A Generic On-the-Fly Solver for Alternation-Free Boolean Equation Systems

Radu Mateescu

INRIA Rhône-Alpes / VASY

655, avenue de l'Europe

F-38330 Montbonnot Saint Martin, France

http://www.inrialpes.fr/vasy

Outline

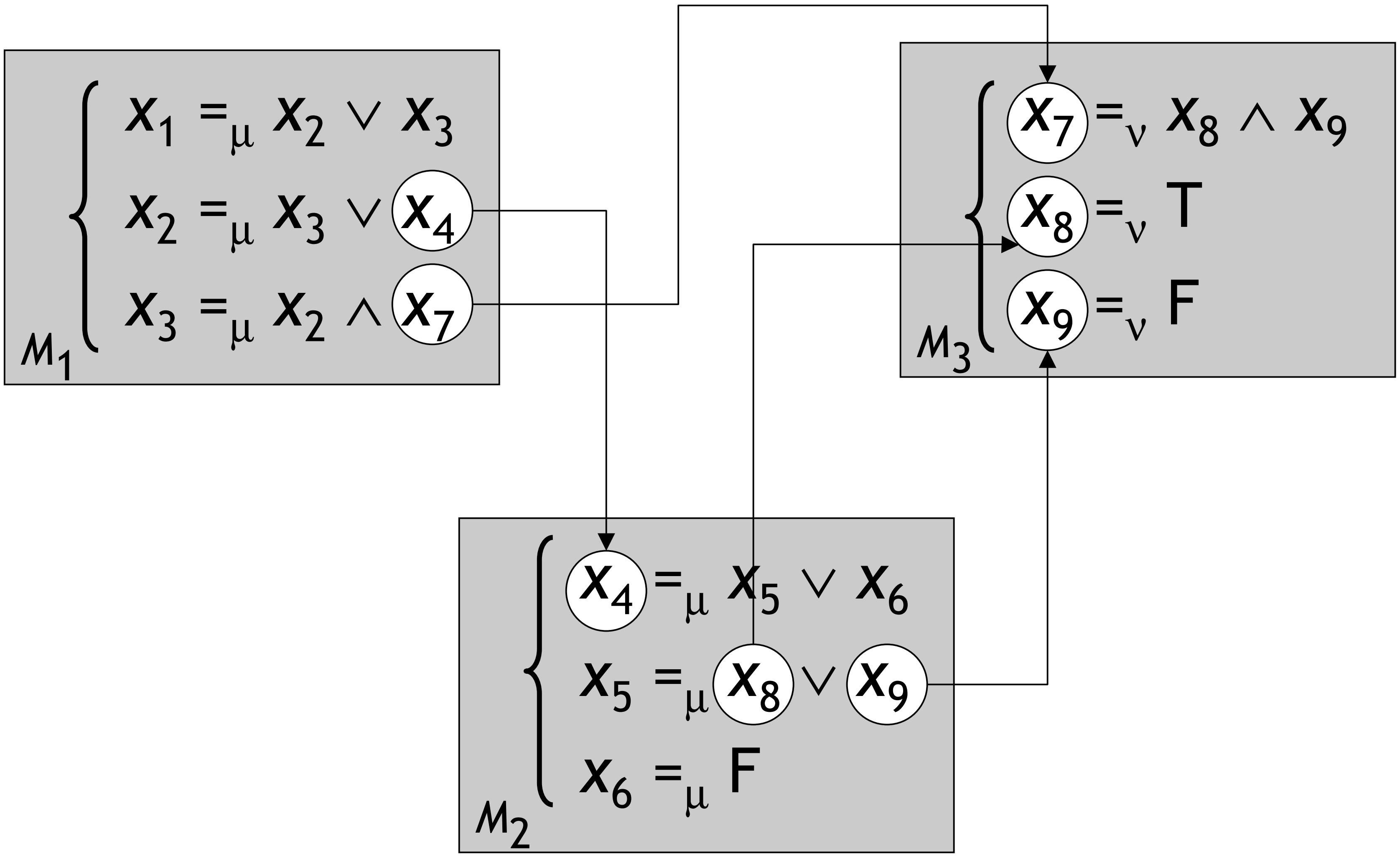
- Introduction
- Boolean Equation Systems
- On-the-fly resolution algorithms
- Equivalence checking and model checking
- Implementation and experiments
- Future work

Introduction

- On-the-fly verification
  - Builds the state space incrementally
  - Allows to detect errors in large systems
- Practical needs
  - Easy construction of on-the-fly verification tools
  - Generic software components for verification
- Boolean Equation Systems (BES)
  - Technology for equivalence checking and model checking
  - On-the-fly resolution and diagnostic generation

