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

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

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

$\Longrightarrow$ IA algorithm based on IA flow equations.

$\Downarrow$

$\Longleftarrow$ 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 (LTS) is a tuple $M = \langle S, A, T, s_0 \rangle$, where :

- S is a finite set of *states*
- A is a finite set of *actions*
- $T \subseteq S \times A \times S$ is the set of labeled *transitions*
- s_0 is the *initial state*

Definition

A control flow graph (CFG) is an LTS where :

- states are *program counters* of the system to be analysed
- actions are the basic *program instructions*, i.e., boolean expressions, assignments of program variables with arithmetical expressions, assertions, and the invisible instruction τ

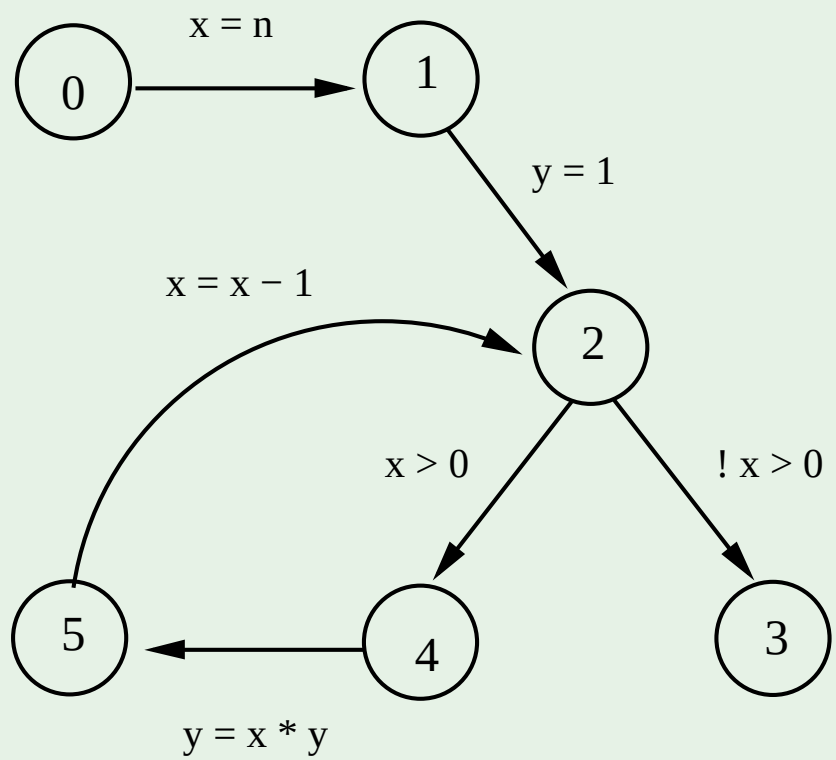
Example : control flow graph of a C program

C program example

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

$\Rightarrow$

Control flow graph example



Abstract control flow graph

Definition

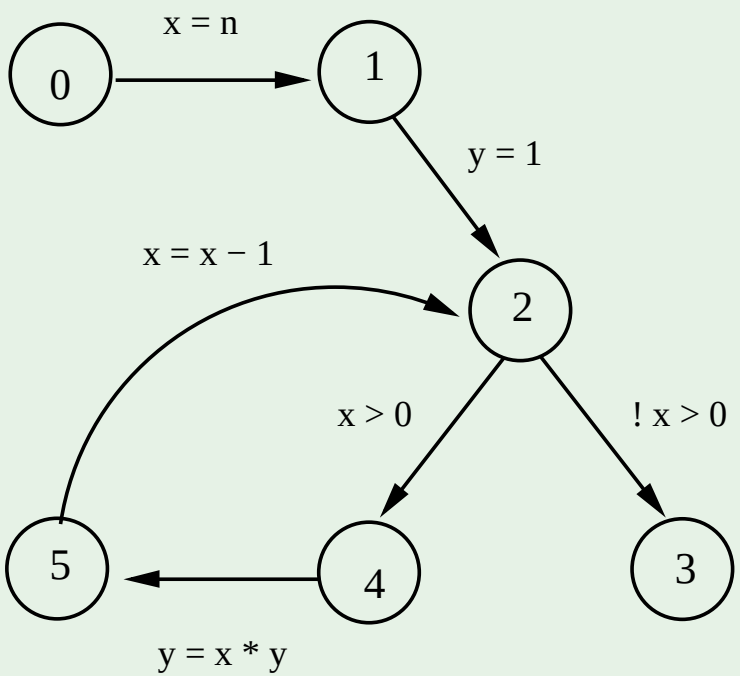
An abstract control flow graph is a CFG whose actions are of the form :

