Model Checking of Action-Based Concurrent Systems

Radu Mateescu
INRIA Rhône-Alpes / VASY
http://www.inrialpes.fr/vasy

Action-based temporal logics

- Introduction
- Modal logics
- Branching-time logics
- Regular logics
- Fixed point logics

Why temporal logics?

- Formalisms for high-level specification of systems
  - Example: all mutual exclusion protocols should satisfy
    - *Mutual exclusion* (at most one process in critical section)
    - *Liveness* (each process should eventually enter its critical section)
- Temporal logics (TLs):

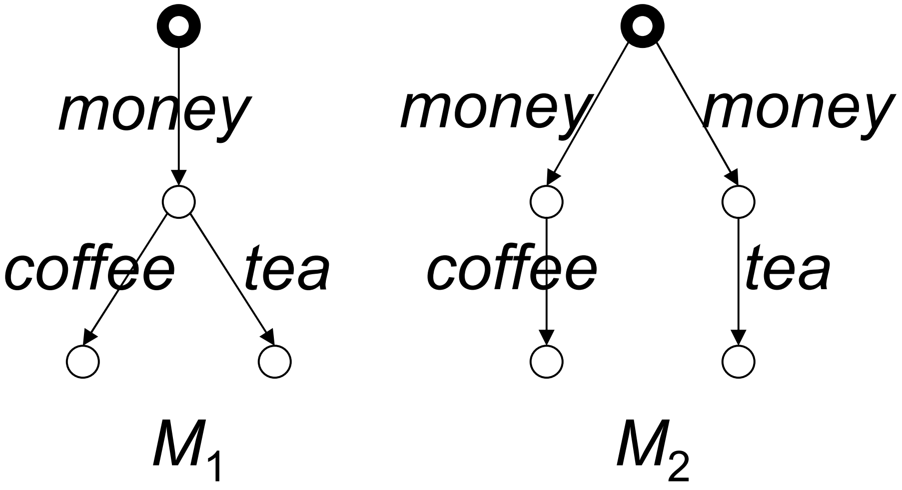
  formalisms describing the ordering of states (or actions) during the execution of a concurrent program
- TL specification = list of logical formulas, each one expressing a property of the program
- Benefits of TL [Pnueli-77]:
  - *Abstraction*: properties expressed in TL are independent from the description/implementation of the system
  - *Modularity*: one can add/remove a property without impacting the other properties of the specification

(Rough) classification of TLs

	State-based	Action-based
Linear-time (properties about execution sequences)	LTL (SPIN tool) linear mu-calculus	TLA (TLA+ tool) action-based LTL (LTSA tool)
Branching-time (properties about execution trees)	CTL (nuSMV tool) CTL*	ACTL (JACK tool) ACTL* modal mu-calculus (CWB, Concurrency Factory, CADP tools)

Example

(coffee machine)



$L(M_1) = L(M_2) =$
{ *money.coffee*, *money.tea* }

- A linear-time TL cannot distinguish the two LTSs M_1 and M_2, which have the same set of execution sequences, but are not behaviourally equivalent (modulo strong bisimulation)
- A branching-time TL can capture nondeterminism and thus can distinguish M_1 and M_2

Interpretation of (branching-time) TLs on LTSs

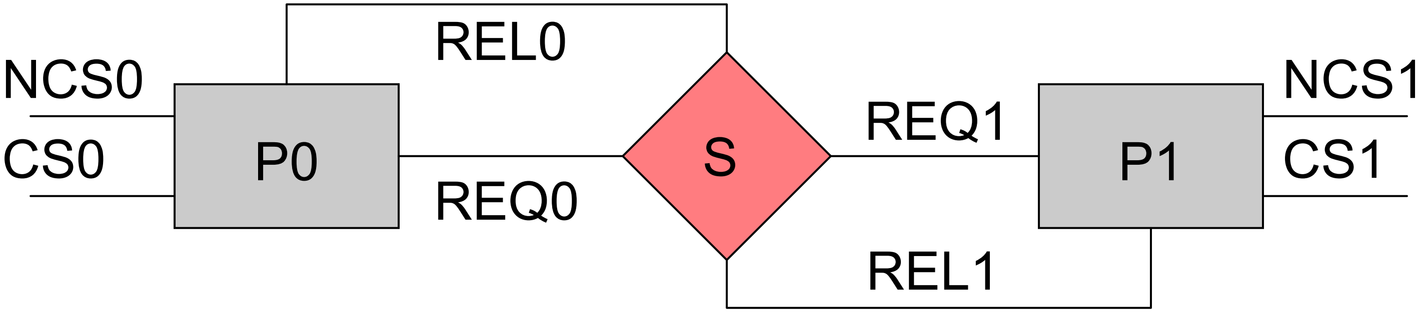
- LTS model $M = \langle\, S, A, T, s_0 \,\rangle$, where:
  - S: set of states
  - A: set of actions (events)
  - $T \in S \times A \times S$: transition relation
  - $s_0 \in S$: initial state
- Interpretation of a formula φ on M:

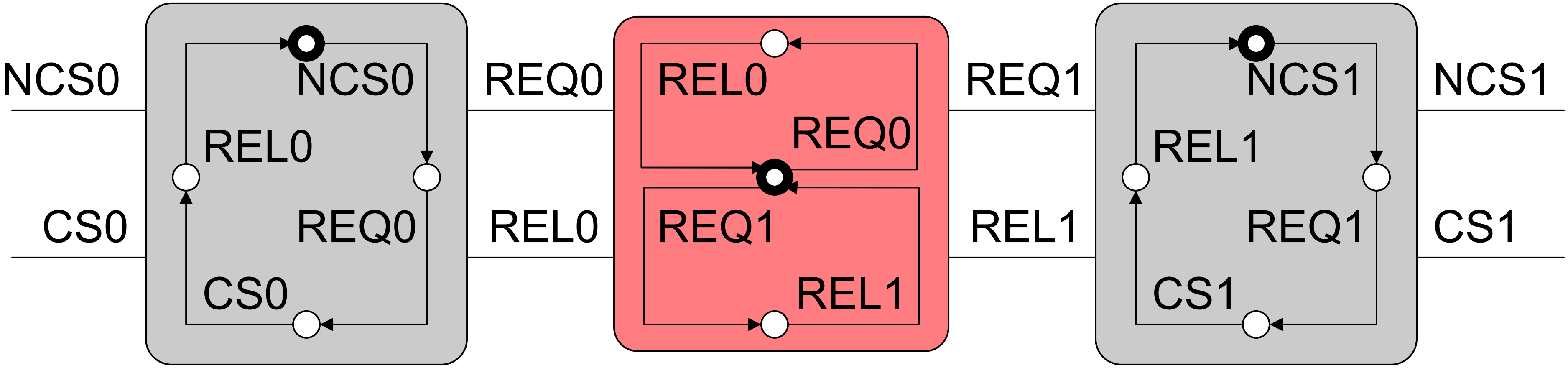
  $$[[\ \varphi\ ]] = \{\ s \in S \mid s \models \varphi\ \}$$

  ($[[\ \varphi\ ]]$ defined inductively on the structure of φ)
- An LTS M satisfies a TL formula φ ($M \models \varphi$) iff its initial state satisfies φ :

  $$M \models \varphi \quad \Leftrightarrow \quad s_0 \models \varphi \quad \Leftrightarrow \quad s_0 \in [[\ \varphi\ ]]$$

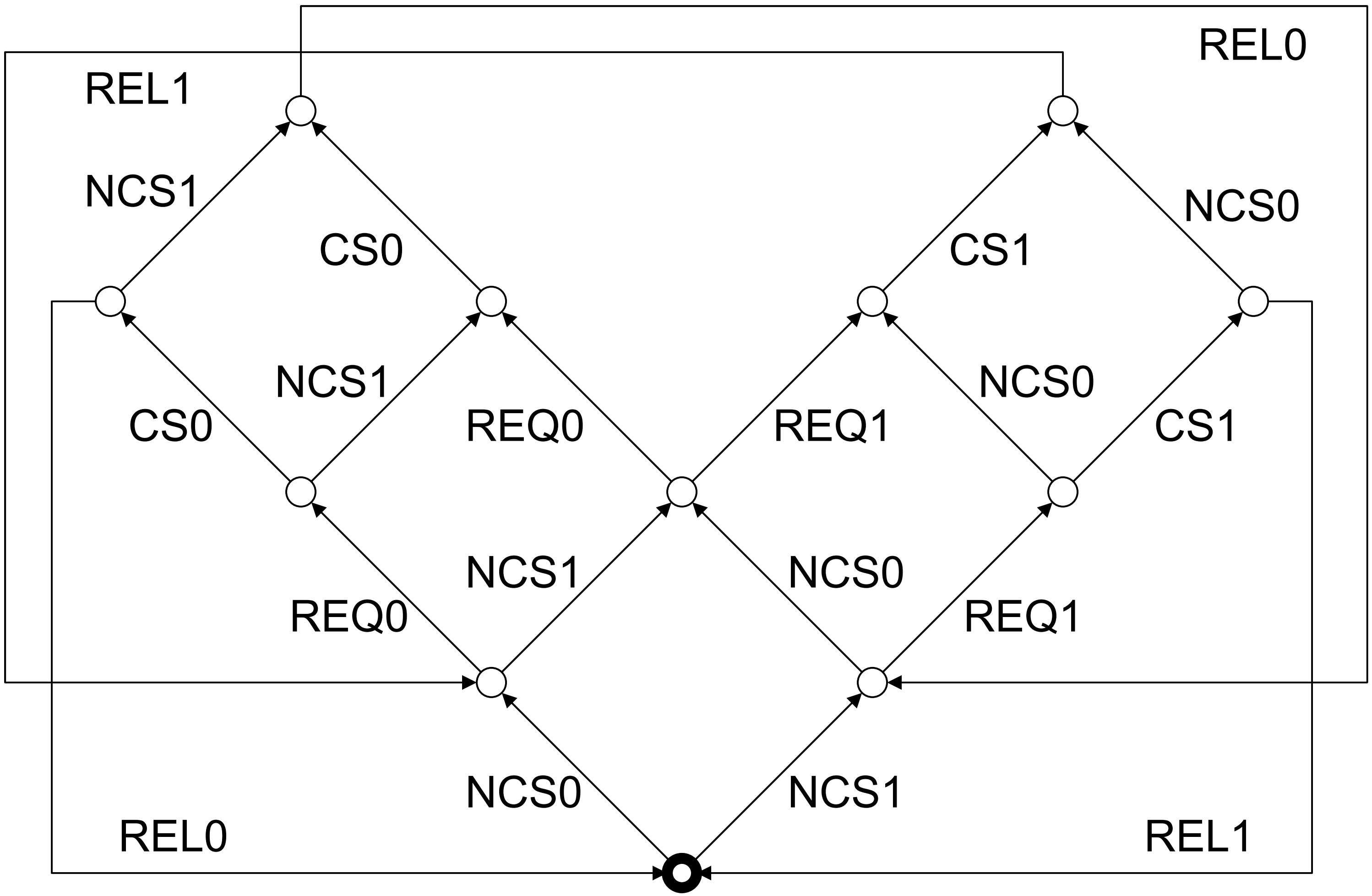
Running example:
mutual exclusion with a semaphore





Description using communicating automata

LTS model


REL0
REL1
NCS1
CS0
NCS0
CS1
NCS1
CS0
REQ0
REQ1
NCS0
CS1
NCS1
NCS0
REQ0
REQ1
NCS0
NCS1
REL0
REL1

Modal logics

- They are the simplest logics allowing to reason about the sequencing and branching of transitions in an LTS
- Basic modal operators:
  - *Possibility*

    from a state, there exists (at least) an outgoing transition labeled by a certain action and leading to a certain state
  - *Necessity*

    from a state, all the outgoing transitions labeled by a certain action lead to certain states
- Hennessy-Milner Logic (HML) [Hennessy-Milner-85]

Action predicates

(syntax)

