Implementing Influence Analysis using Parameterised Boolean Equation Systems

María del Mar Gallardo, Christophe Joubert and Pedro Merino

University of Málaga / GISUM
http://www.lcc.uma.es/gisum

November 18th, 2006
2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation
Introduction

Context

Definition

A data flow analysis (DFA) is a static analysis of the definition and usage of program data, e.g., variables, expressions, definitions.

- Interest: program optimizations, e.g., dead code elimination, and state space reductions during program verification, e.g., by avoiding enumerating all possible data values

Definition

An influence variables analysis (IA) is a data flow analysis technique that detect variables that do not influence the properties to be verified.

- Interest: program state space reduction during enumerative verification
Classical influence analysis (for a given property)

C program example

```c
void fact ( int n ) {
    int x = n;
    int y = 1;
    while ( x > 0 ) {
        y = x * y;
        x = x - 1;
    }
}
```

⇒ IA algorithm based on IA flow equations.

⇓ Variable list on program points enjoying the property, e.g., reachability.
Motivation

Combine two approaches to fight state space explosion

1. Influence analysis
2. On-the-fly verification
   - Incremental state space construction

For each data flow analysis problem...

- Influence variables
- Live variables
- Dead variables
- Very busy expressions
- Available expressions
- Reachable definitions, ...

... only one solution:

- Translation to a boolean equation system (BES) resolution
- Use of generic BES solver and on-the-fly annotator
Outline

1. Control flow model
2. Influence analysis as model checking and BES resolution
3. BES encoding of classical data flow analyses
4. Experimental results
5. Conclusion and future work
Outline

1. Control flow model
   - Abstract control flow graph as Labeled Transition System
   - Example: abstract model of a C program

2. Influence analysis as model checking and BES resolution

3. BES encoding of classical data flow analyses

4. Experimental results

5. Conclusion and future work
Control flow graph

Definition

A Labeled Transition System \((\text{LTS})\) is a tuple \(M = \langle S, A, T, s_0 \rangle\), where:

- \(S\) is a finite set of \textit{states}
- \(A\) is a finite set of \textit{actions}
- \(T \subseteq S \times A \times S\) is the set of labeled \textit{transitions}
- \(s_0\) is the \textit{initial state}

Definition

A control flow graph \((\text{CFG})\) is an LTS where:

- states are \textit{program counters} of the system to be analysed
- actions are the basic \textit{program instructions}, i.e., boolean expressions, assignments of program variables with arithmetical expressions, assertions, and the invisible instruction \(\tau\)
Example: control flow graph of a C program

C program example

```c
void fact ( int n ) {
    int x = n;
    int y = 1;
    while ( x > 0 ) {
        y = x * y;
        x = x - 1;
    }
}
```

Control flow graph example
Abstract control flow graph

**Definition**

An abstract control flow graph is a CFG whose actions are of the form:

\[ \vec{e} : MODIFY \quad \vec{v} : USE \quad \vec{w} : (BOOL|ASSERT) \]

- \( \vec{e} \) is the list of non-trivial expressions
- \( \vec{v} \) is the list of modified variables in the instruction
- \( \vec{w} \) is the list of used (i.e., read) variables
- two types of values, \( var \) and \( expr \), denoting the set of program variables and expressions
Example: abstract model of a C program

Control flow graph example

Abstract control flow graph

\[
\begin{align*}
&\text{MODIFY } x : \text{USE } n \\
&\text{MODIFY } y \\
&\text{x } - 1 : \text{MODIFY } x : \text{USE } x \\
&\text{x } > 0 : \text{USE } x : \text{BOOL} \\
&\text{x } > 0 : \text{USE } x : \text{BOOL} \\
&\text{x } * y : \text{MODIFY } y : \text{USE } x y
\end{align*}
\]
Outline

1. Control flow model

2. Influence analysis as model checking and BES resolution
   - Model checking (IA-Mc)
   - BES resolution (IA-BES)

3. BES encoding of classical data flow analyses

4. Experimental results

5. Conclusion and future work
Influence analysis as model checking (IA-Mc)

Abstract control flow graph

Modal formula.

Model checker.