➔ *Goal: provide generic software (libraries)*

Alternation-free BES



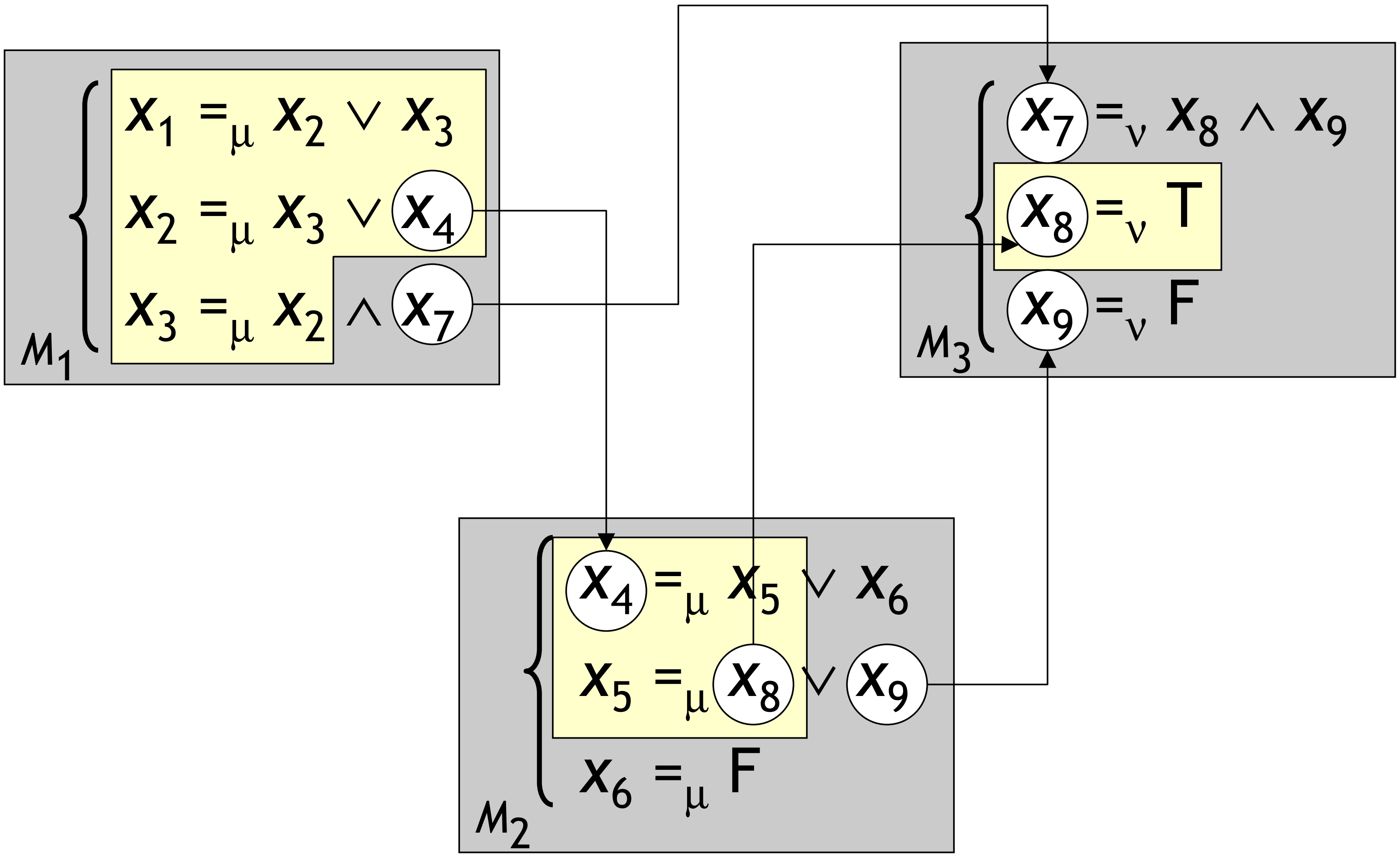
On-the-fly resolution

- Alternation-free BES $B = (x, M_1, \ldots, M_n)$
  - Compute x without solving the whole BES
- Approach:
  - Associate a resolution routine R_i to block M_i
  - $R_i (x_j)$ computes the value of x_j in M_i
  - Evaluation of right-hand side formulas and substitution
  - Bounded call stack $R_1 (x) \rightarrow \ldots \rightarrow R_n (x_k)$

➔ *Simple algorithms (a single kind of fixed points)*

➔ *Easy to optimize (particular kinds of blocks)*

Example


$x_1 =_\mu x_2 \vee x_3$
$x_2 =_\mu x_3 \vee x_4$
$x_3 =_\mu x_2 \wedge x_7$
M_1
$x_7 =_\nu x_8 \wedge x_9$
$x_8 =_\nu T$
$x_9 =_\nu F$
M_3
$x_4 =_\mu x_5 \vee x_6$
$x_5 =_\mu x_8 \vee x_9$
$x_6 =_\mu F$
M_2