α ::=	a	atomic proposition ($a \in A$)
|	tt	constant "true"
|	ff	constant "false"
|	$\alpha_1 \vee \alpha_2$	disjunction
|	$\alpha_1 \wedge \alpha_2$	conjunction
|	$\neg\alpha_1$	negation
|	$\alpha_1 \Rightarrow \alpha_2$	implication ($\neg\alpha_1 \vee \alpha_2$)
|	$\alpha_1 \Leftrightarrow \alpha_2$	equivalence ($\alpha_1 \Rightarrow \alpha_2 \wedge \alpha_1 \Rightarrow \alpha_2$)

Action predicates

(semantics)

Let $M = (S, A, T, s_0)$. Interpretation $[[\ \alpha\ ]] \subseteq A$:

- $[[\ a\ ]] = \{\ a\ \}$
- $[[\ \mathrm{tt}\ ]] = A$
- $[[\ \mathrm{ff}\ ]] = \emptyset$
- $[[\ \alpha_1 \vee \alpha_2\ ]] = [[\ \alpha_1\ ]] \cup [[\ \alpha_2\ ]]$
- $[[\ \alpha_1 \wedge \alpha_2\ ]] = [[\ \alpha_1\ ]] \cap [[\ \alpha_2\ ]]$
- $[[\ \neg\alpha_1\ ]] = A \setminus [[\ \alpha_1\ ]]$
- $[[\ \alpha_1 \Rightarrow \alpha_2\ ]] = (A \setminus [[\ \alpha_1\ ]]) \cup [[\ \alpha_2\ ]]$
- $[[\ \alpha_1 \Leftrightarrow \alpha_2\ ]] = ((A \setminus [[\ \alpha_1\ ]]) \cup [[\ \alpha_2\ ]])$
  $\cap\ ((A \setminus [[\ \alpha_2\ ]]) \cup [[\ \alpha_1\ ]])$

Examples

A = { NCS_0, NCS_1, CS_0, CS_1, REQ_0, REQ_1, REL_0, REL_1 }

- $[[\ tt\ ]] = \{ NCS_0, NCS_1, CS_0, CS_1, REQ_0, REQ_1, REL_0, REL_1 \}$
- $[[\ ff\ ]] = \emptyset$
- $[[\ NCS_0\ ]] = \{ NCS_0 \}$
- $[[\ \neg NCS_0\ ]] = \{ NCS_1, CS_0, CS_1, REQ_0, REQ_1, REL_0, REL_1 \}$
- $[[\ NCS_0 \wedge \neg NCS_1\ ]] = \{ NCS_0 \} = [[\ NCS_0\ ]]$
- $[[\ NCS_0 \vee NCS_1\ ]] = \{ NCS_0, NCS_1 \}$
- $[[\ (NCS_0 \vee NCS_1) \wedge (NCS_0 \vee REQ_0)\ ]] = \{ NCS_0 \}$
- $[[\ NCS_0 \wedge NCS_1\ ]] = \emptyset = [[\ ff\ ]]$
- $[[\ NCS_0 \vee \neg NCS_0\ ]] =$
  $\{ NCS_0, NCS_1, CS_0, CS_1, REQ_0, REQ_1, REL_0, REL_1 \} = [[\ tt\ ]]$

HML logic

(syntax)

$\varphi ::=$	tt	constant “true”
$\mid$	ff	constant “false”
$\mid$	$\varphi_1 \vee \varphi_2$	disjunction
$\mid$	$\varphi_1 \wedge \varphi_2$	conjunction
$\mid$	$\neg\varphi_1$	negation
$\mid$	$\langle \alpha \rangle \varphi_1$	possibility
$\mid$	$[\alpha] \varphi_1$	necessity

- Duality: $[\alpha] \varphi = \neg\langle \alpha \rangle \neg\varphi$

HML logic

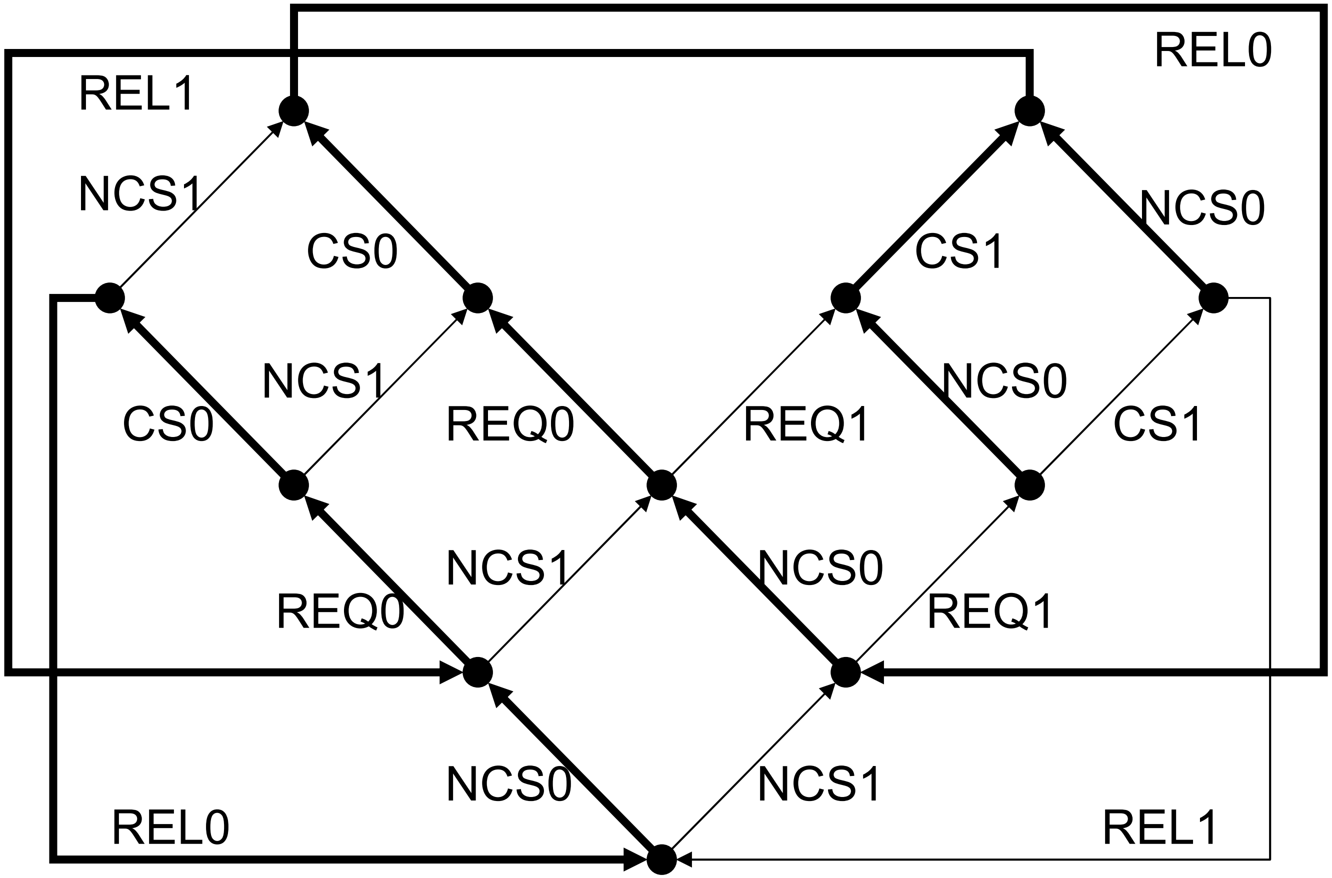
(semantics)

Let $M = (S, A, T, s_0)$. Interpretation $[[\ \varphi\ ]] \subseteq S$:

- $[[\ tt\ ]] = S$
- $[[\ ff\ ]] = \varnothing$
- $[[\ \varphi_1 \vee \varphi_2\ ]] = [[\ \varphi_1\ ]] \cup [[\ \varphi_2\ ]]$
- $[[\ \varphi_1 \wedge \varphi_2\ ]] = [[\ \varphi_1\ ]] \cap [[\ \varphi_2\ ]]$
- $[[\ \neg\varphi_1\ ]] = S \setminus [[\ \varphi_1\ ]]$
- $[[\ \langle\ \alpha\ \rangle\ \varphi_1\ ]] = \{\ s \in S \mid \exists\ (s, a, s') \in T\ .$ $a \in [[\ \alpha\ ]] \wedge s' \in [[\ \varphi_1\ ]]\ \}$
- $[[\ [\ \alpha\ ]\ \varphi_1\ ]] = \{\ s \in S \mid \forall\ (s, a, s') \in T\ .$ $a \in [[\ \alpha\ ]] \Rightarrow s' \in [[\ \varphi_1\ ]]\ \}$

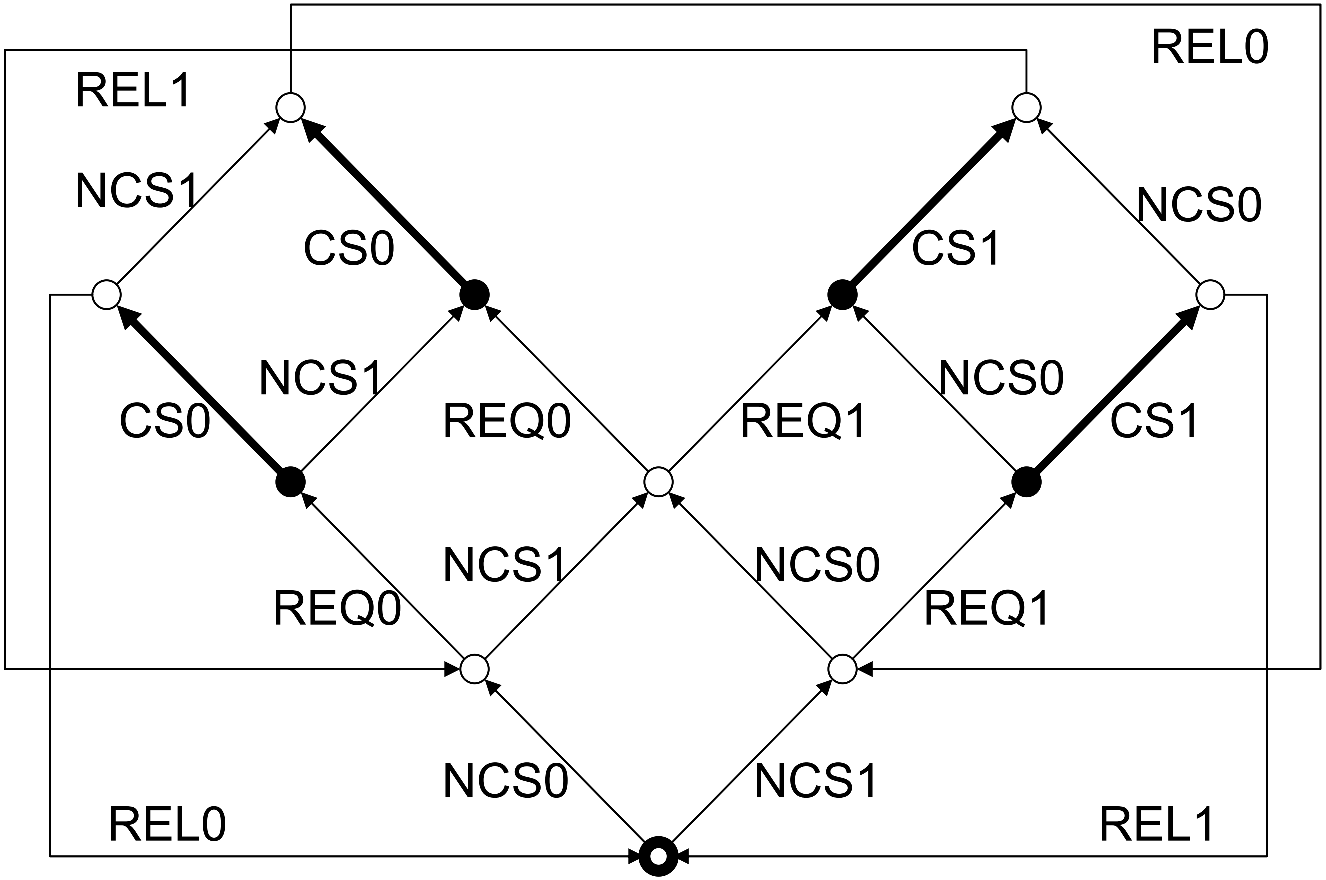
Example (1/4)