Variable list on states satisfying the property, e.g., reachability.
Value-based alternation-free modal mu-calculus ($L^1_{\mu}$)

Syntax

\[
\phi ::= \text{false} | \text{true} | \phi_1 \lor \phi_2 | \phi_1 \land \phi_2 | \langle a \rangle \phi | [a] \phi | X(\vec{e}) \\
| \mu X(\vec{x} : \vec{t} := \vec{e}).\phi | \nu X(\vec{x} : \vec{t} := \vec{e}).\phi
\]

- Alternation-free fragment: no mutual recursion between minimal and maximal fixed point variables \[\text{Emerson-Lei-86}\]

Example

$L^1_{\mu}$ formula of Live variables analysis:

- “For each program point, which variables may be live at the exit point”

\[
\phi(v) = \mu Z. (\langle a \mid \text{used}(v, a) \rangle \text{true} \lor (\langle a \mid \neg \text{modified}(v, a) \rangle Z)
\]

[Steffen-91][Schmidt-98]
Encoding IA as value-based $L^1_\mu$ formula

Let $M = \langle S, A, T, s_0 \rangle$ be an LTS.

Reachability influence analysis

For each program point, which variables may be needed to preserve the code reachability tree at the exit from the point.

$\phi(p) = \mu Y(v : \text{var} := p). (\langle a | \text{used}(v, a) \land \text{bool}(a) \rangle \text{true} \lor \\
\langle a | \text{modified}(z, a) \land \text{used}(v, a) \rangle Y(z) \lor \\
\langle a | \neg \text{modified}(v, a) \rangle Y(v))$

where $s, s' \in S, \ a \in A, \ and \ v, z \in \text{var}$

- $\text{used}(v, a)$ is true if variable $v$ is used (i.e., read) in instruction $a$
- $\text{bool}(a)$ is true if instruction $a$ is a boolean expression
- $\text{modified}(v, a)$ is true if variable $v$ is modified on instruction $a$
Encoding IA as value-based $L^1_\mu$ formula (ctnd.)

Assertion influence analysis

- Reachability IA-Mc slightly extended:
  \[
  \langle a \mid used(v, a) \land (bool(a) \lor assert(a)) \rangle \text{ true } \cdots
  \]
- Interest: safety properties

Formula influence analysis

- Reachability IA-Mc slightly extended:
  \[
  \cdots \langle a \mid v \in \text{formula} \lor (used(v, a) \land (bool(a) \lor \text{modify_formula}(a))) \rangle \text{ true } \cdots
  \]
- Interest: temporal formulas
Influence analysis as BES resolution (IA-BES)

Abstract control flow graph

0 1 2 3 4 5

| MODIFY x : USE n | x |
| MODIFY y        |   |
| x - 1 : MODIFY x : USE x | x |
| x > 0 : USE x : BOOL |   |
| x > 0 : USE x : BOOL |   |
| x * y : MODIFY y : USE x y | x |

Boolean equation system.

⇓

⇒

BES solver + on-the-fly annotator.

⇓

Variable list on states satisfying the property, e.g., reachability.
Parameterised boolean equation system (PBES)

Definition

A **Boolean equation system** (BES) is a tuple $B = \langle x, M_1, \ldots, M_n \rangle$ s.t.:

- $x \in \mathcal{X}$ is a boolean variable, $\mathcal{X}$ a set of boolean variables
- $M_i$ are equation blocks ($i \in [1, n]$)

A **block of equations** $M_i = \{ x_{ij}^{\sigma_i} \overset{\text{op}_{ij}}{=} \{} X_{ij} \} \overset{j}{\in}[1, m_i]$ is a set of fixed point equations with sign $\sigma_i = \mu \text{ or } \nu$, where:

- $x_{ij}$ is a pure disjunctive or conjunctive formula obtained by applying a boolean operator $\text{op}_{ij} \in \{ \lor, \land \}$ to a set of variables $X_{ij} \subseteq \mathcal{X}$

Definition

A **Parameterised BES** (PBES) is a tuple $PB = \langle x (\vec{z} : \vec{t}), M_1, \ldots, M_n \rangle$,

where $x \in \mathcal{X}$ is a boolean variable parameterised by data variables in $\vec{z}$ typed by $\vec{t}$. 
Encoding IA as PBES

Let $M = \langle S, A, T, s_0 \rangle$ be an LTS.