$$\vec{e} : MODIFY\ \vec{v} : USE\ \vec{w} : (BOOL|ASSERT)$$

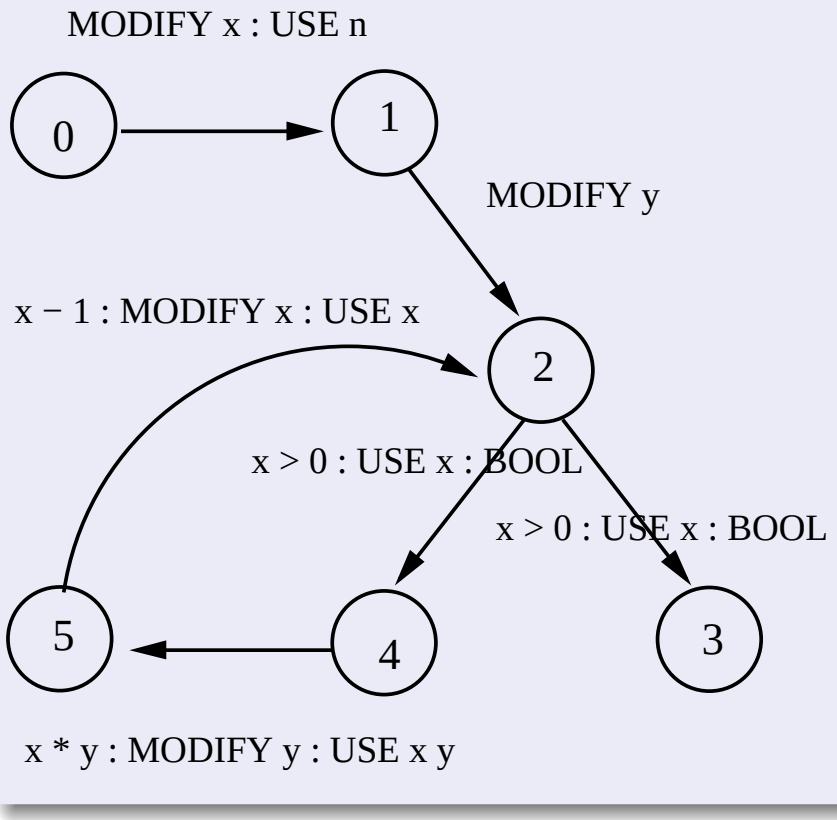
- $\vec{e}$ is the list of non-trivial expressions
- $\vec{v}$ is the list of modified variables in the instruction
- $\vec{w}$ is the list of used (i.e., read) variables
- two types of values, *var* and *expr*, denoting the set of program variables and expressions

Example : abstract model of a C program

Control flow graph example



Abstract control flow graph



Outline

1 Control flow model

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

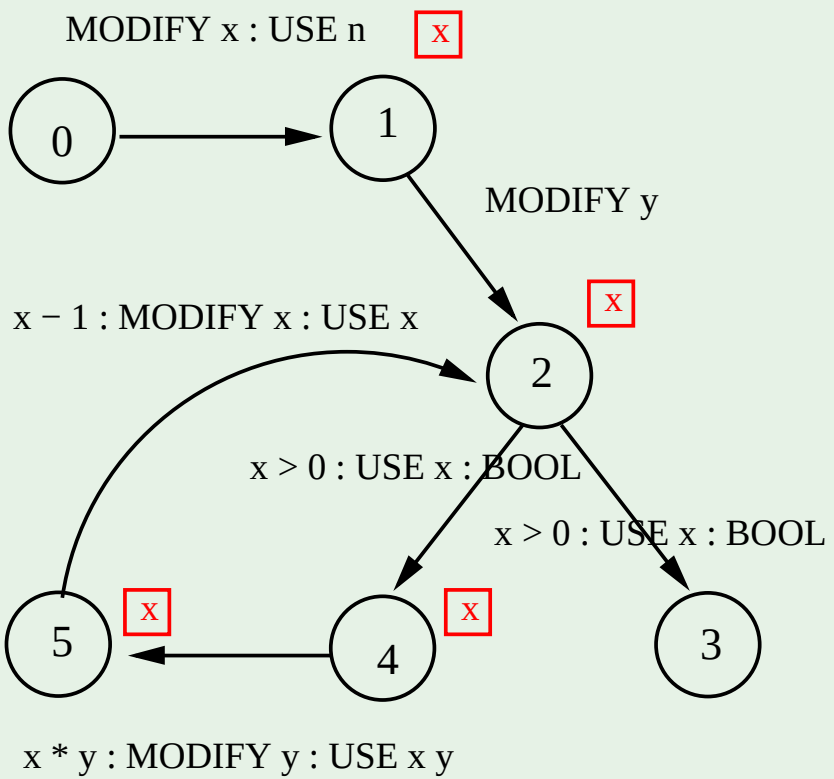
3 BES encoding of classical data flow analyses