Resolution algorithms: Principles

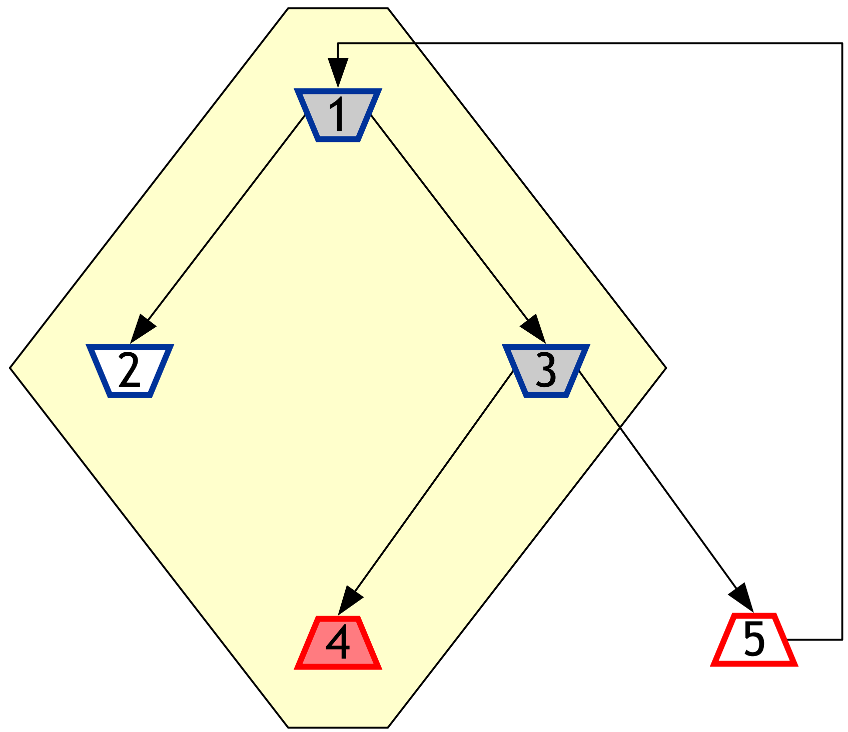
- Represent blocks as *boolean graphs* [Andersen-94]
- Block M represented by boolean graph $G = (V, E, L)$:
  - V: set of vertices (variables)
  - E: set of edges (dependencies between variables)
  - $L : V \rightarrow \{ \vee, \wedge \}$: vertex labeling (disjunctive/conjunctive)
- Principle of resolution algorithms:
  - *Forward* exploration of G starting at $x \in V$
  - *Backward* propagation of stable (computed) variables
  - *Termination* when x is stable or G is entirely explored
  - *Diagnostic* by keeping relevant successors [Mateescu-00]

Example

BES (μ-block)

$$\begin{cases} x_1 =_\mu x_2 \vee x_3 \\ x_2 =_\mu F \\ x_3 =_\mu x_4 \vee x_5 \\ x_4 =_\mu T \\ x_5 =_\mu x_1 \end{cases}$$

boolean graph



: ∨-variables

: ∧-variables

Three effectiveness criteria

For each resolution routine *R*:

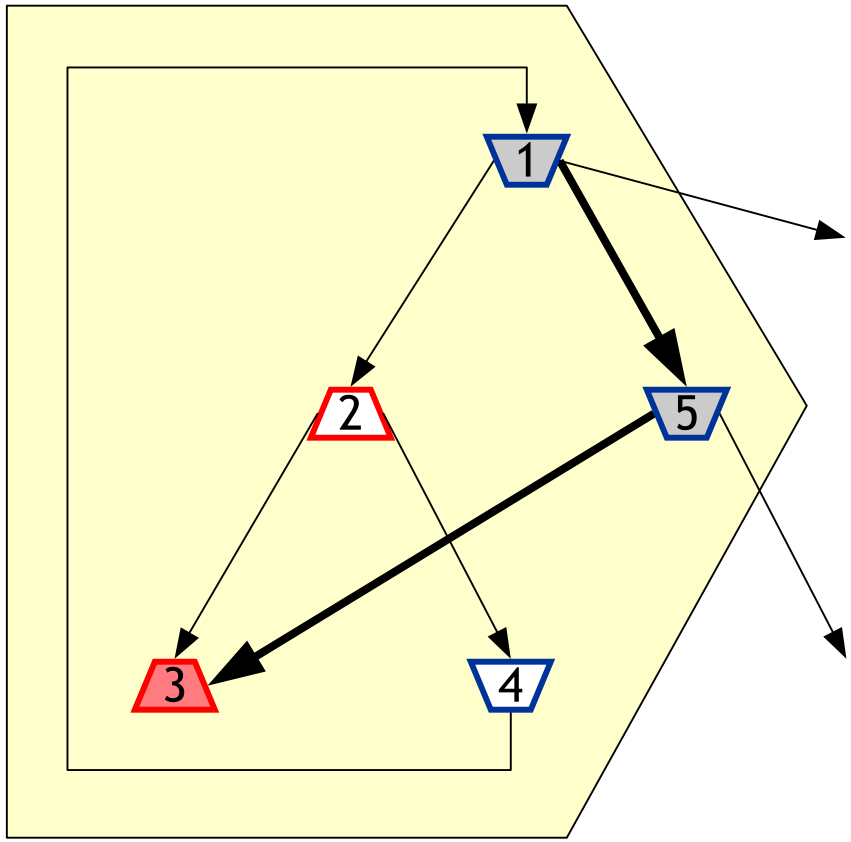
A. The worst-case complexity of a call *R* (*x*) must be *O* (|*V*|+|*E*|)
   ➔ *linear-time complexity for the overall BES resolution*

B. While executing *R* (*x*), every variable explored must be « linked » to *x* via unstable variables
   ➔ *graph exploration limited to « useful » variables*

C. After termination of *R* (*x*), all variables explored must be stable
   ➔ *keep resolution results between subsequent calls of R*