Deadlock freedom: $\langle \mathit{tt} \rangle \mathit{tt}$



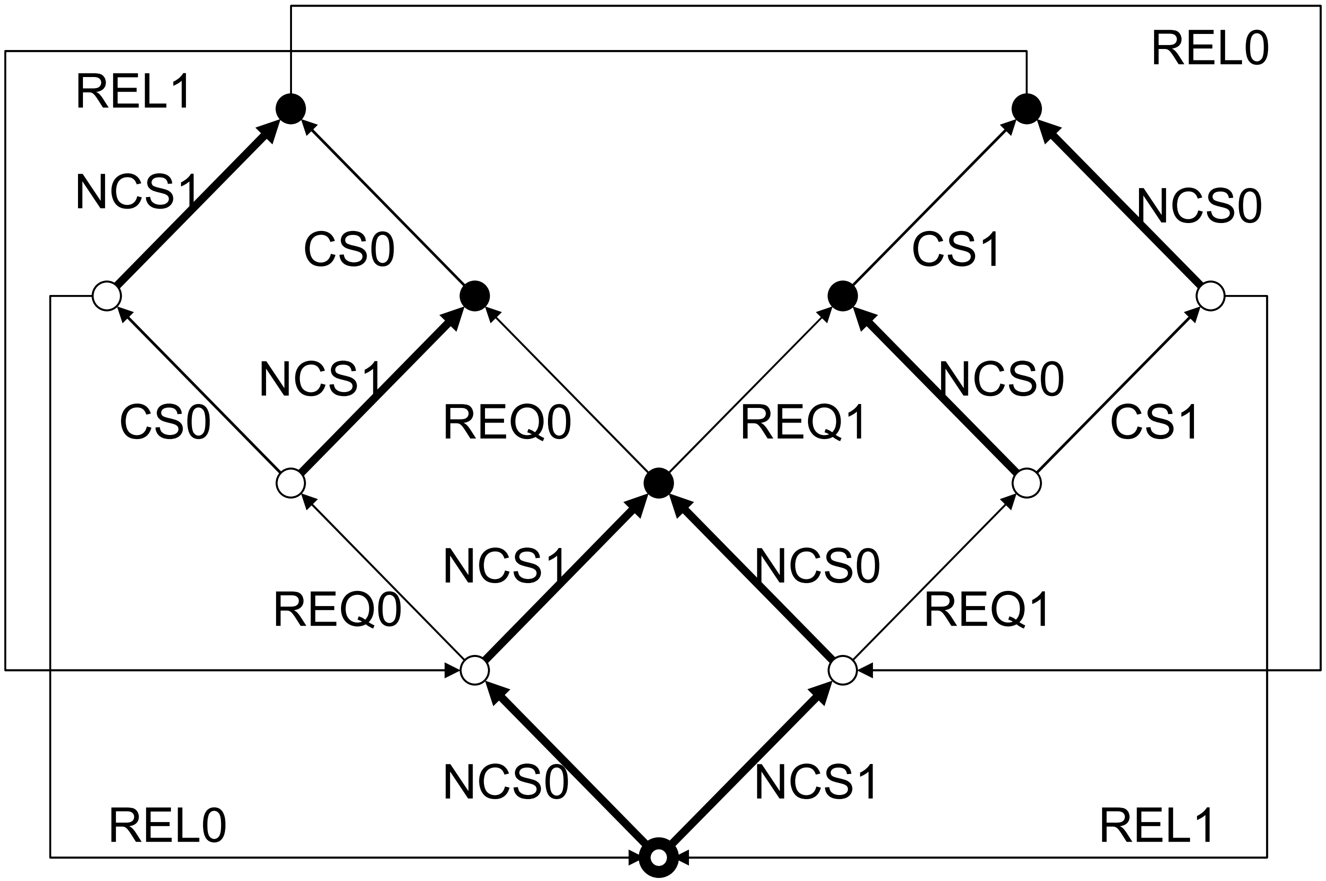
Example (2/4)

Possible execution of a set of actions: $\langle CS_0 \vee CS_1 \rangle tt$



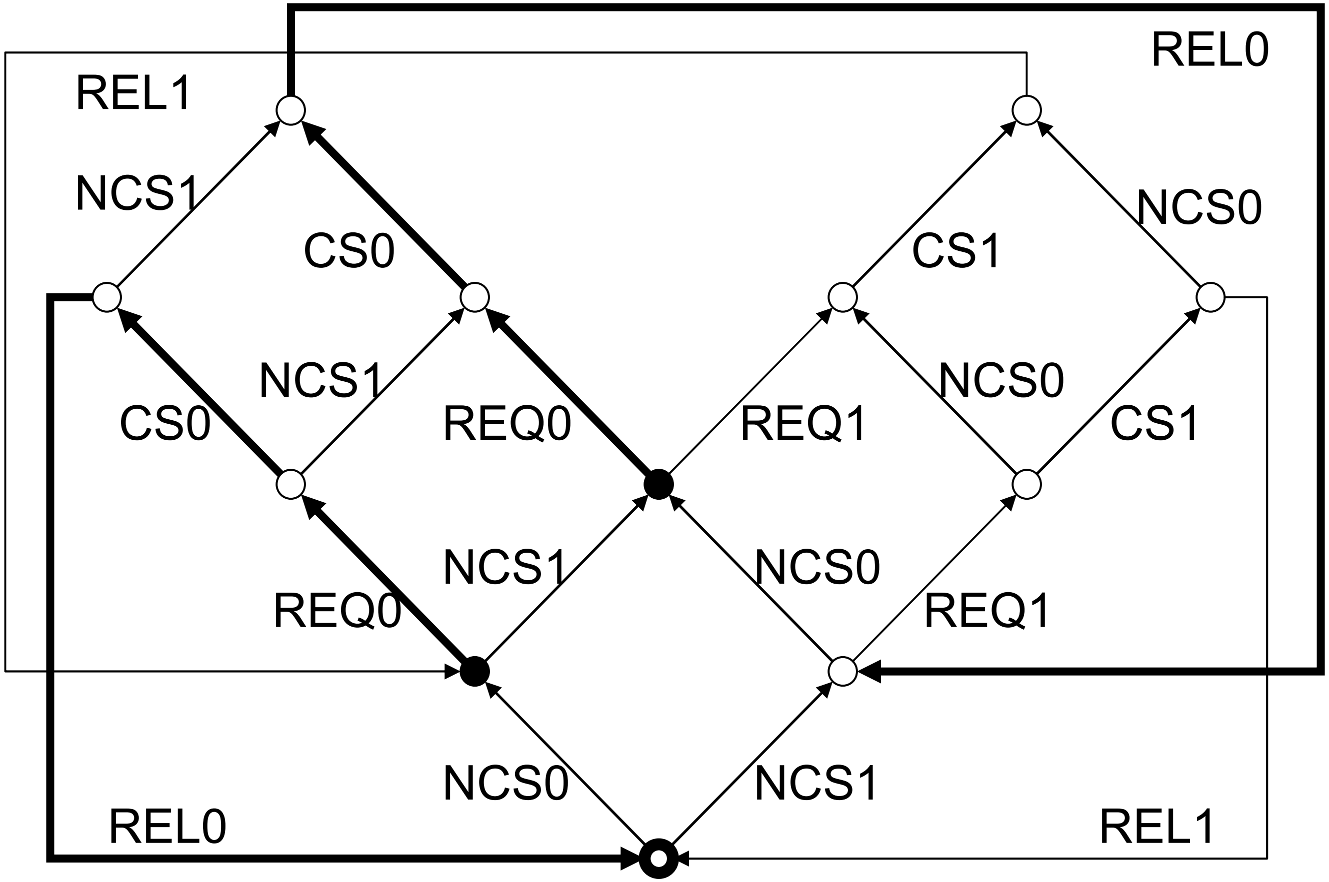
Example (3/4)

Forbidden execution of a set of actions: [$NCS_0 \vee NCS_1$] ff



Example (4/4)

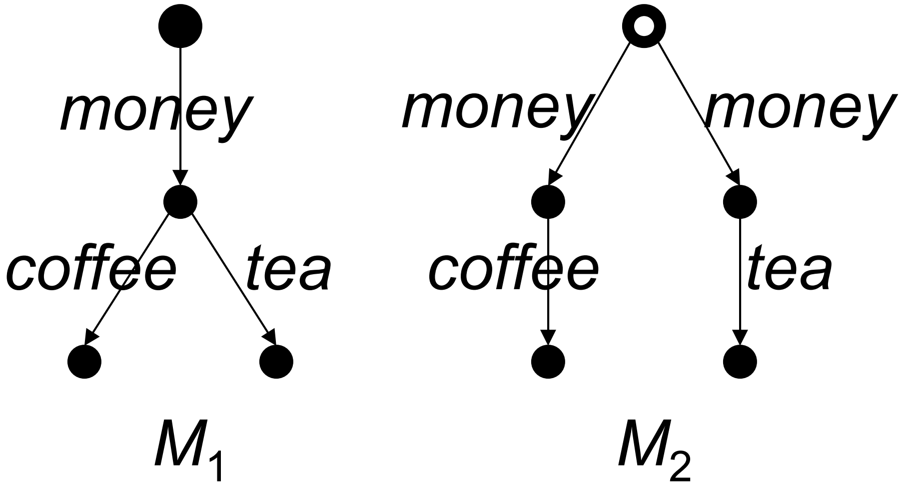
Execution of an action sequence: $\langle$ REQ$_0$ $\rangle$ $\langle$ CS$_0$ $\rangle$ $\langle$ REL$_0$ $\rangle$ tt



Some identities

- Tautologies:
  - $\langle \alpha \rangle \mathrm{ff} = \langle \mathrm{ff} \rangle \varphi = \mathrm{ff}$
  - $[\, \alpha \,]\, \mathrm{tt} = [\, \mathrm{ff} \,]\, \varphi = \mathrm{tt}$
- Distributivity of modalities over $\vee$ and $\wedge$:
  - $\langle \alpha \rangle \varphi_1 \vee \langle \alpha \rangle \varphi_2 = \langle \alpha \rangle (\varphi_1 \vee \varphi_2)$
  - $\langle \alpha_1 \rangle \varphi \vee \langle \alpha_2 \rangle \varphi = \langle \alpha_1 \vee \alpha_2 \rangle \varphi$
  - $[\, \alpha \,]\, \varphi_1 \wedge [\, \alpha \,]\, \varphi_2 = [\, \alpha \,]\, (\varphi_1 \wedge \varphi_2)$
  - $[\, \alpha_1 \,]\, \varphi \wedge [\, \alpha_2 \,]\, \varphi = [\, \alpha_1 \vee \alpha_2 \,]\, \varphi$
- Monotonicity of modalities over φ and α:
  - $(\varphi_1 \Rightarrow \varphi_2) \Rightarrow (\langle \alpha \rangle \varphi_1 \Rightarrow \langle \alpha \rangle \varphi_2) \wedge ([\, \alpha \,]\, \varphi_1 \Rightarrow [\, \alpha \,]\, \varphi_2)$
  - $(\alpha_1 \Rightarrow \alpha_2) \Rightarrow (\langle \alpha_1 \rangle \varphi \Rightarrow \langle \alpha_2 \rangle \varphi) \wedge ([\, \alpha_2 \,]\, \varphi \Rightarrow [\, \alpha_1 \,]\, \varphi)$