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_μ)

Syntax

$$\phi \quad ::= \quad \text{false} \mid \text{true} \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \langle a \rangle \phi \mid [a] \phi \mid X(\vec{e}) \\ \mid \mu X(\vec{x} : \vec{t} := \vec{e}).\phi \mid \nu X(\vec{x} : \vec{t} := \vec{e}).\phi$$

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

Example

L^1_μ formula of *Live variables analysis* :

- "For each program point, which variables *may* be live at the exit point"
- $\phi(v) = \mu Z.\ (\ \langle a \mid \mathit{used}(v,a) \rangle\ \text{true}\ ) \vee (\langle a \mid \neg \mathit{modified}(v,a) \rangle\ Z)$ [Steffen-91][Schmidt-98]

Encoding IA as value-based L^1_μ 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.

$$\phi(p) = \mu Y(v : var := p).(\langle a \mid used(v,a) \wedge bool(a) \rangle \text{ true} \vee \langle a \mid modified(z,a) \wedge used(v,a) \rangle \; Y(z) \vee \langle a \mid \neg modified(v,a) \rangle \; Y(v))$$

where $s, s' \in S$, $a \in A$, *and* $v, z \in var$

- $used(v,a)$ is true if variable v is used (i.e., read) in instruction a
- $bool(a)$ is true if instruction a is a boolean expression
- $modified(v,a)$ is true if variable v is modified on instruction a

Encoding IA as value-based L^1_μ formula (ctnd.)

Assertion influence analysis

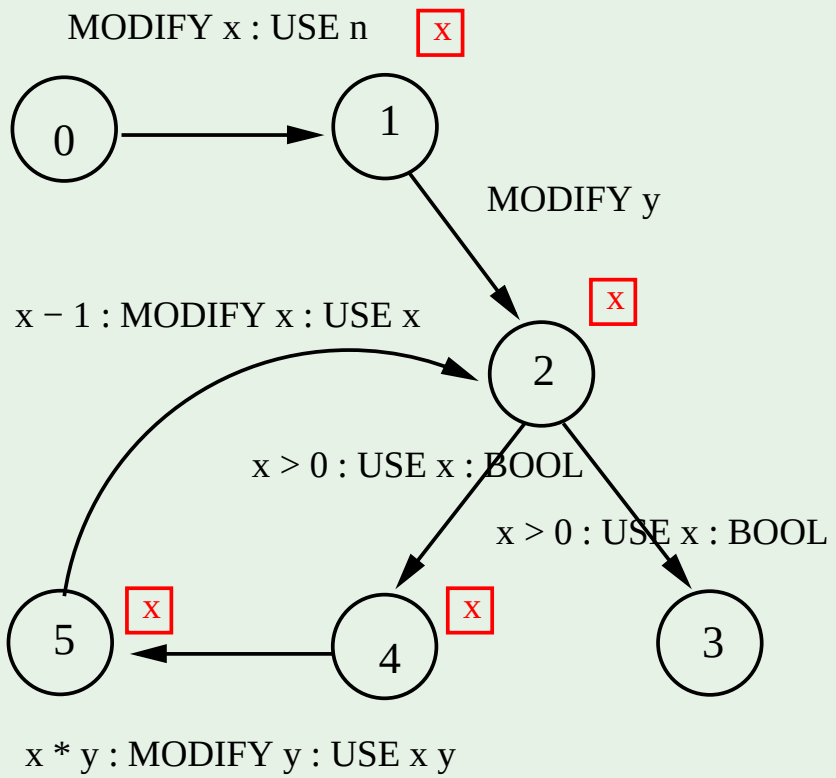
- Reachability IA-MC slightly extended :
  $\ldots \langle a \mid used(v, a) \wedge (bool(a) \vee assert(a)) \rangle$ true $\cdots$
- Interest : safety properties

Formula influence analysis

- Reachability IA-MC slightly extended :
  $\cdots \langle a \mid v \in formula \vee (used(v, a) \wedge (bool(a) \vee modify\_formula(a))) \rangle$ true $\cdots$
- Interest : temporal formulas

Influence analysis as BES resolution (IA-BES)

Abstract control flow graph



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 $M_i = \{x_{ij} \stackrel{\sigma_i}{=} op_{ij} \boldsymbol{X}_{ij}\}_{j \in [1, m_i]}$ is a set of fixed point equations with sign $\sigma_i = \mu$ *or* ν, where :

- x_{ij} is a pure disjunctive or conjunctive formula obtained by applying a boolean operator $op_{ij} \in \{\vee, \wedge\}$ to a set of variables $\boldsymbol{X}_{ij} \subseteq \mathcal{X}$

Definition

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

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

where $s, s' \in S,\ a \in A,$ *and* $v, z \in var$

- Interest : direct translation into PBES from value-based L^1_μ formula

Encoding IA as PBES (ctnd.)