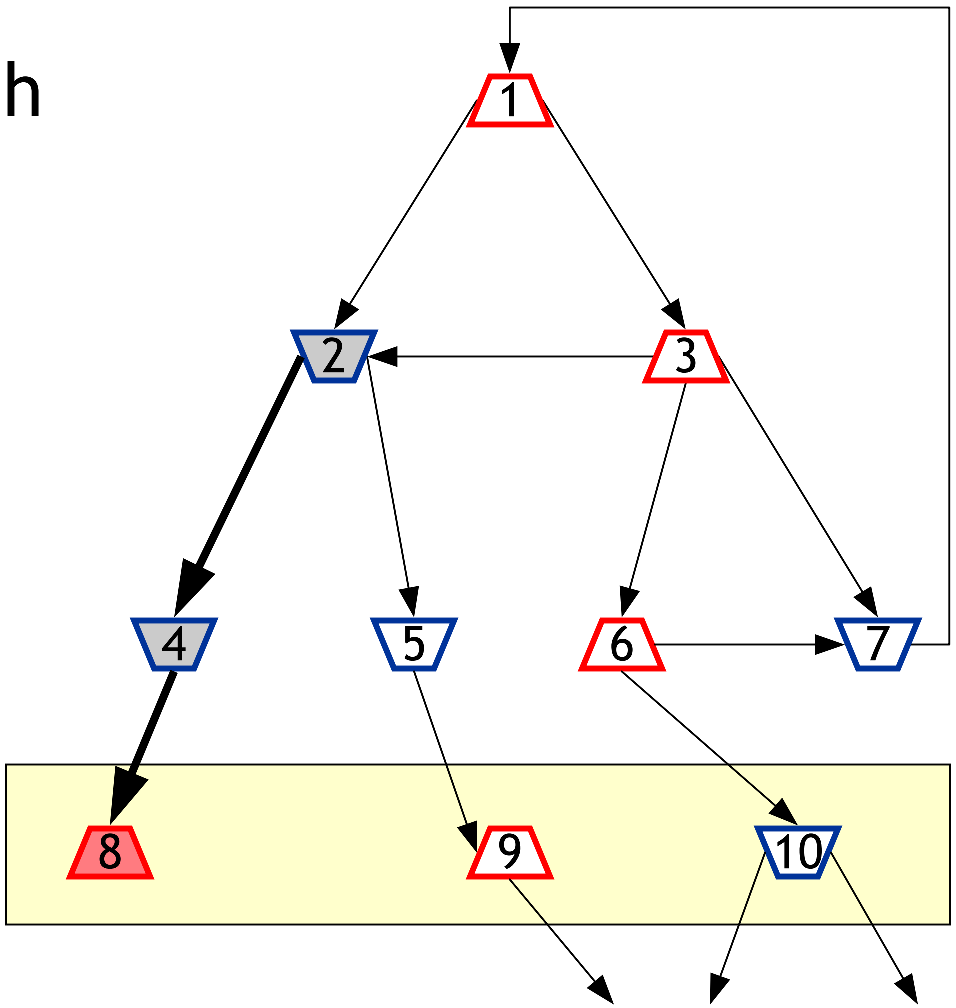
Algorithm A1
(general)

- DFS of the boolean graph
- Satisfies A, B, C
- Memory complexity
  $O(|V|+|E|)$
- Optimized version of [Andersen-94]
- Developed for model checking regular alternation-free μ-calculus [Mateescu-Sighireanu-00]



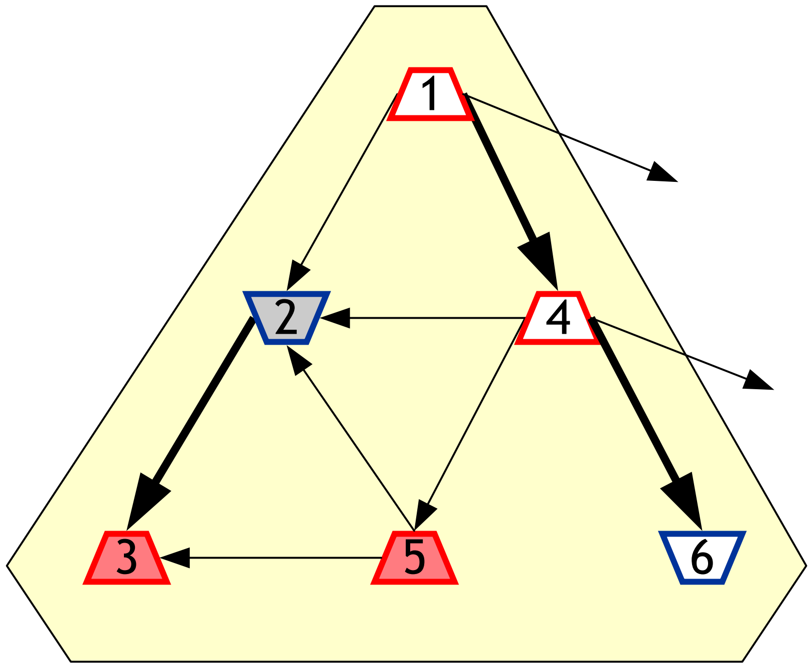
Algorithm A2
(general)

- BFS of the boolean graph
- Satisfies A, C
  (risk of computing useless variables)
- Slightly slower than A1
- Memory complexity
  $O(|V|+|E|)$
- Low-depth diagnostics