Reachability influence analysis

For each program point, which variables may be needed to preserve the code reachability tree at the exit from the point.

$$X_{s,v} \overset{\mu}{=} \bigvee \left( \begin{array}{l}
\{ \text{true} \mid s \xrightarrow{a} s' \land \text{used}(v, a) \land \text{bool}(a) \} \cup \\
\{ X_{s',z} \mid s \xrightarrow{a} s' \land \text{modified}(z, a) \land \text{used}(v, a) \} \cup \\
\{ X_{s',v} \mid s \xrightarrow{a} s' \land \neg \text{modified}(v, a) \} \right)$$

where $s, s' \in S$, $a \in A$, and $v, z \in \text{var}$

- Interest : direct translation into PBES from value-based $L^{1}_{\mu}$ formula
Encoding \( \text{IA} \) as \( \text{PBES} \) (ctnd.)

**Assertion influence analysis**

- Reachability \( \text{IA-PBES} \) slightly extended:
  \[
  \{ \text{true} \mid s \xrightarrow{a} s' \land \text{used}(v, a) \land (\text{bool}(a) \lor \text{assert}(a)) \}\]
- Interest: safety properties

**Formula influence analysis**

- Reachability \( \text{IA-PBES} \) slightly extended:
  \[
  \{ \text{true} \mid v \in \text{formula} \lor (s \xrightarrow{a} s' \land \text{used}(v, a) \land (\text{bool}(a) \lor \text{modify_formula}(a))) \}\]
- Interest: temporal formulas
### Comparison

<table>
<thead>
<tr>
<th>Program representation</th>
<th>Classical IA</th>
<th>IA-Mc</th>
<th>IA-PBES</th>
</tr>
</thead>
<tbody>
<tr>
<td>control flow graph</td>
<td>Kripke/transition system</td>
<td>labeled transition system</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem statement</th>
<th>flow equation system over sets</th>
<th>value-based $L^1_\mu$ formula</th>
<th>PBES</th>
</tr>
</thead>
</table>

| Computation of the solution | IA algorithm (adhoc) | (global) model checker (generic) | on-the-fly BES solver + annotator (generic) |

### Example

Gcc compiler, $> 1\,000\,000$ lines of code $\Rightarrow$ program representation construction and handling is a bottleneck.
On-the-fly data flow analysis algorithm

Algorithm

- Dynamic construction and traversal of element set of interest (variables, expressions, definitions, etc.)
- Dynamic construction and traversal of program model and BES
- Use of persistent computation results between successive calls to BES solver for each state and data element of interest

Complexity results

Linear time and memory complexities, both worst case being \( O(|S|+|T|) \), given an LTS \( M = \langle S, A, T, s_0 \rangle \)
Outline

1. Control flow model

2. Influence analysis as model checking and BES resolution

3. **BES encoding of classical data flow analyses**
   - Live variables and very busy expressions analyses
   - Available expressions and reachable definitions analyses

4. Experimental results

5. Conclusion and future work
Live variables analysis

\[
\begin{align*}
X_{s,v} & \equiv \\
& \mu \left( \{ \text{true} | \ s \xrightarrow{a} s' \land used(v,a) \} \cup \\
& \{ X_{s',v} | \ s \xrightarrow{a} s' \land \neg modified(v,a) \} \right)
\end{align*}
\]

where \( s, s' \in S, \ a \in A, \ v \in \text{var} \)

- **Interest**: dead code elimination and register allocation

Very busy expressions analysis

\[
\begin{align*}
X_{s,e} & \equiv \\
& \nu \left( \{ Y_{s,e} \} \cup \{ \text{true} | \ s \xrightarrow{a} s' \land used(e,a) \} \cup \\
& \{ \text{false} | \ s \xrightarrow{a} s' \land \neg used(e,a) \land modified(e,a) \} \cup \\
& \{ X_{s',e} | \ s \xrightarrow{a} s' \land \neg modified(e,a) \land \neg used(e,a) \} \right)
\end{align*}
\]

\[
Y_{s,e} \equiv \nu \left( \{ \text{true} | \ s \xrightarrow{a} s' \} \right)
\]

where \( s, s' \in S, \ a \in A \) and \( e \in \text{expr} \)