Assertion influence analysis

- Reachability IA-PBES slightly extended :
  $\ldots \{\text{true} \mid s \xrightarrow{a} s' \wedge \mathit{used}(v, a) \wedge (\mathit{bool}(a) \vee \mathit{assert}(a))\} \cdots$
- Interest : safety properties

Formula influence analysis

- Reachability IA-PBES slightly extended :
  $\ldots \{\text{true} \mid v \in \mathit{formula} \vee (s \xrightarrow{a} s' \wedge \mathit{used}(v, a) \wedge (\mathit{bool}(a) \vee \mathit{modify\_formula}(a)))\} \cdots$
- Interest : temporal formulas

Comparison

	Classic IA	IA-MC	IA-PBES
Program representation	control flow graph	Kripke/transition sytem	labeled transition system
Problem statement	flow equation system over sets	value-based L^1_μ formula	PBES
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 contruction 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

$$\left\{ X_{s,v} \overset{\mu}{=} \bigvee(\{\text{true} \mid s \xrightarrow{a} s' \wedge \mathit{used}(v,a)\} \cup \{X_{s',v} \mid s \xrightarrow{a} s' \wedge \neg \mathit{modified}(v,a)\}) \right\} \; s, s' \in S,\ a \in A,\ v \in \mathit{var}$$

- Interest : dead code elimination and register allocation

Very busy expressions analysis

$$\left\{ \begin{array}{lcl} X_{s,e} & \overset{\nu}{=} & \bigwedge(\{Y_{s,e}\} \cup \{\text{true} \mid s \xrightarrow{a} s' \wedge \mathit{used}(e,a)\} \cup \\ & & \quad \{\text{false} \mid s \xrightarrow{a} s' \wedge \neg \mathit{used}(e,a) \wedge \mathit{modified}(e,a)\} \cup \\ & & \quad \{X_{s',e} \mid s \xrightarrow{a} s' \wedge \neg \mathit{modified}(e,a) \wedge \neg \mathit{used}(e,a)\}) \\ Y_{s,e} & \overset{\nu}{=} & \bigvee(\{\text{true} \mid s \xrightarrow{a} s'\}) \end{array} \right\}$$

where $s, s' \in S,\ a \in A$ *and* $e \in \mathit{expr}$

- Interest : code hoisting

Available expressions analysis

$$
\begin{cases}
X_{s,e} \overset{\nu}{=} \bigwedge(\{Y_{s',e} \mid s \xrightarrow{a} s' \wedge ((\neg used(e,a) \wedge Z_{s,e} \notin computed(Z)) \\
\qquad \vee modified(e,a)) \wedge Y_{s,e} \notin computed(Y)\} \cup \\
\qquad \{Z_{s',e} \mid s \xrightarrow{a} s' \wedge used(e,a) \wedge \neg modified(e,a) \\
\qquad \wedge Z_{s,e} \notin computed(Z)\} \cup \{X_{s',e} \vee \text{true} \mid s \xrightarrow{a} s'\}) \\
Y_{s,e} \overset{\nu}{=} \bigwedge(\{Y_{s',e} \mid s \xrightarrow{a} s' \wedge (\neg used(e,a) \vee modified(e,a))\}) \\
Z_{s,e} \overset{\nu}{=} \bigwedge(\{Z_{s',e} \mid s \xrightarrow{a} s' \wedge \neg modified(e,a)\})
\end{cases}
$$

where $s, s' \in S,\ a \in A$ *and* $e \in expr$

- Interest : common sub-expressions elimination

Reachable definitions analysis

$$
\begin{cases}
X_{s,(o,v,t)} \overset{\nu}{=} \bigwedge(\{Y_{t,(o,v,t)} \mid s = o\} \cup \{X_{s',(o,v,t)} \vee \text{true} \mid s \rightarrow s'\}) \\
Y_{s,(o,v,t)} \overset{\nu}{=} \bigwedge(\{Y_{s',(o,v,t)} \mid s \xrightarrow{a} s' \wedge (\neg modified(v,a))\})
\end{cases}
$$