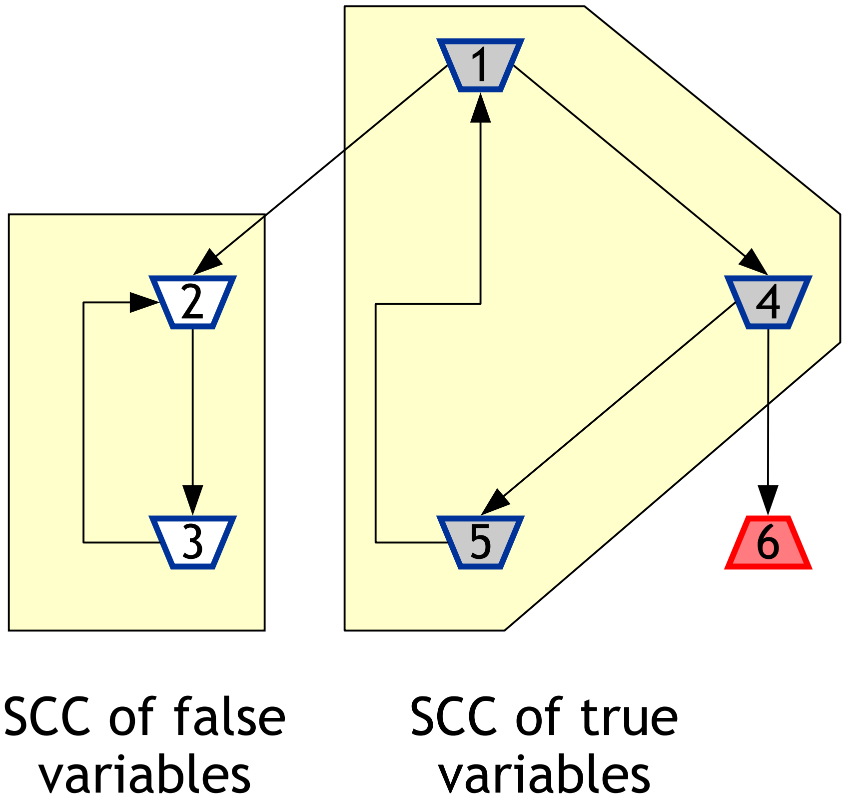
Algorithm A3
(acyclic)

- DFS of the boolean graph
- Back-propagation of stable variables on the DFS stack only
- Satisfies A, B, C
- Avoids storing edges
- Memory complexity *O* (|*V*|)
- Developed for trace-based verification [Mateescu-02]



Algorithm A4
(disjunctive / conjunctive)

- DFS of the boolean graph
- Detection and stabilization of SCCs
- Satisfies A, B, C
- Avoids storing edges
- Memory complexity $O(|V|)$
- Developed for model checking ACTL and PDL

