 0;
   } else {
8:   res = 1;
   }

Original program $p$          Slice $q$ w.r.t. line 8
Example: divisibility test

1: quo = 0;
2: r = a;
3: while (b <= r) {
   4:   quo = quo + 1;
   5:   r = r - b;
}
6: if (r != 0) {
   7:   res = 0;
} else {
   8:   res = 1;
}

Original program $p$

Slice $q$ w.r.t. line 8
Example: divisibility test

```
1 : quo = 0;

2 : r = a;
3 : while (b <= r) {
   4 : quo = quo + 1;
   5 : r = r - b;
}
6 : if (r != 0) {
   7 : res = 0;
} else {
   8 : res = 1;
}
```

Original program $p$

```
2 : r = a;
3 : while (b <= r) {
   5 : r = r - b;
}
6 : if (r != 0) {
   8 : res = 1;
} else {
```

Slice $q$ w.r.t. line 8
Plan

**Arbitrary control dependence**

Context: static backward program slicing

Definitions of control dependence

**Danicic’s algorithm**

Description

Illustration

Formalization in Coq

**A new optimized algorithm**

Presentation

Formalization in Why3

Experiments

Conclusion
On a concrete structured language

if \((l: b)\) {
  ...
  \(l_{\text{then}}\): stmt;
  ...
} else {
  ...
  \(l_{\text{else}}\): stmt;
  ...
}

while \((l: b)\) {
  ...
  \(l_{\text{body}}\): stmt;
  ...
}
On a control flow graph

Using post-dominance (for ex. [Ferrante et al., 1987])

- $v$ is control-dependent on $u$ iff $u$ has two children $u_1$ and $u_2$ such that $u_1$ is post-dominated by $v$, but not $u_2$
On a control flow graph

Using post-dominance (for ex. [Ferrante et al., 1987])

- \( v \) is **control-dependent** on \( u \) iff \( u \) has two children \( u_1 \) and \( u_2 \) such that \( u_1 \) is post-dominated by \( v \), but not \( u_2 \)

```
start

quo=0

r=a

while (b<=r)

quo=quo+1

r=r-b

if (r!=0)

res=0

u

res=1

u_1

end
```

Jean-Christophe Léchenet  Why3 a dit : gardez le contrôle en toute situation 26 janvier 2018 9 / 30
On a finite directed graph

- Elegant generalization of Danicic et al. in 2011
- A subset $V'$ is closed under weak control dependence (or weakly control-closed) iff every node reachable from $V'$ has at most one first-reachable node (observable) in $V'$.
On a finite directed graph

- Elegant generalization of Danicic et al. in 2011
- A subset $V'$ is closed under weak control dependence (or weakly control-closed) iff every node reachable from $V'$ has at most one first-reachable node (observable) in $V'$.
- $\text{weak control-closure}(V') = V' \cup \{ \text{all the vertices both reachable from } V' \text{ and } V'\text{-weakly deciding} \}$
- $V'$-weakly deciding $= \text{all the nodes giving rise to two non-trivial paths reaching } V' \text{ that share no vertex except their origin.}$

\[ u \rightarrow v \rightarrow v_1 \rightarrow \cdots \rightarrow x \rightarrow y \rightarrow \cdots \rightarrow w \]

\[ \text{obs}(x) = \{x\} \]
\[ \text{obs}(w) = \{x\} \]
\[ \text{obs}(u) = \{v_1, v_2\} \]
\[ \text{obs}(v) = \{v_1, v_2\} \]
Running example

\[ V' = \{u_1, u_3\} \]
Running example

\[ V' = \{ u_1, u_3 \} \]
Running example

\[ V' = \{ u_1, u_3 \} \]
Running example

\[ V' = \{u_1, u_3\} \]
Running example

\[ V' = \{ u_1, u_3 \} \]

Closure: \( \{ u_0, u_1, u_2, u_3, u_4, u_6 \} \)
Plan

Arbitrary control dependence
   Context: static backward program slicing
   Definitions of control dependence

Danicic’s algorithm
   Description
   Illustration
   Formalization in Coq

A new optimized algorithm
   Presentation
   Formalization in Why3
   Experiments

Conclusion
Plan

Arbitrary control dependence
  Context: static backward program slicing
  Definitions of control dependence

Danicic’s algorithm
  Description
  Illustration
  Formalization in Coq

A new optimized algorithm
  Presentation
  Formalization in Why3
  Experiments

Conclusion
Idea

• Iterative algorithm
• Predicate $H(u, V')$ such that:
  (H1) If $H(u, V')$ then $u$ is $V'$-weakly deciding and reachable from $V'$
  (H2) If there is no node $u$ satisfying $H(u, V')$, then there is no
        $V'$-weakly deciding vertex reachable from $V'$