Characterization of branching



- Modal formula distinguishing between M_1 and M_2:

$$\varphi = [\,\mathit{money}\,]\ (\langle\, \mathit{coffee} \,\rangle\, \mathrm{tt}\ \wedge\ \langle\, \mathit{tea} \,\rangle\, \mathrm{tt}\,)$$

$$M_1 \models \varphi \quad \text{and} \quad M_2 \not\models \varphi$$

Modal logics

(summary)

- Are able to express simple branching-time properties involving states $s \in S$ and actions $a \in A$ of an LTS
- But:
  - Take into account only a finite number of steps around a state (nesting of modalities)
  - Cannot express properties about transition sequences or subtrees of arbitrary length
- Example: the property

  “from a state *s*, there exists a sequence leading to a state *s*’ where the action *a* is executable”

  cannot be expressed in modal logic

  (it would need a formula $\langle \text{tt} \rangle \langle \text{tt} \rangle \ldots \langle \text{tt} \rangle \langle a \rangle \text{tt}$)

Branching-time logics

- They are logics allowing to reason about the (infinite) execution trees contained in an LTS
- Basic temporal operators:
  - *Potentiality*

    from a state, there exists an outgoing, finite transition sequence leading to a certain state
  - *Inevitability*

    from a state, all outgoing transition sequences lead, after a finite number of steps, to certain states
- Action-based Computation Tree Logic (ACTL) [DeNicola-Vaandrager-90]

ACTL logic

(syntax)

$\varphi ::=$	$tt \mid ff$	boolean constants
$\mid$	$\varphi_1 \vee \varphi_2 \mid \neg\varphi_1$	connectors
$\mid$	$E\,[\,\varphi_{1 \alpha 1}\ U\ \varphi_2\,]$	potentiality 1
$\mid$	$E\,[\,\varphi_{1 \alpha 1}\ U_{\alpha 2}\ \varphi_2\,]$	potentiality 2
$\mid$	$A\,[\,\varphi_{1 \alpha 1}\ U\ \varphi_2\,]$	inevitability 1
$\mid$	$A\,[\,\varphi_{1 \alpha 1}\ U_{\alpha 2}\ \varphi_2\,]$	inevitability 2

ACTL logic

(derived operators)

$EF_\alpha\ \varphi = E\ [\ tt_\alpha\ U\ \varphi\ ]$	basic potentiality
$AF_\alpha\ \varphi = A\ [\ tt_\alpha\ U\ \varphi\ ]$	basic inevitability
$AG_\alpha\ \varphi = \neg\ EF_\alpha\ \neg\varphi$	invariance
$EG_\alpha\ \varphi = \neg\ AF_\alpha\ \neg\varphi$	trajectory
$\langle\ \alpha\ \rangle\ \varphi = E\ [\ tt_{ff}\ U_\alpha\ \varphi\ ]$	possibility
$[\ \alpha\ ]\ \varphi = \neg\ \langle\ \alpha\ \rangle\ \neg\ \varphi$	necessity

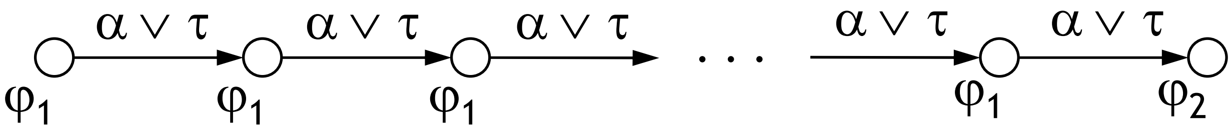
dualities

ACTL logic

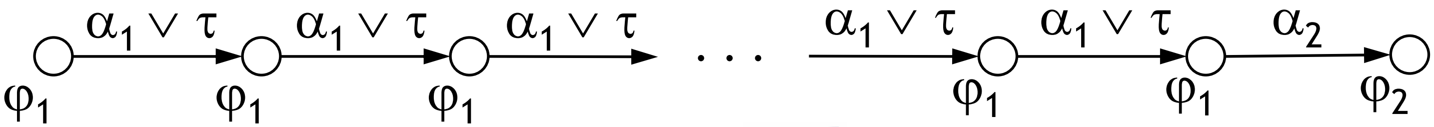
(semantics – potentiality operators)

Let $M = (S, A, T, s_0)$. Interpretation $[[\ \varphi\ ]] \subseteq S$:

- $[[\ E\ [\ \varphi_{1\alpha}\ U\ \varphi_2\ ]\ ]] = \{\ s \in S \mid \exists s(=s_0) \rightarrow^{a0} s_1 \rightarrow^{a1} s_2 \rightarrow \ldots\ .\ \exists k \geq 0.\ \forall 0 \leq i < k.\ (s_i \in [[\ \varphi_1\ ]] \wedge a_i \in [[\ \alpha \vee \tau\ ]]) \wedge s_k \in [[\ \varphi_2\ ]]\ \}$



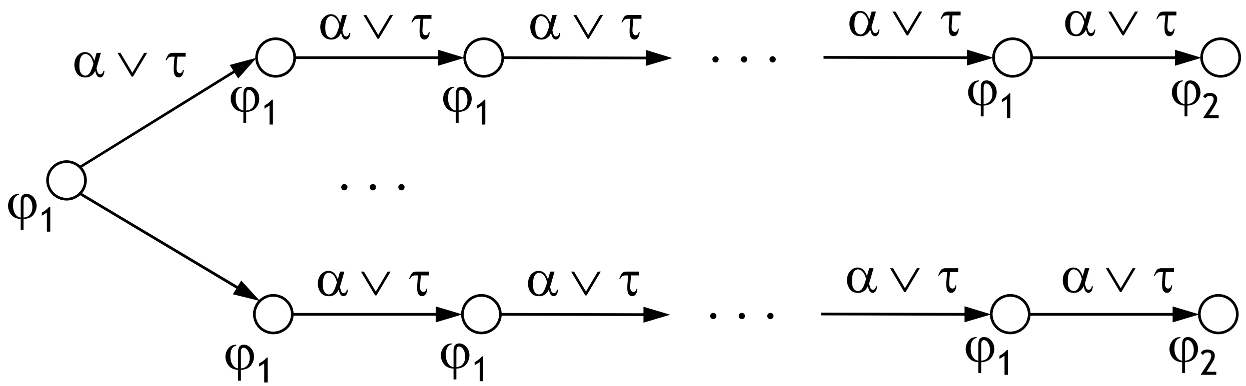
- $[[\ E\ [\ \varphi_{1\alpha 1}\ U_{\alpha 2}\ \varphi_2\ ]\ ]] = \{\ s \in S \mid \forall s(=s_0) \rightarrow^{a0} s_1 \rightarrow^{a1} s_2 \rightarrow \ldots\ .\ \exists k \geq 0.\ \forall 0 \leq i < k.\ (s_i \in [[\ \varphi_1\ ]] \wedge a_i \in [[\ \alpha_1 \vee \tau\ ]] \wedge s_k \in [[\ \varphi_1\ ]] \wedge a_k \in [[\ \alpha_2\ ]] \wedge s_{k+1} \in [[\ \varphi_2\ ]]\ \}$



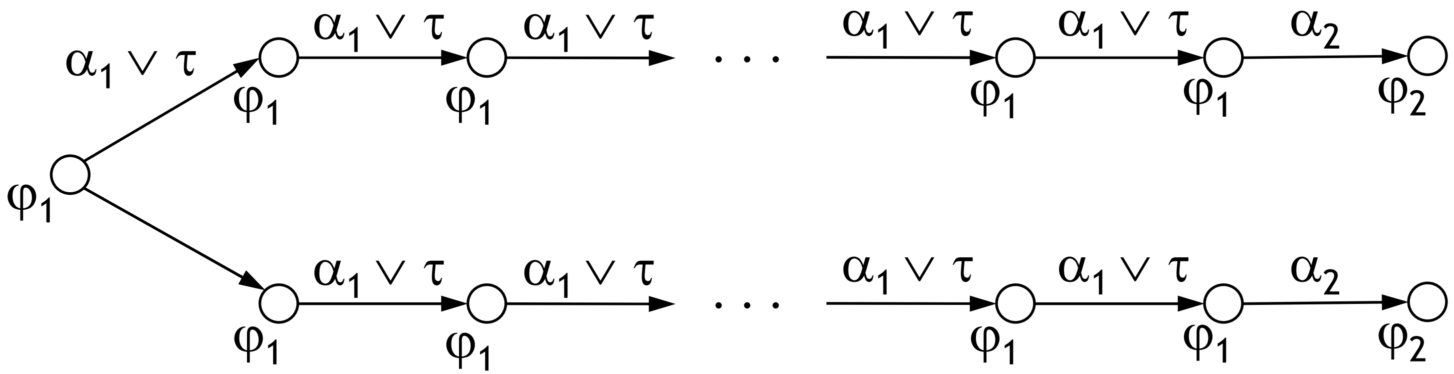
ACTL logic

(semantics – inevitability operators)

- $[[\ A\ [\ \varphi_{1\alpha}\ U\ \varphi_2\ ]\ ]]$:

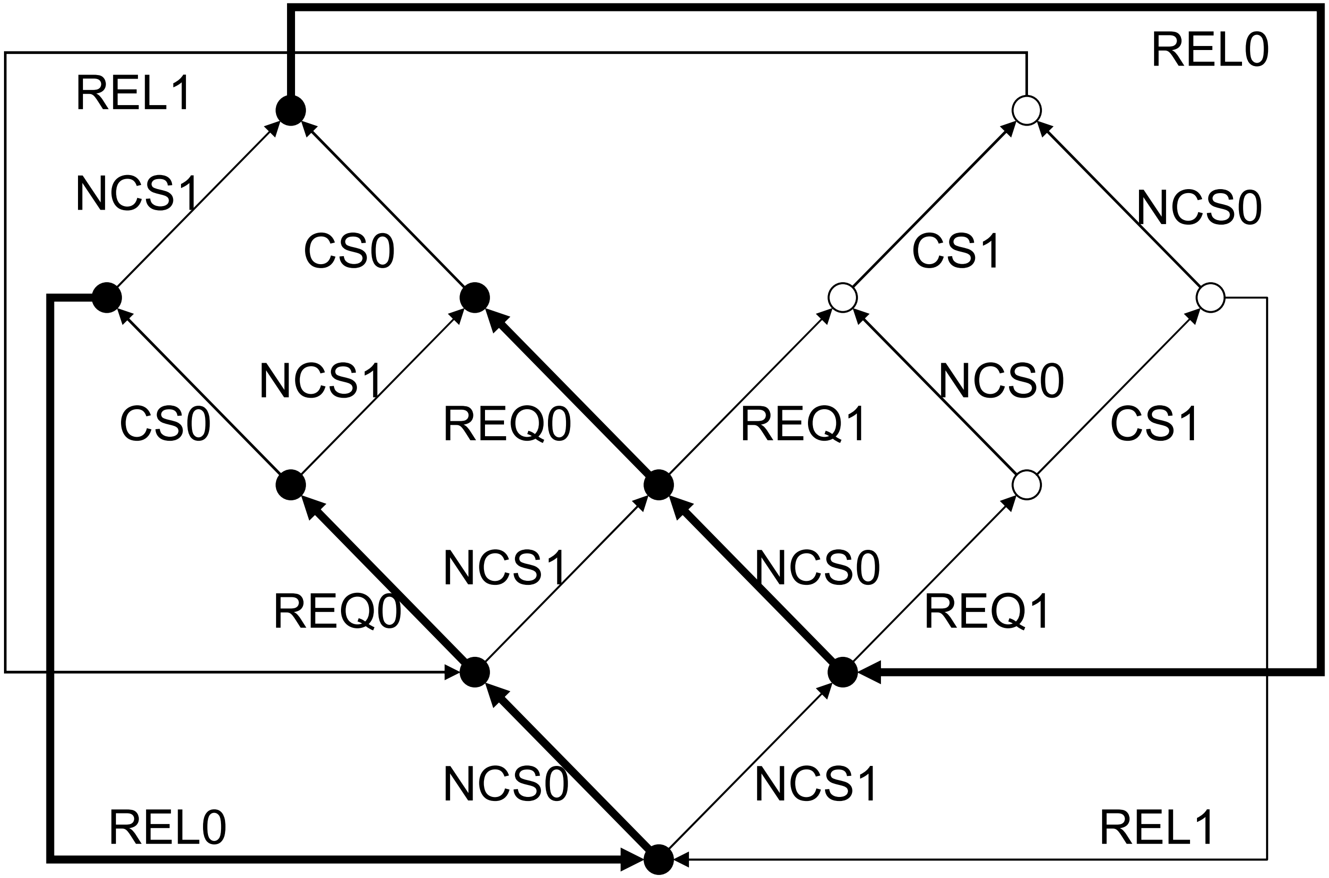


- $[[\ A\ [\ \varphi_{1\alpha 1}\ U_{\alpha 2}\ \varphi_2\ ]\ ]]$:



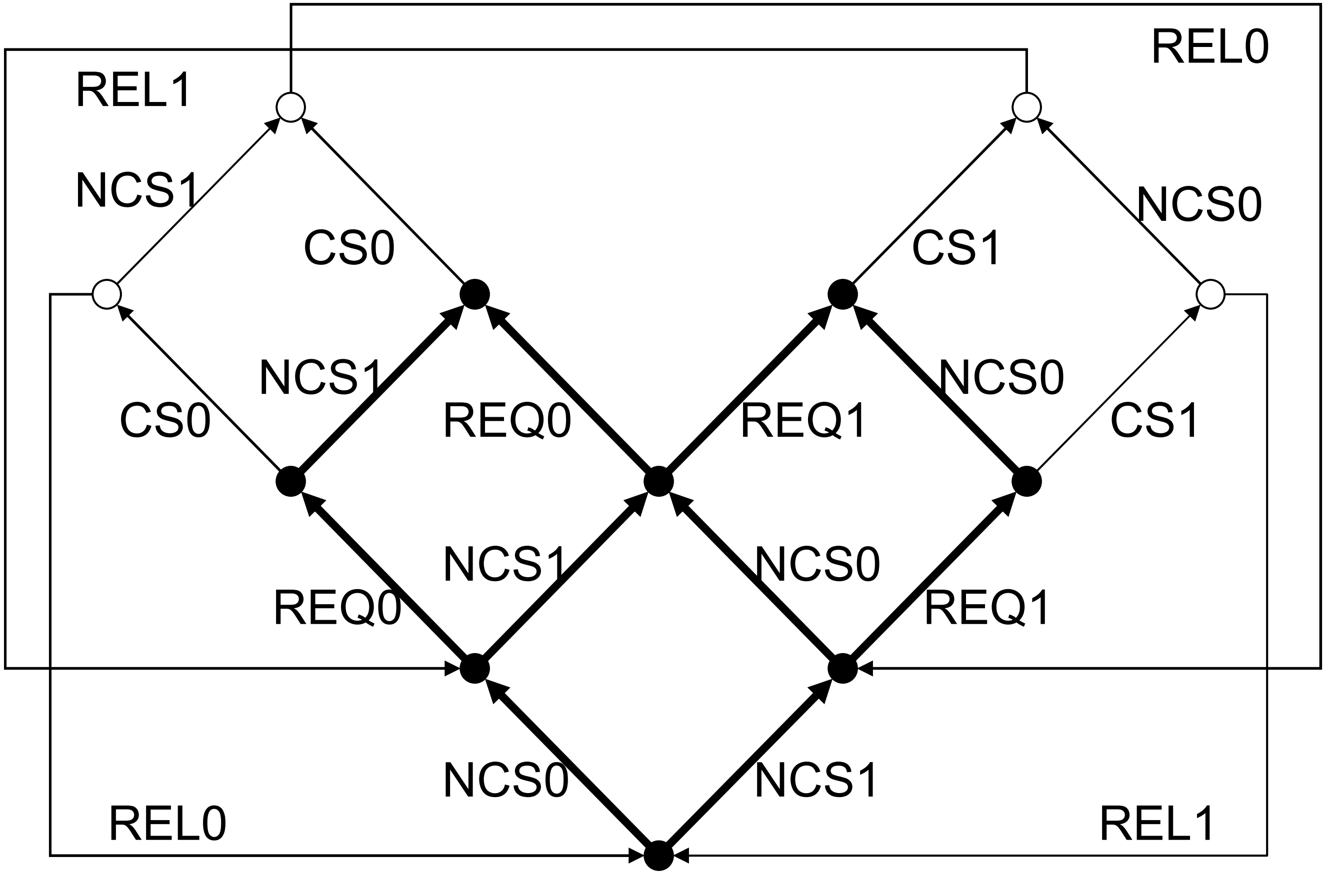
Example (1/4)

Potential reachability: $\mathrm{EF}_{\neg\,\mathrm{REL1}}\,\langle \mathrm{CS}_0 \rangle\,\mathrm{tt}$



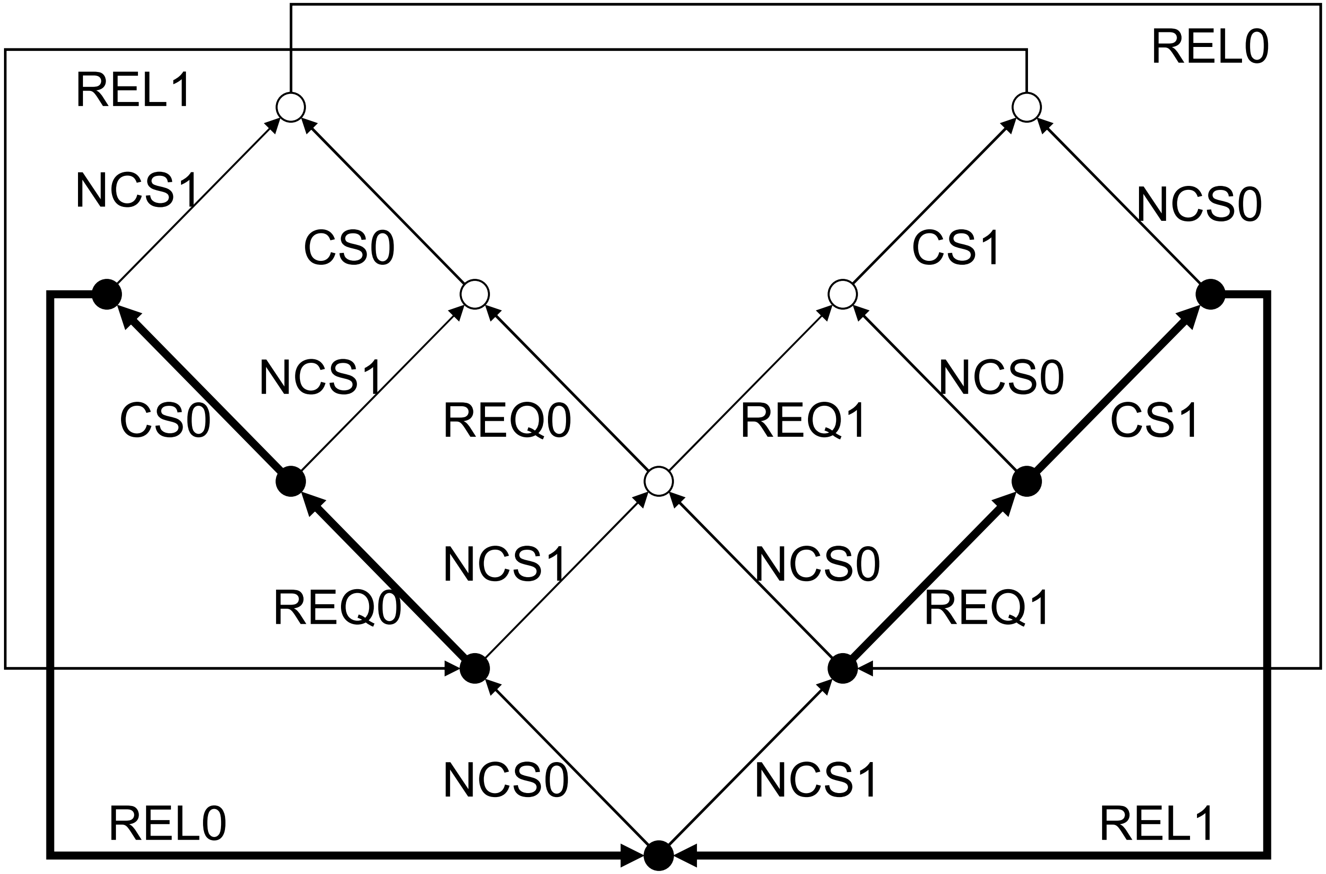
Example (2/4)

Inevitable reachability: $AF_{\neg(REL0 \vee REL1)} \langle CS_0 \vee CS_1 \rangle tt$



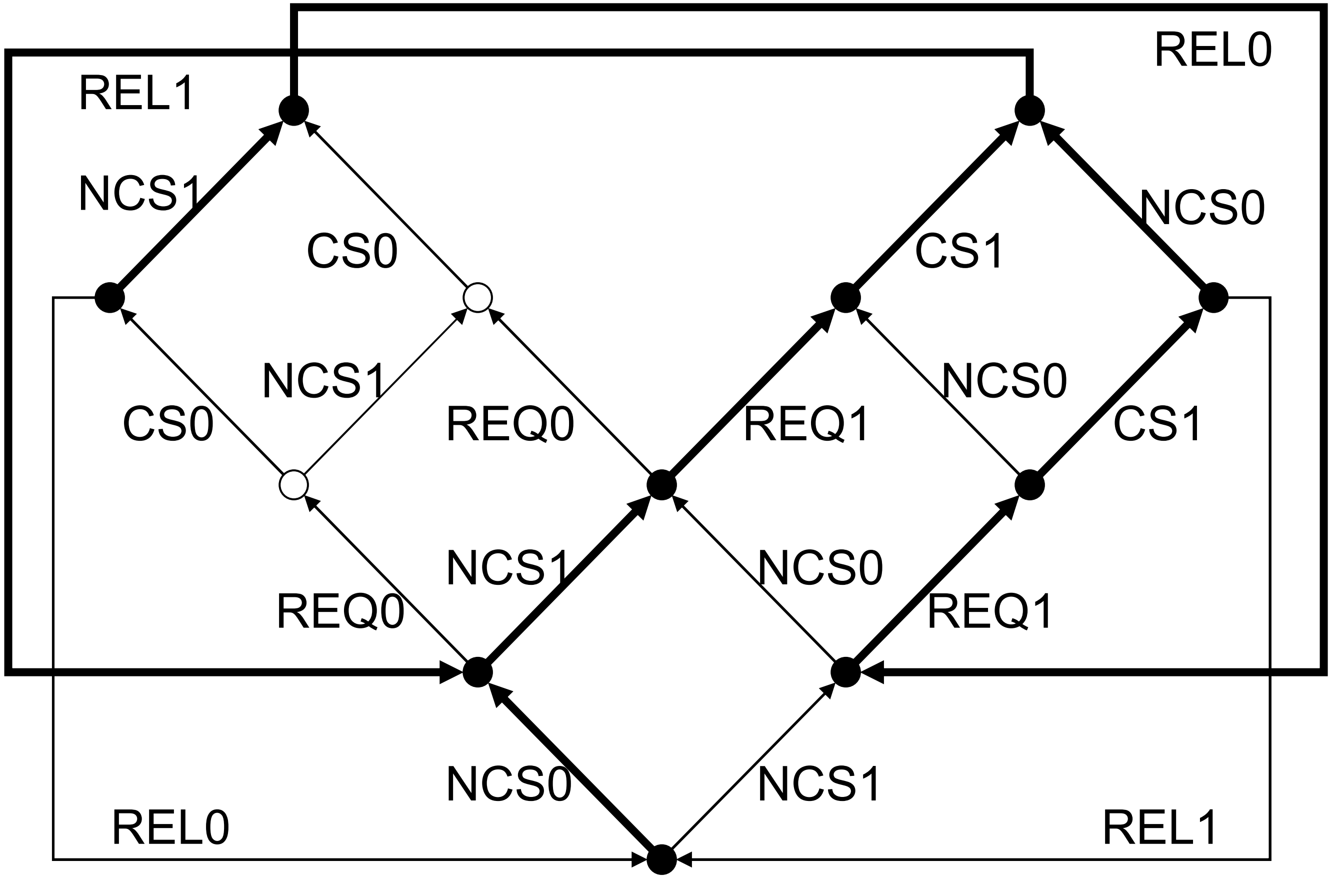
Example (3/4)