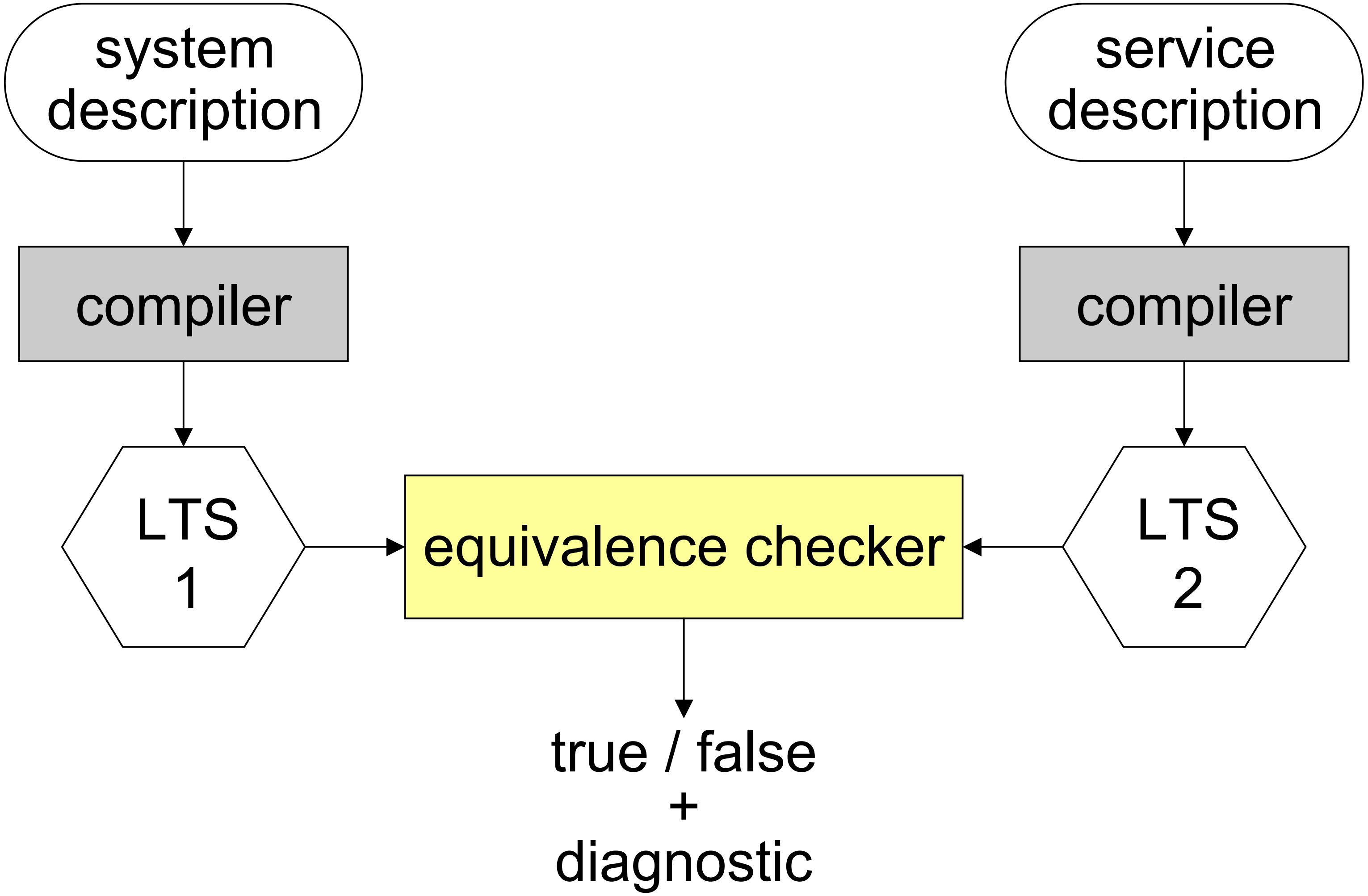


Resolution algorithms: Summary

- A1 (DFS, general)
  - Satisfies A, B, C
  - Memory complexity $O(|V|+|E|)$
- A2 (BFS, general)
  - Satisfies A, C + « small » diagnostics
  - Memory complexity $O(|V|+|E|)$
- A3 (DFS, acyclic)
  - Satisfies A, B, C
  - Memory complexity $O(|V|)$
- A4 (DFS, disjunctive/conjunctive)
  - Satisfies A, B, C
  - Memory complexity $O(|V|)$

} Time complexity $O(|V|+|E|)$

Equivalence checking



From equivalences to BESs

- Strong equivalence: $s_1 \approx s_2$ iff $X_{s1,s2}$ is true

$$\begin{cases} X_{s1,s2} =_\nu (\wedge_{s1 \to a\, s1'} Y_{a,s1',s2}) \wedge (\wedge_{s2 \to a\, s2'} Z_{a,s1,s2'}) \\ Y_{a,s1',s2} =_\nu \vee_{s2 \to a\, s2'} X_{s1',s2'} \\ Z_{a,s1,s2'} =_\nu \vee_{s1 \to a\, s1'} X_{s1',s2'} \end{cases}$$

$s_1 \leq s_2$ (preorder)

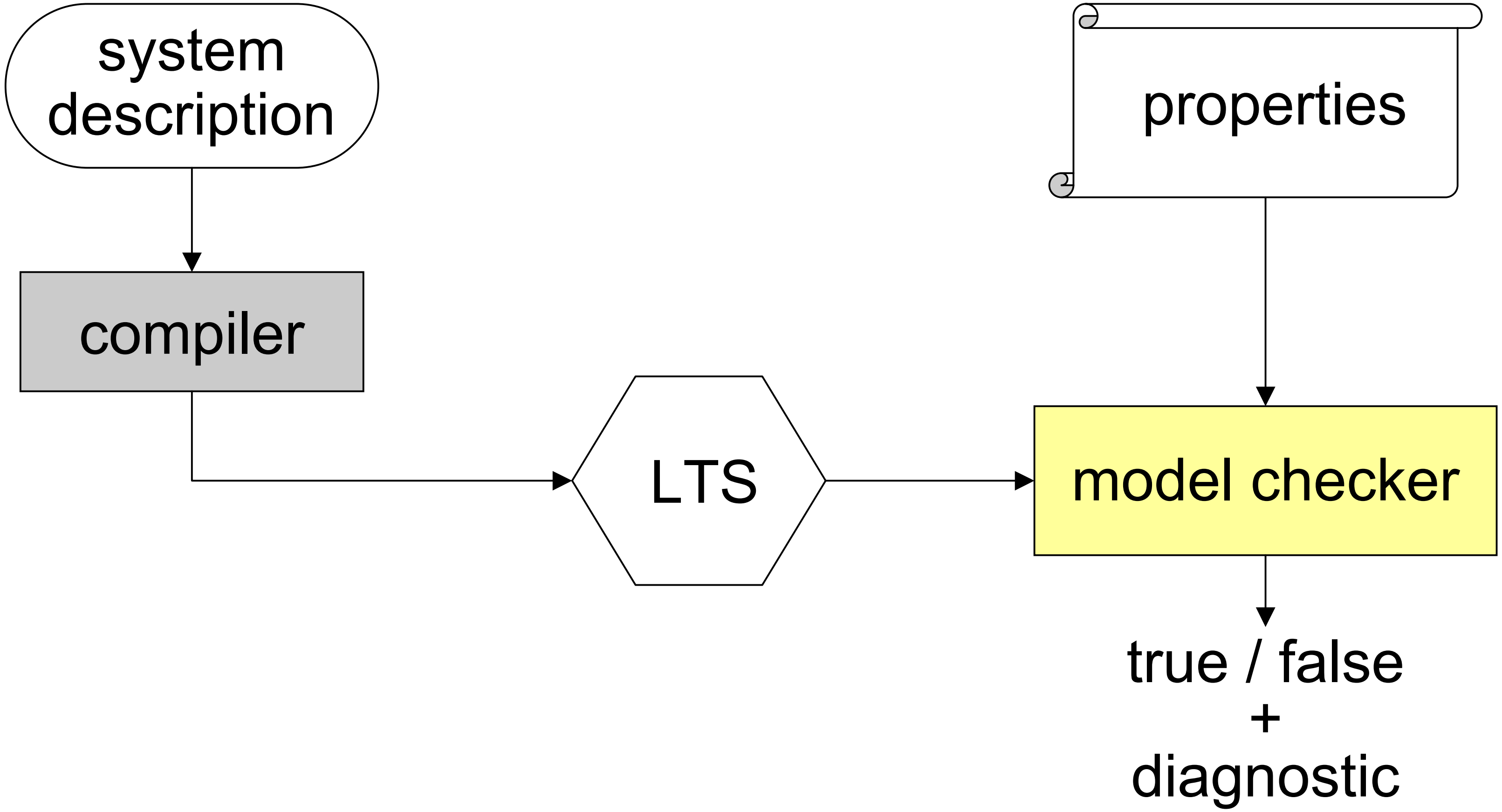
- Weak equivalences:
  - Similar scheme, with transitive closure over τ-transitions
  - Branching, observational, τ*.a, safety, delay, ...