**$H$’s definition**

$H(u, V')$: $u$ is reachable from $V'$, $|\text{obs}(u)| \geq 2$ and one of its
children $v$ satisfies $|\text{obs}(v)| = 1$. 
Danicic’s method to compute weak control closure

begin
  \( W \leftarrow V' \); \\
  \textbf{while} there exists a node \( u \) satisfying \( H(u, W) \) in \( V \) \textbf{do} \\
  \hspace{1em} choose such a node \( u \); \\
  \hspace{1em} \( W \leftarrow W \cup \{u\} \) \\
  \textbf{end} \\
\textbf{return} \( W \)
end

Key ideas:

- At each iteration, the weak control-closure of \( W \) is equal to the weak-control closure of \( V' \) (due to (H1)).
- At the end, \( W \) is weakly-control closed (due to (H2)).
Plan

Arbitrary control dependence
  Context: static backward program slicing
  Definitions of control dependence

Danicic’s algorithm
  Description
  Illustration
  Formalization in Coq

A new optimized algorithm
  Presentation
  Formalization in Why3
  Experiments

Conclusion
Danicic’s algorithm on an example

\[ V' = \{u_1, u_3\} \]
Danicic’s algorithm on an example

\[ V' = \{ u_1, u_3 \} \]
Danicic’s algorithm on an example

\[ V' = \{u_1, u_3\} \]
Danicic’s algorithm on an example

\[ V' = \{ u_1, u_3 \} \]
Danicic’s algorithm on an example

\[ \mathcal{V}' = \{ u_1, u_3 \} \]
Danicic’s algorithm on an example

\[ V' = \{ u_1, u_3 \} \]
Danicic’s algorithm on an example

\[ V' = \{ u_1, u_3 \} \]
Danicic’s algorithm on an example

\[ V' = \{ u_1, u_3 \} \]
Danicic’s algorithm on an example

\[ V' = \{ u_1, u_3 \} \]

Closure: \( \{ u_0, u_1, u_2, u_3, u_4, u_6 \} \)
Plan

Arbitrary control dependence
  Context: static backward program slicing
  Definitions of control dependence

Danicic’s algorithm
  Description
  Illustration
  Formalization in Coq

A new optimized algorithm
  Presentation
  Formalization in Why3
  Experiments

Conclusion
A few words about the formalization in Coq

- A subset of Danicic’s theory was formalized in Coq
- Danicic’s algorithm was implemented and proved correct
- Size: 4000 loc of spec, 8000 loc of proof
- A Coq library à la OCamlgraph was missing
Limitations of Danicic’s algorithm

A few small optimizations are possible:

- At each iteration, add all the nodes satisfying $H(u, W)$ instead of just one
- Weakening $H$: 2 and 1 are arbitrary, what is important is that $1 \leq |\text{obs}(v)| < |\text{obs}(u)|$.

More fundamentally, Danicic’s algorithm does not take advantage of previous iterations to speed up the following ones.
Plan

Arbitrary control dependence
Context: static backward program slicing
Definitions of control dependence

Danicic’s algorithm
Description
Illustration
Formalization in Coq

A new optimized algorithm
Presentation
Formalization in Why3
Experiments

Conclusion
Plan

Arbitrary control dependence
  Context: static backward program slicing
  Definitions of control dependence

Danicic’s algorithm
  Description
  Illustration
  Formalization in Coq

A new optimized algorithm
  Presentation
  Formalization in Why3
  Experiments

Conclusion
The optimized algorithm

- Again an iterative algorithm: start with $W = V'$ and make $W$ grow
- Each vertex is labeled with a node in $W$ which is a good candidate for an observable, but sometimes is not
- This labeling survives the iterations and can be reused
- At the end, $W$ is the weak control-closure of $V'$ and each node is labeled with its observable in the closure
The optimized algorithm on an example

\[ V' = \{u_1, u_3\} \]
The optimized algorithm on an example

\[ V' = \{u_1, u_3\} \]
The optimized algorithm on an example

\[ V' = \{ u_1, u_3 \} \]

Propagation of \( u_3 \)
The optimized algorithm on an example

\[ V' = \{u_1, u_3\} \]

After propagation of \( u_3 \)
The optimized algorithm on an example

\[ V' = \{ u_1, u_3 \} \]

Propagation of \( u_0 \)
The optimized algorithm on an example

\[ V' = \{ u_1, u_3 \} \]