Invariance: $AG_{\neg (NCS0 \vee NCS1)} \langle NCS_0 \vee NCS_1 \rangle tt$



Example (4/4)

Trajectory: $EG_{\neg CS0}\,[\,CS_0\,]\,ff$

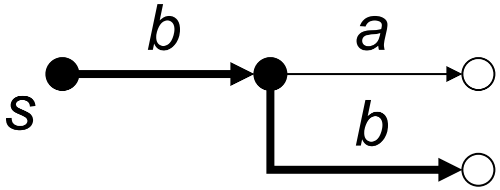


Remark about inevitability

- *Inevitable reachability*: all sequences going out of a state lead to states where an action a is executable

  $AF_{tt} \langle a \rangle tt$

- *Inevitable execution*: all sequences going out of a state contain the action a
- Inevitable execution ⇒ inevitable reachability but the converse does not hold:



$s \models AF_{tt} \langle a \rangle tt$

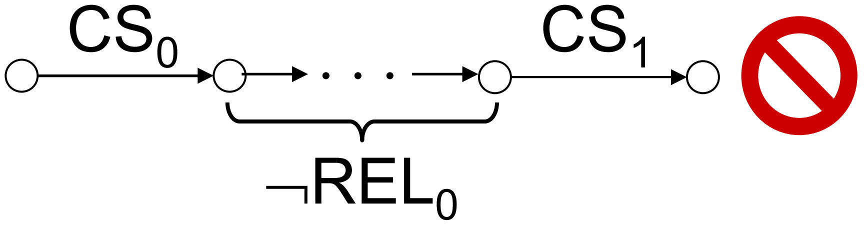
- Inevitable execution must be expressed using the inevitability operators of ACTL:

$$s \not\models A\,[\,tt_{tt}\ U_a\ tt\,]$$

Safety properties

- Informally, safety properties specify that
  "something bad never happens"
  during the execution of the system
- One way of expressing safety properties:
  forbid undesirable execution sequences
  - Mutual exclusion:

$$\neg \langle CS_0 \rangle EF_{\neg REL0} \langle CS_1 \rangle tt$$
$$= [CS_0] AG_{\neg REL0} [CS_1] ff$$



- In ACTL, forbidding a sequence is expressed by combining the $[\alpha] \varphi$ and $AG_\alpha \varphi$ operators

Liveness properties

- Informally liveness properties specify that
  “something good eventually happens”
  during the execution of the system
- One way of expressing liveness properties:

  require desirable execution sequences / trees

  - Potential release of the critical section:
    $\langle NCS_0 \rangle EF_{tt} \langle REQ_0 \rangle EF_{tt} \langle REL_0 \rangle tt$
  - Inevitable access to the critical section:

    $A\ [\ tt_{tt}\ U_{CS0}\ tt\ ]$

- In ACTL, the existence of a sequence is expressed by combining the $\langle \alpha \rangle \varphi$ and $EF_{\alpha}\ \varphi$ operators

Branching-time logics

(summary)

- The temporal operators of ACTL: strictly more powerful than the HML modalities $\langle\ \alpha\ \rangle\ \varphi$ and $[\ \alpha\ ]\ \varphi$
- They allow to express branching-time properties on an unbounded depth in an LTS
- But:
  - They do not allow to express the unbounded repetition of a subsequence
- Example: the property

  "from a state *s*, there exists a sequence *a*.*b*.*a*.*b* … *a*.*b* leading to a state *s*' where an action *c* is executable"

  cannot be expressed in ACTL

Regular logics

- They allow to reason about the regular execution sequences of an LTS
- Basic operators:
  - *Regular formulas*

    two states are linked by a sequence whose concatenated actions form a word of a regular language
  - *Modalities on sequences*

    from a state, some (all) outgoing regular transition sequences lead to certain states
- Propositional Dynamic Logic (PDL) [Fischer-Ladner-79]

Regular formulas

(syntax)

$\beta ::=$	α	one-step sequence
$\mid$	nil	empty sequence
$\mid$	$\beta_1 . \beta_2$	concatenation
$\mid$	$\beta_1 \mid \beta_2$	choice
$\mid$	$\beta_1{}^*$	iteration ($\geq$ 0 times)
$\mid$	$\beta_1{}^+$	iteration ($\geq$ 1 times)

- Some identities:

$$\text{nil} = \text{ff}\ ^* \qquad \beta^+ = \beta\ .\ \beta^*$$

Regular formulas

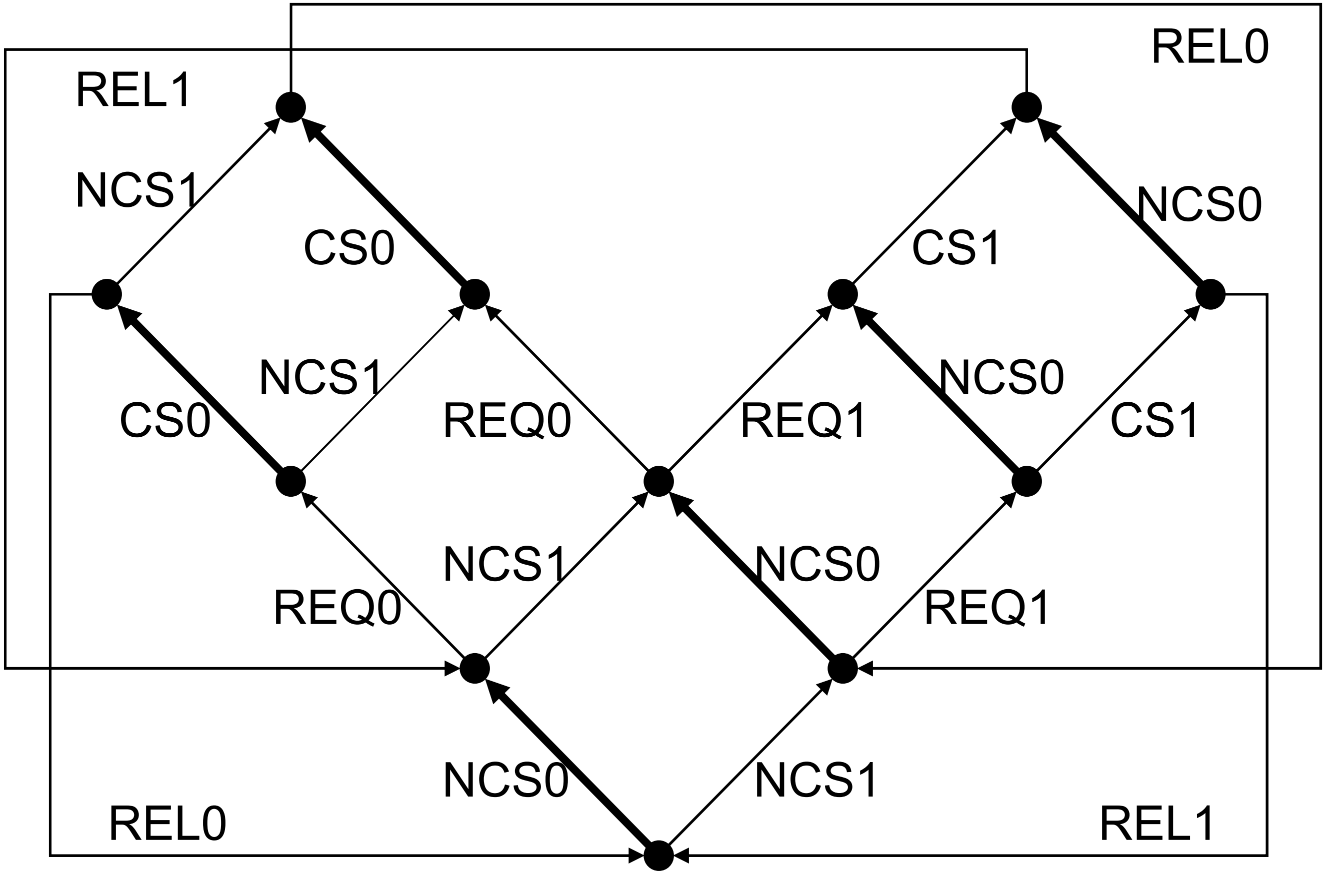
(semantics)

Let $M = (S, A, T, s_0)$. Interpretation $[[\ \beta\ ]] \subseteq S \times S$:

- $[[\ \alpha\ ]] = \{ (s, s') \mid \exists a \in [[\ \alpha\ ]] \,.\, (s, a, s') \in T \}$
- $[[\ \text{nil}\ ]] = \{ (s, s) \mid s \in S \}$ (identity)
- $[[\ \beta_1 \,.\, \beta_2\ ]] = [[\ \beta_1\ ]] \circ [[\ \beta_2\ ]]$ (composition)
- $[[\ \beta_1 \mid \beta_2\ ]] = [[\ \beta_1\ ]] \cup [[\ \beta_2\ ]]$ (union)
- $[[\ \beta_1^*\ ]] = [[\ \beta_1\ ]]^*$ (transitive refl. closure)
- $[[\ \beta_1^+\ ]] = [[\ \beta_1\ ]]^+$ (transitive closure)

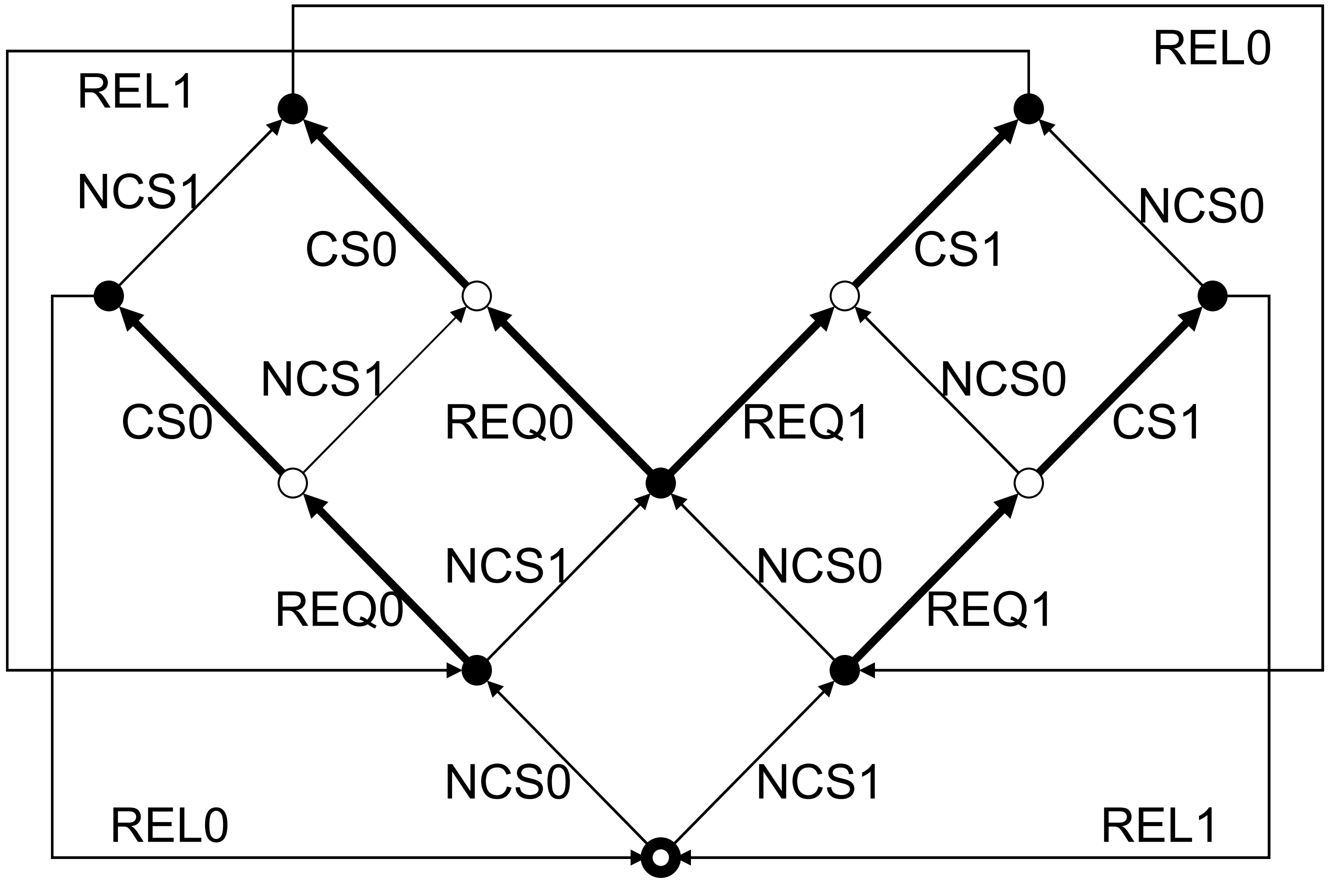
Example (1/3)