- **Interest**: code hoisting
Available expressions analysis

\[
\begin{align*}
X_{s,e} & \overset{\nu}{=} \land \left( \{ Y_{s',e} \mid s \overset{a}{\to} s' \land \left( \neg \text{used}(e, a) \land Z_{s,e} \notin \text{computed}(Z) \right) \lor \text{modified}(e, a) \land Y_{s,e} \notin \text{computed}(Y) \} \cup \\
& \quad \{ Z_{s',e} \mid s \overset{a}{\to} s' \land \text{used}(e, a) \land \neg \text{modified}(e, a) \land Z_{s,e} \notin \text{computed}(Z) \} \cup \{ X_{s',e} \lor \text{true} \mid s \overset{a}{\to} s' \} \right) \\
Y_{s,e} & \overset{\nu}{=} \land \left( \{ Y_{s',e} \mid s \overset{a}{\to} s' \land \left( \neg \text{used}(e, a) \lor \text{modified}(e, a) \right) \} \right) \\
Z_{s,e} & \overset{\nu}{=} \land \left( \{ Z_{s',e} \mid s \overset{a}{\to} s' \land \neg \text{modified}(e, a) \} \right)
\end{align*}
\]

where \( s, s' \in S \), \( a \in A \) and \( e \in \text{expr} \)

- Interest: common sub-expressions elimination

Reachable definitions analysis

\[
\begin{align*}
X_{s,(o,v,t)} & \overset{\nu}{=} \land \left( \{ Y_{t,(o,v,t)} \mid s = o \} \cup \{ X_{s',(o,v,t)} \lor \text{true} \mid s \overset{a}{\to} s' \} \right) \\
Y_{s,(o,v,t)} & \overset{\nu}{=} \land \left( \{ Y_{s',(o,v,t)} \mid s \overset{a}{\to} s' \land \neg \text{modified}(v, a) \} \right)
\end{align*}
\]

where \( s, s', o, t \in S \), \( a \in A \) and \( v \in \text{var} \)

- Interest: constant and copy propagation
Outline

1. Control flow model
2. Influence analysis as model checking and BES resolution
3. BES encoding of classical data flow analyses
4. Experimental results
   - ANNOTATOR: a modular tool for on-the-fly data flow analysis
   - Data flow analysis of C, PROMELA, and LOTOS programs
5. Conclusion and future work
Experimental results

ANNOTATOR: a modular tool for on-the-fly data flow analysis

specification ... C program
(.lotos) (.c)

CAESAR

C2LTS

LTS (protocol)

(exploration)

caesar library
(caesar_solve library)

OPEN/CAESAR environment

LTS exploration

(static analysis encoding)

(state)

(property satisfaction)

Live variables
Dead variables
Very busy expressions
Available expressions
Reaching definitions
Reachability
Assertion
Formula

(optional input/output arguments
(--bes, --bfs, --formula, --xml, etc.)

annotating function
(.xml, .txt, implicit, etc.)

Gallardo, Joubert, Merino (UMA/GISUM)
Implementing IA using PBES
ISOLA’06 (Paphos, Cyprus)
Data flow analysis of programs

C and PROMELA examples

- 12 classical C program examples [Nielson-Nielson-Hankin-05], 2 PROMELA program examples [Camara-Gallardo-Merino-06], and 10 additional specific C program examples
- Described as Binary Coded Graphs (BCG)
- Immediate results returned by ANNOTATOR

Dekker mutual exclusion protocol in LOTOS

- http://www.inrialpes.fr/vasy/cadp/demos
- 89 lines of LOTOS, 2 processes, 9 variables, 954 states, 1908 transitions, and 17 labels
- Abstract control flow graph (25 states, 134 transitions)
- No live variables, 162 program definitions, and no very busy nor available expressions among the 9 discovered in the program

Gallardo, Joubert, Merino (UMA/GISUM) Implementing IA using PBES ISOLA’06 (Paphos, Cyprus)
Outline

1. Control flow model
2. Influence analysis as model checking and BES resolution
3. BES encoding of classical data flow analyses
4. Experimental results
5. Conclusion and future work
Conclusion and future work

Summary:

- **Abstract control flow graphs** from different programming languages
- **Encoding** of data flow and influence analyses in $L^1_{\mu}/$Bess
- **On-the-fly static analyser** ANNOTATOR using OPEN/CAESAR
- **Experiments** on numerous standard examples

Ongoing and future work:

- Further experiments and benchmarks
- **Automatic abstract matching** during verification of (C) programs
- Connection with tools extending SPIN (OCKET-Mc and $\alpha$SPIN)
- Encoding of **other static analyses** (Reset Variables)