After propagation of \( u_0 \)
The optimized algorithm on an example

\[ V' = \{ u_1, u_3 \} \]

After propagation of \( u_2 \)
The optimized algorithm on an example

\[ V' = \{ u_1, u_3 \} \]
The optimized algorithm on an example

\[ V' = \{u_1, u_3\} \]

After propagation of \( u_4 \)
The optimized algorithm on an example

\[ V' = \{u_1, u_3\} \]

After propagation of \( u_6 \)
The optimized algorithm on an example

\[ V' = \{ u_1, u_3 \} \]

Closure: \( \{ u_0, u_1, u_2, u_3, u_4, u_6 \} \)
Plan

Arbitrary control dependence
- Context: static backward program slicing
- Definitions of control dependence

Danicic’s algorithm
- Description
- Illustration
- Formalization in Coq

A new optimized algorithm
- Presentation
- Formalization in Why3
- Experiments

Conclusion
A few words about the formalization in Why3

The Why3 development is split into two parts:

- a small part of weak control dependence’s theory (80 loc)
  - everything proved
  - except one lemma is admitted (but is proved in the Coq formalization)
- the new algorithm (250 loc)
  - split into 4 functions
  - a lot of proofs are automatic
  - the preservations of the main invariants were done in Coq
Plan

Arbitrary control dependence
- Context: static backward program slicing
- Definitions of control dependence

Danicic’s algorithm
- Description
- Illustration
- Formalization in Coq

A new optimized algorithm
- Presentation
- Formalization in Why3
- Experiments

Conclusion
Experiments

- Both algorithms were implemented in OCaml using OCamlGraph.
- They were run on randomly generated graphs.
- Checked against the Coq extraction on small graphs.
Plan

Arbitrary control dependence
  Context: static backward program slicing
  Definitions of control dependence

Danicic’s algorithm
  Description
  Illustration
  Formalization in Coq

A new optimized algorithm
  Presentation
  Formalization in Why3
  Experiments

Conclusion
Conclusion:

- Formalization in Coq of an elegant theory of control dependence on finite directed graphs and of an algorithm computing closure under control dependence (Danicic et al., 2011)
- Design of an optimization of this algorithm
- Proof in Why3 of this new algorithm
- Experiments confirm the new algorithm outperforms Danicic’s method

Future work:

- Integrate this work in a theory of program slicing
- Weak control dependence $\rightarrow$ strong control dependence
Weak control-closure on euclidean division

start

quo=0

r=a

while (b<=r)

quo=quo+1

r=r-b

if (r!=0)

res=0

end

res=1
Weak control-closure on euclidean division

```
start

quo=0

r=a

while (b<=r)

quo=quo+1

r=r-b

if (r!=0)

res=0

end

res=1
```

Jean-Christophe Léchenet Why3 a dit : gardez le contrôle en toute situation 26 janvier 2018 31 / 30
Graph theory

<table>
<thead>
<tr>
<th></th>
<th>Alt-Ergo (1.30)</th>
<th>CVC4 (1.5)</th>
<th>Coq (8.6.1)</th>
<th>Eprover (2.0)</th>
<th>Z3 (4.5.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>10</td>
<td>14</td>
<td>4</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Min time (s)</td>
<td>0</td>
<td>0,02</td>
<td>0,27</td>
<td>0,01</td>
<td>0</td>
</tr>
<tr>
<td>Max time (s)</td>
<td>0,01</td>
<td>0,67</td>
<td>0,37</td>
<td>0,44</td>
<td>0</td>
</tr>
<tr>
<td>Avg time (s)</td>
<td>0,01</td>
<td>0,083</td>
<td>0,3</td>
<td>0,093</td>
<td>N/A</td>
</tr>
</tbody>
</table>

+ 1 axiom (but proved in the Coq formalization)
## Algorithm

<table>
<thead>
<tr>
<th></th>
<th>Alt-Ergo (1.30)</th>
<th>CVC4 (1.5)</th>
<th>Coq (8.6.1)</th>
<th>Eprover (2.0)</th>
<th>Z3 (4.5.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>233</td>
<td>12</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Min time (s)</td>
<td>0,01</td>
<td>0,08</td>
<td>0,32</td>
<td>0,08</td>
<td>0,34</td>
</tr>
<tr>
<td>Max time (s)</td>
<td>3,96</td>
<td>0,83</td>
<td>0,76</td>
<td>2,35</td>
<td>3,18</td>
</tr>
<tr>
<td>Avg time (s)</td>
<td>0,18</td>
<td>0,46</td>
<td>0,48</td>
<td>0,72</td>
<td>1,76</td>
</tr>
</tbody>
</table>