where $s, s', o, t \in S,\ a \in A$ *and* $v \in var$

- Interest : constant and copy propagation

Outline

1 Control flow model

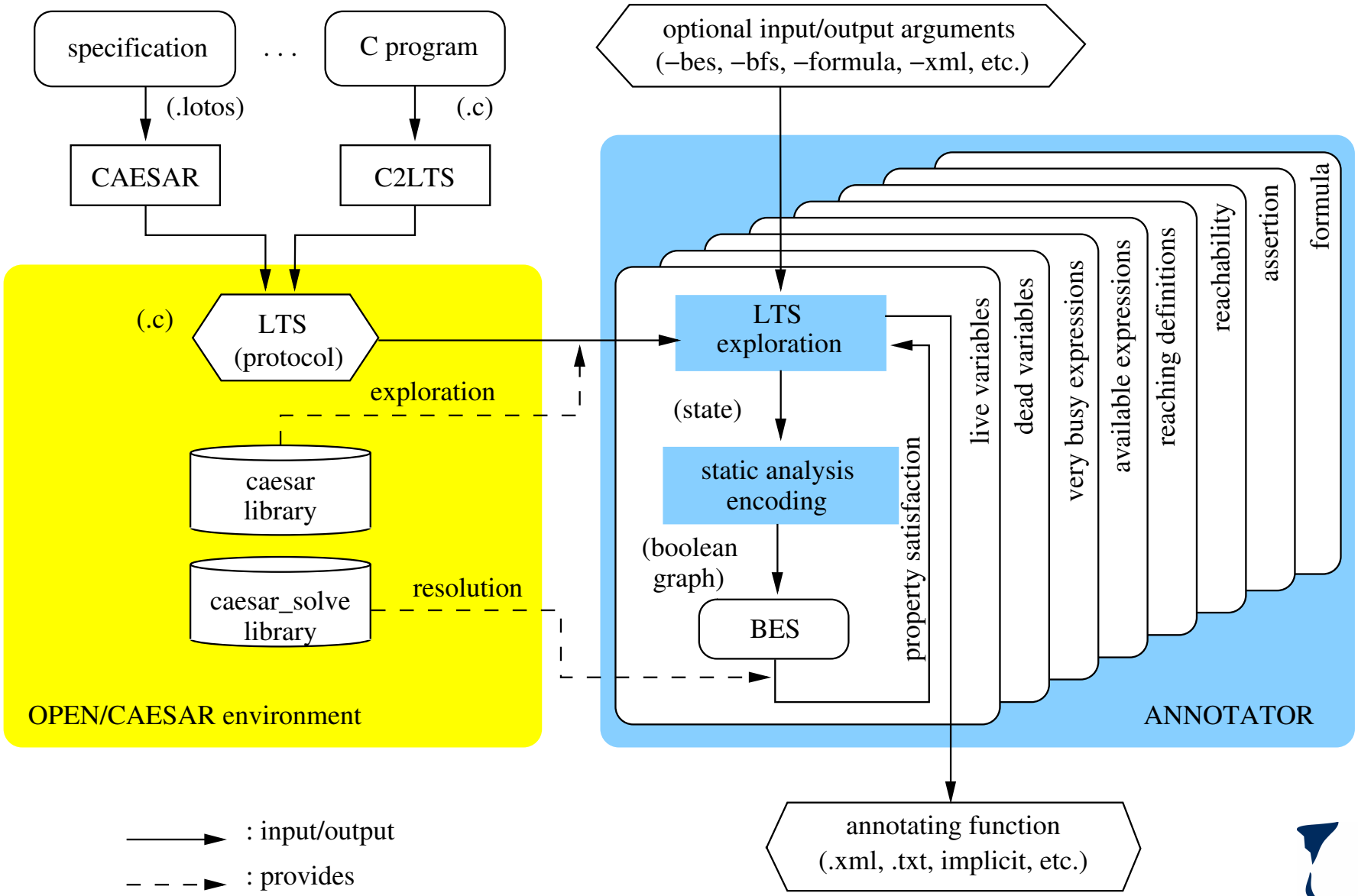
2 Influence analysis as model checking and BES resolution

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

ANNOTATOR : a modular on-the-fly data flow analyser



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, 1 908 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

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_μ/BES
- On-the-fly static analyser ANNOTATOR using OPEN/CÆSAR
  `http://www.lcc.uma.es/gisum/tools/annotator`
- 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 (SOCKET-MC and αSPIN)
- Encoding of other static analyses (Reset Variables)