One-step sequences: $NCS_0 \vee CS_0$



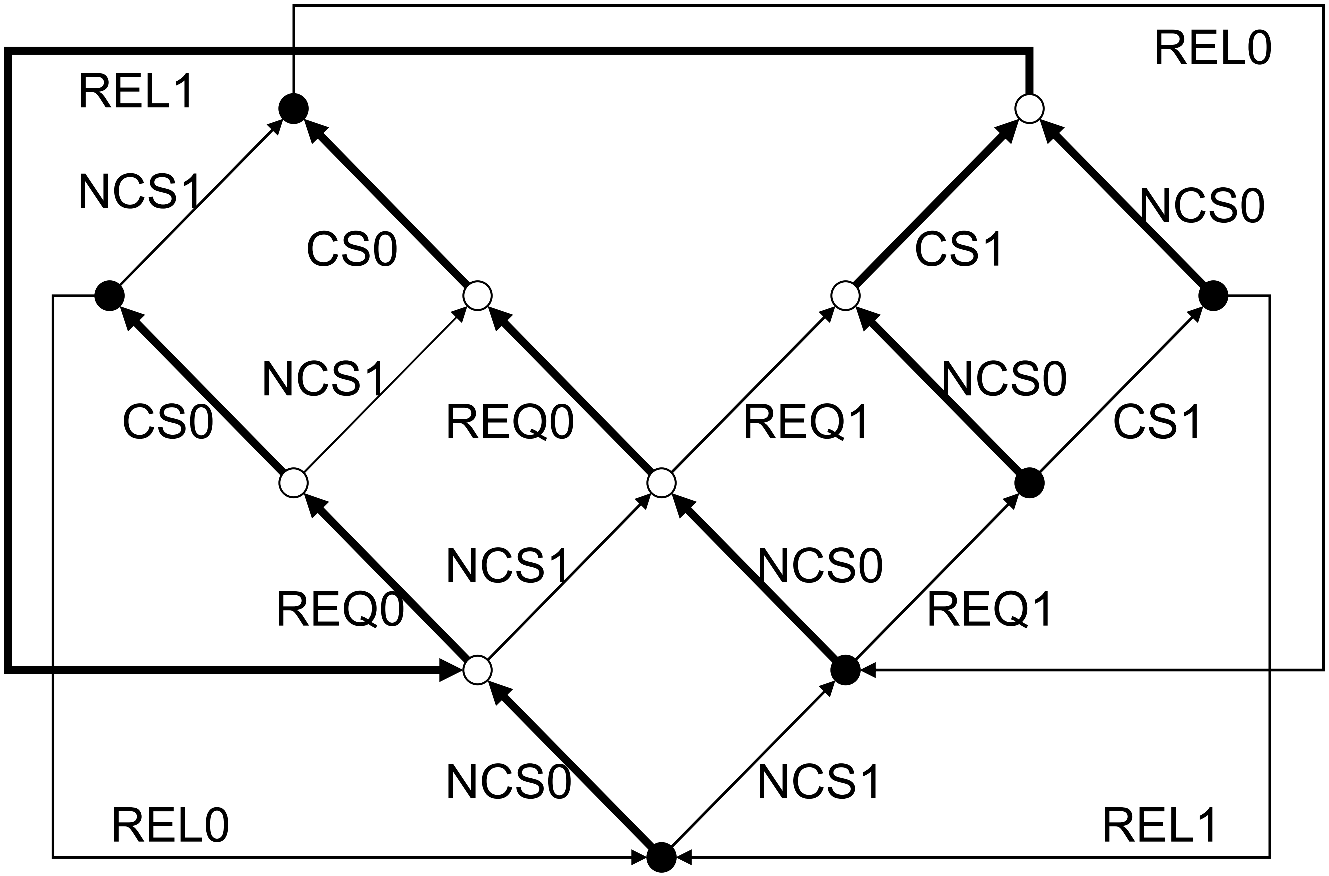
Example (2/3)

Alternative sequences: $(REQ_0 \,.\, CS_0)$ | $(REQ_1 \,.\, CS_1)$



Example (3/3)

Sequences with repetition: NCS_0 . $(\neg NCS_1)^*$. CS_0



PDL logic

(syntax)

$\varphi ::=$	$tt \mid ff$	boolean constants
$\mid$	$\varphi_1 \vee \varphi_2$	disjunction
$\mid$	$\varphi_1 \wedge \varphi_2$	conjunction
$\mid$	$\neg\varphi_1$	negation
$\mid$	$\langle \beta \rangle \varphi_1$	possibility
$\mid$	$[\beta] \varphi_1$	necessity

- Duality: $[\beta] \varphi = \neg \langle \beta \rangle \neg\varphi$

PDL logic

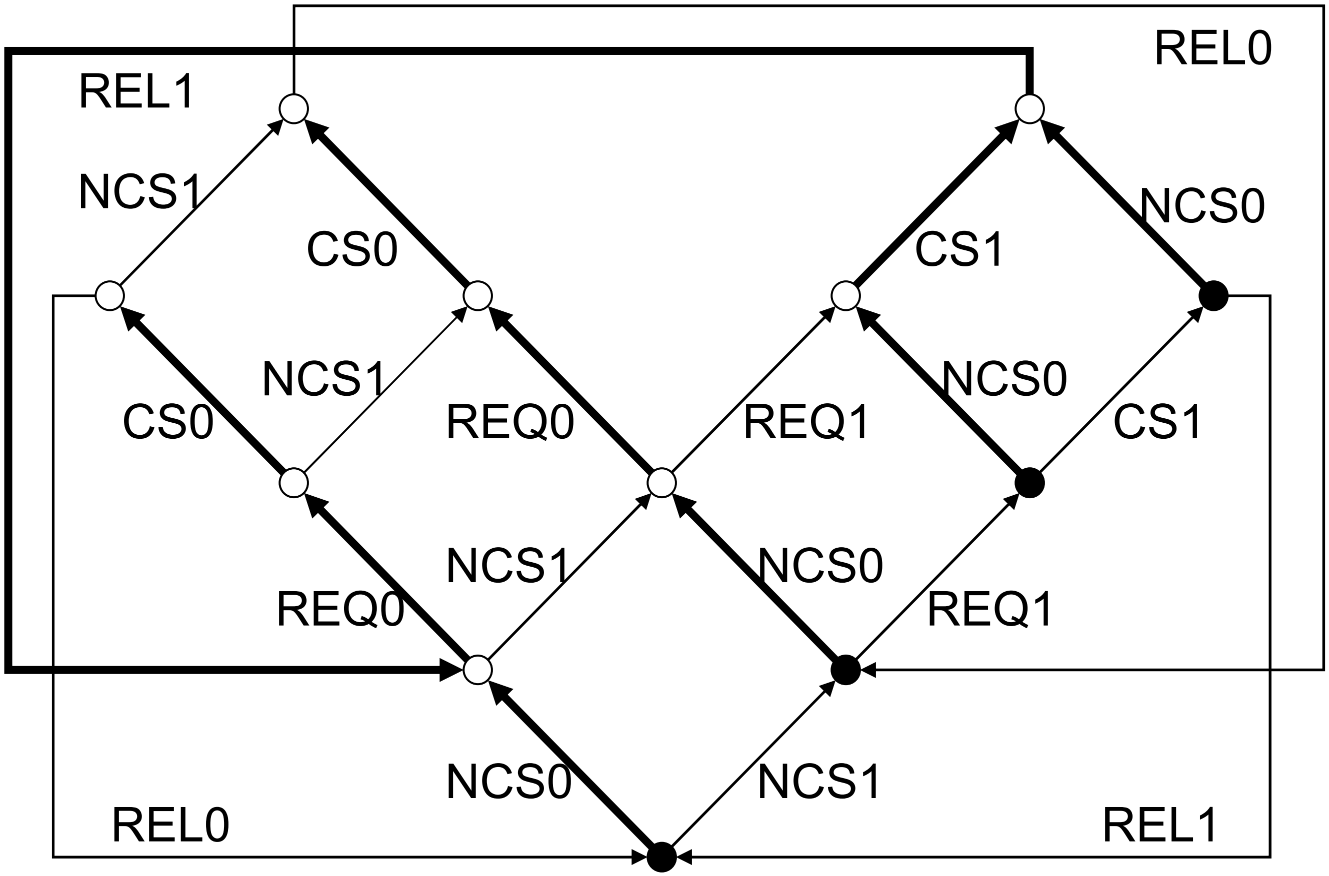
(semantics)

Let $M = (S, A, T, s_0)$. Interpretation $[[\ \varphi\ ]] \subseteq S$:

- $[[\ \mathrm{tt}\ ]] = S$
- $[[\ \mathrm{ff}\ ]] = \varnothing$
- $[[\ \varphi_1 \vee \varphi_2\ ]] = [[\ \varphi_1\ ]] \cup [[\ \varphi_2\ ]]$
- $[[\ \varphi_1 \wedge \varphi_2\ ]] = [[\ \varphi_1\ ]] \cap [[\ \varphi_2\ ]]$
- $[[\ \neg\varphi_1\ ]] = S \setminus [[\ \varphi_1\ ]]$
- $[[\ \langle\ \beta\ \rangle\ \varphi_1\ ]] = \{\ s \in S \mid \exists\ s' \in S\ .$
  $(s, s') \in [[\ \beta\ ]] \wedge s' \in [[\ \varphi_1\ ]]\ \}$
- $[[\ [\ \beta\ ]\ \varphi_1\ ]] = \{\ s \in S \mid \forall\ s' \in S\ .$
  $(s, s') \in [[\ \beta\ ]] \Rightarrow s' \in [[\ \varphi_1\ ]]\ \}$