➔ *Translation allows to build the LTS on-the-fly*

Equivalence checking: Summary

- *General* boolean graph:
  - All equivalences and their preorders
  - Algorithms A1 and A2 (counterexample depth ↓)
- *Acyclic* boolean graph:
  - Strong equivalence: one of the LTS acyclic
  - $\tau^*.a$ and safety: one LTS acyclic (τ-circuits allowed)
  - Branching and observational: both LTS acyclic
  - Algorithm A3 (memory ↓)
- *Conjunctive* boolean graph:
  - All equivalences: one of the LTS deterministic
  - Algorithm A4 (memory ↓)

Model checking



From temporal logics to BESs

- Alternation-free μ-calculus: $s \models \varphi$ iff φ_s is true
- Potential reachability of an action *a*:

$$\mu X \,.\, \varphi \vee \langle a \rangle X$$

$$\left\{ X_s =_\mu \varphi_s \vee \bigvee_{s \rightarrow a\, s'} X_{s'} \right.$$

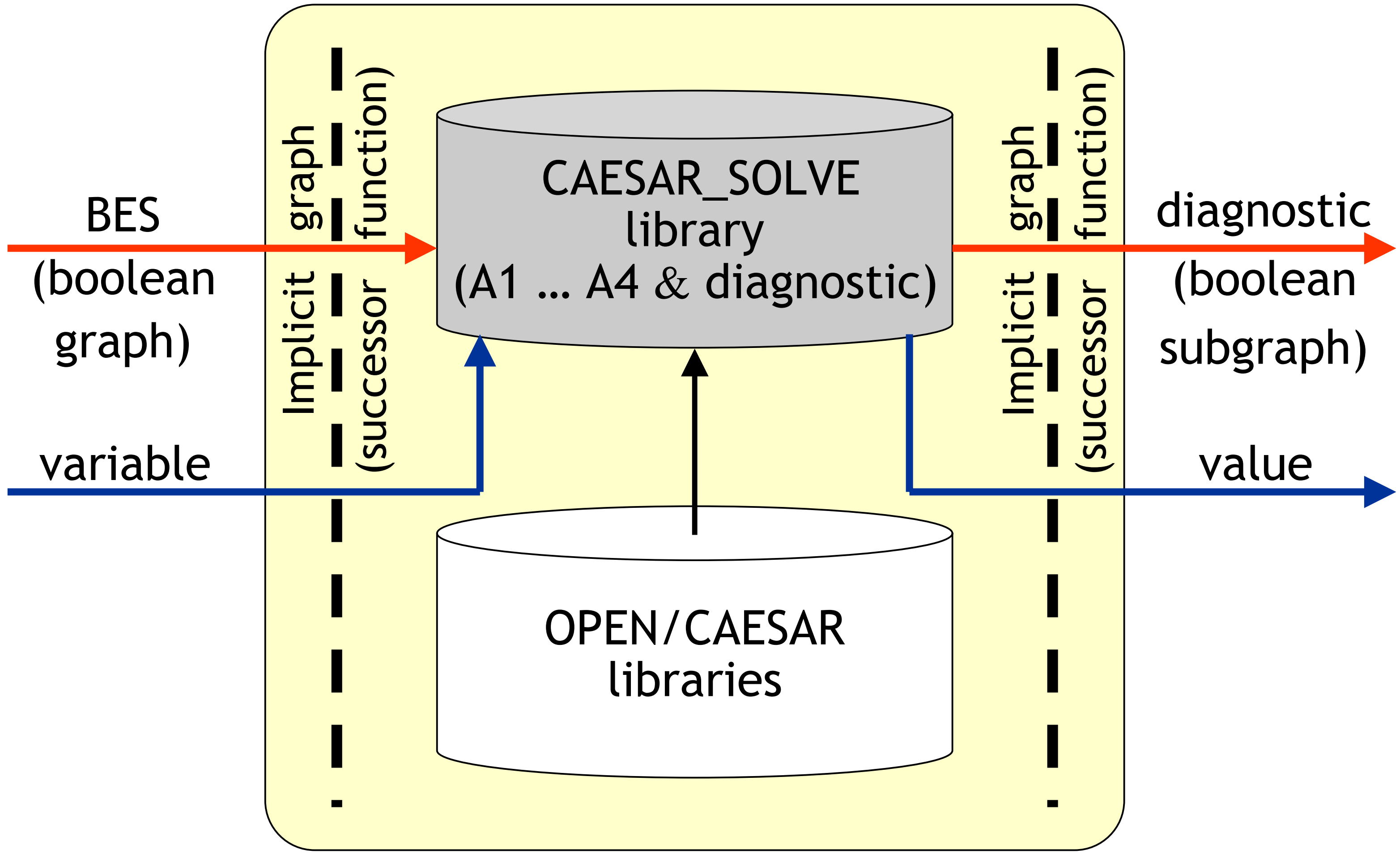
- Other temporal logics:
  - Similar scheme (via translation to μ-calculus)
  - CTL, ACTL (Action CTL), PDL

➔ *Translation allows to build the LTS on-the-fly*

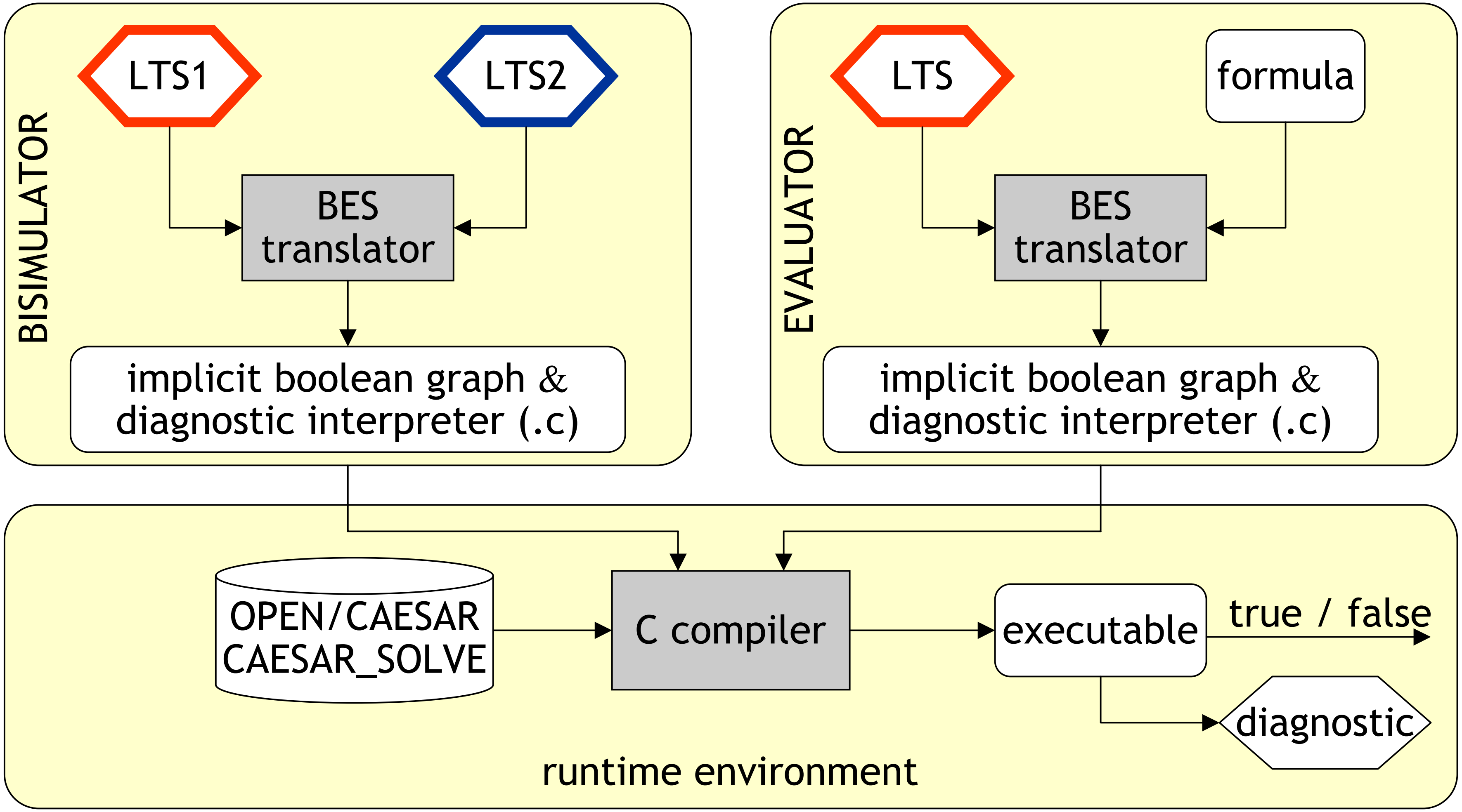
Model checking: Summary

- *General* boolean graph:
  - Any LTS and any alternation-free μ-calculus formula
  - Algorithms A1 and A2 (diagnostic depth $\downarrow$)
- *Acyclic* boolean graph:
  - Acyclic LTS and guarded formula (CTL, ACTL)
  - Acyclic LTS and μ-calculus formula (via reduction)
  - Algorithm A3 (memory $\downarrow$)
- *Disjunctive/conjunctive* boolean graph:
  - Any LTS and any formula of CTL, ACTL, PDL
  - Algorithm A4 (memory $\downarrow$)

CAESAR_SOLVE library



BISIMULATOR and EVALUATOR



Performance measures

- A2 versus A1:
  - Compare LTS - erroneous LTS (strong equivalence)
  - Check invalid properties on the LTS

  ➔ *Reductions 75 % - 99 % in diagnostic depth*
- A3 versus A1:
  - Inclusion of sequences (100,000 transitions) in the LTS
  - Check valid properties on sequences

  ➔ *Reductions 15 % - 27 % in memory*
- A4 versus A1:
  - Compare LTS – service LTS ($\tau^*.a$ equivalence)
  - Check valid properties (ACTL + PDL) on the LTS

  ➔ *Reductions 12 % - 63 % in memory*

Future work

- New algorithms within CAESAR_SOLVE
  - Single-scan & low-memory algorithms for trace-based verification (low-depth acyclic boolean graphs)
  - Further resolution strategies (combined DFS-BFS, random exploration, …)
- New applications of CAESAR_SOLVE
  - Detection of τ-confluent transitions [CAV 2003]
  - Test generation
  - Discrete controller synthesis

  } using diagnostic generation
- Distributed resolution algorithms

  ➔ Distributed equivalence checking and model checking