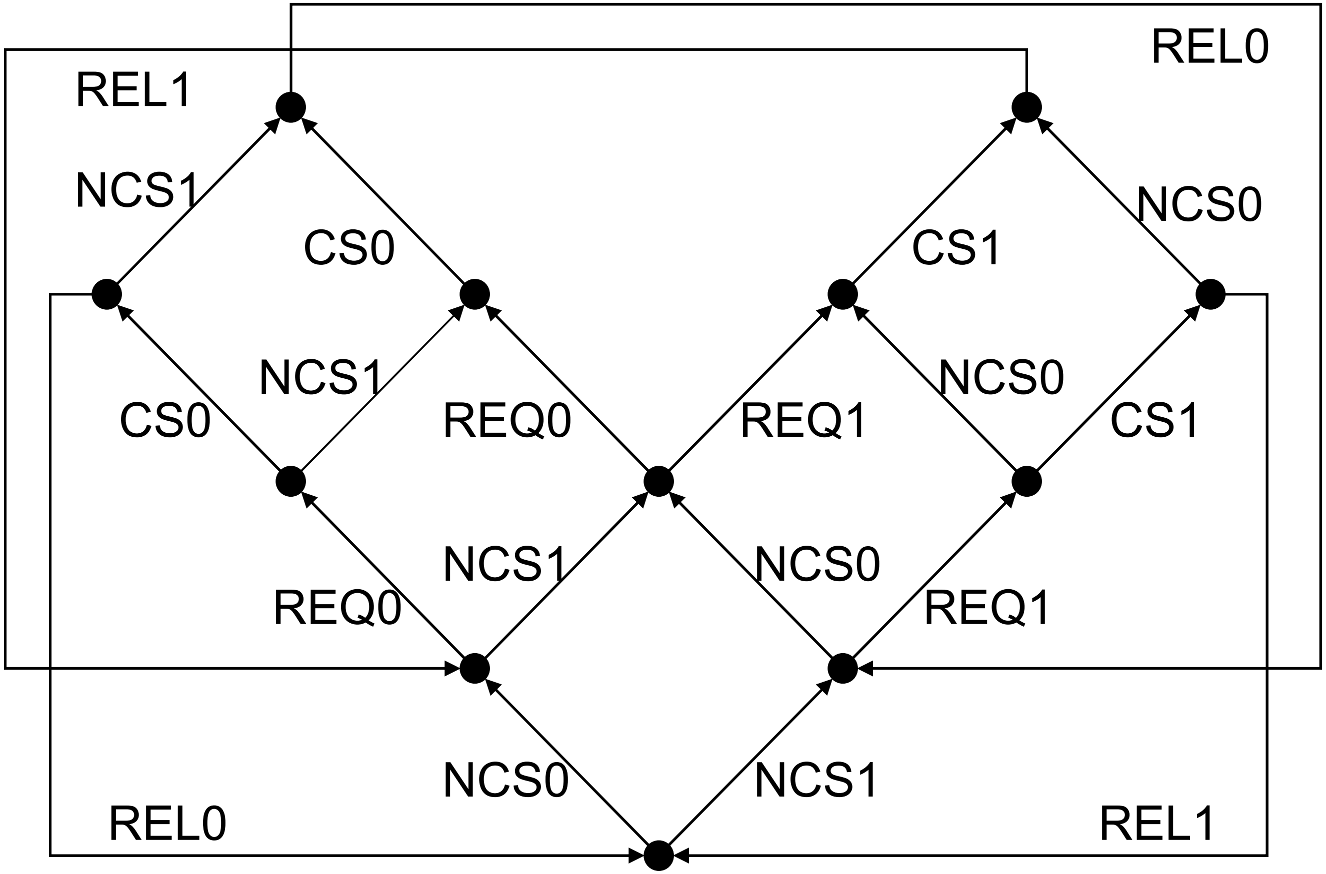
Example (1/2)

Potential reachability of critical section: $\langle NCS_0 \;.\; tt\;^{*} \;.\; CS_0 \rangle\; tt$



Example (2/2)

Mutual exclusion: $[\ CS_0\ .\ (\neg REL_0)^*\ .\ CS_1\ ]\ ff$

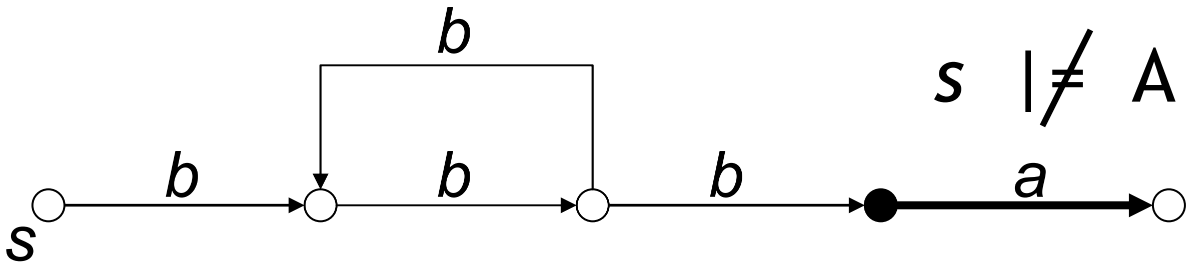


Some identities

- Distributivity of regular operators over $\langle\ \rangle$ and [ ]:
  - $\langle \beta_1 . \beta_2 \rangle \varphi = \langle \beta_1 \rangle \langle \beta_2 \rangle \varphi$
  - $\langle \beta_1 \mid \beta_2 \rangle \varphi = \langle \beta_1 \rangle \varphi \vee \langle \beta_2 \rangle \varphi$
  - $\langle \beta^* \rangle \varphi = \varphi \vee \langle \beta \rangle \langle \beta^* \rangle \varphi$
  - $[\beta_1 . \beta_2] \varphi = [\beta_1] [\beta_2] \varphi$
  - $[\beta_1 \mid \beta_2] \varphi = [\beta_1] \varphi \wedge [\beta_2] \varphi$
  - $[\beta^*] \varphi = \varphi \wedge [\beta] [\beta^*] \varphi$
- Potentiality and invariance operators of ACTL:
  - $EF_\alpha \varphi = \langle \alpha^* \rangle \varphi$
  - $AG_\alpha \varphi = [\alpha^*] \varphi$

Fairness properties

- Problem: from the initial state of the LTS, there is no inevitable execution of action $CS_0 \Rightarrow$ process P_1 can enter its critical section indefinitely often



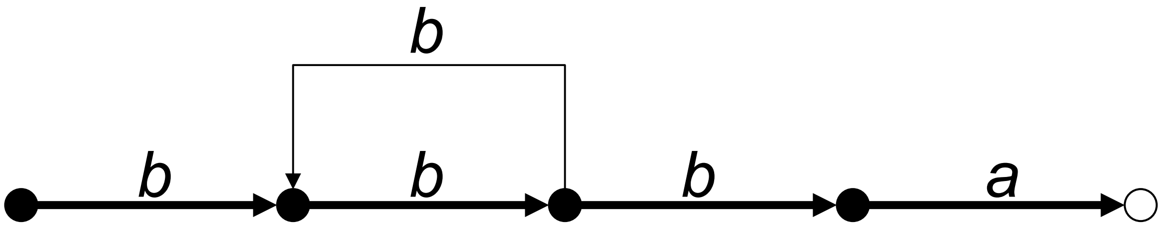
$s \not\models A\,[\,tt_{tt}\ U_a\ tt\,]$

- *Fair execution* of an action *a*: from a state, all transition sequences that do not cycle indefinitely contain action *a*
- Action-based counterpart of the *fair reachability of predicates* [Queille-Sifakis-82]

Fair execution

- Fair execution of an action *a* expressed in PDL:

$$\text{fair}\ (a) = [\ (\neg a)^*\ ]\ \langle\ \text{tt}^*.\ a\ \rangle\ \text{tt}$$

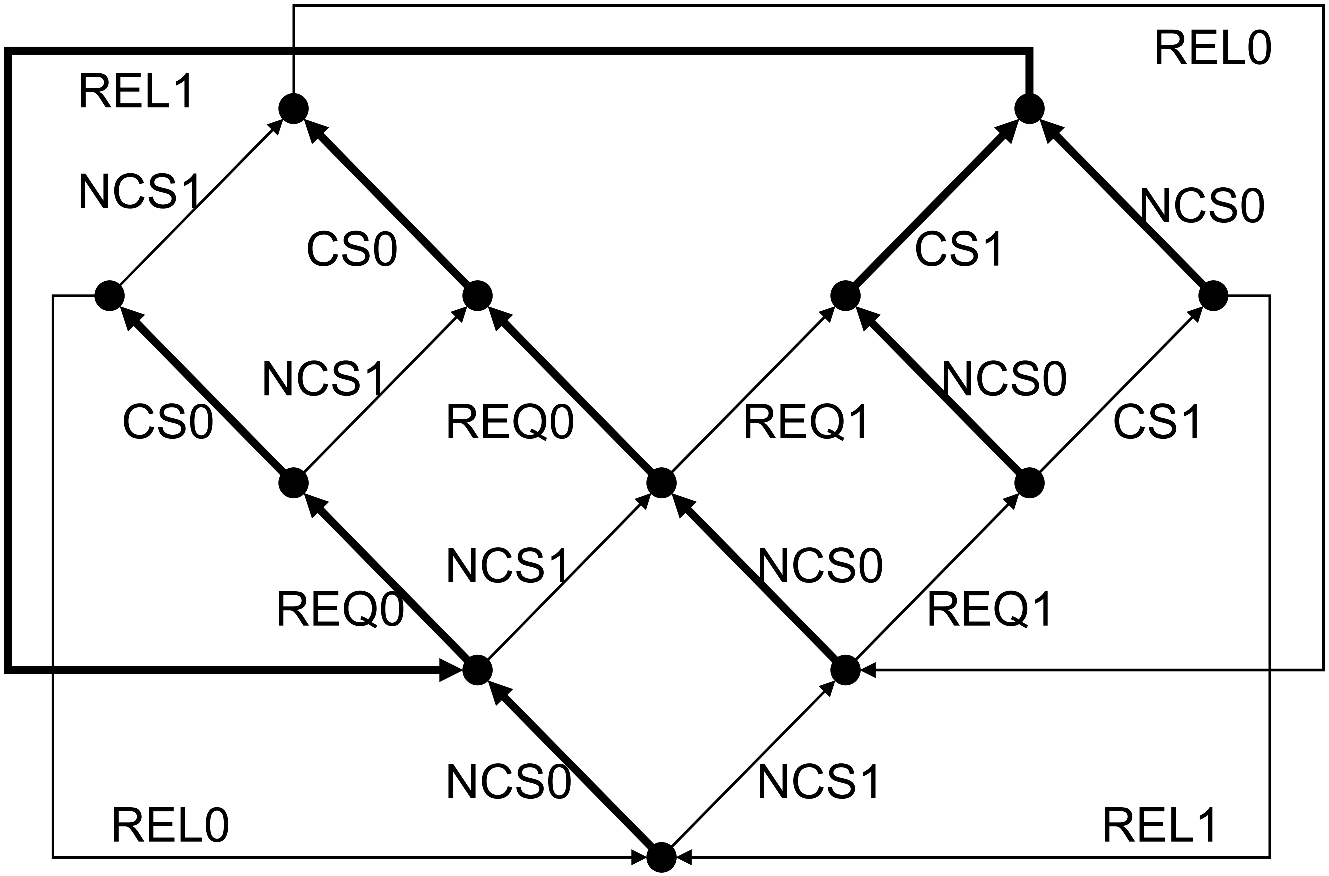


- Equivalent formulation in ACTL:

$$\text{fair}\ (a) = \text{AG}_{\neg a}\ \text{EF}_{\text{tt}}\ \langle\ a\ \rangle\ \text{tt}$$

Example

Fair execution of critical section: $[\ (\neg CS_0)^*\ ]\ \langle\ tt^*.\ CS_0\ \rangle\ tt$



Regular logics

(summary)

- They allow a direct and natural description of regular execution sequences in LTSs

- More intuitive description of safety properties:
  - Mutual exclusion:

    $[\ CS_0\ ]\ AG_{\neg REL0}\ [\ CS_1\ ]\ ff \quad =$ (in ACTL)

    $[\ CS_0\ .\ (\neg REL_0)^*\ .\ CS_1\ ]\ ff$ (in PDL)

- But:
  - Not sufficiently powerful to express inevitability operators (expressiveness uncomparable with branching-time logics)

Fixed point logics

- Very expressive logics (“temporal logic assembly languages”) allowing to characterize finite or infinite tree-like patterns in LTSs
- Basic temporal operators:
  - *Minimal fixed point* (μ)

    “recursive function” defined over the LTS: *finite* execution trees going out of a state
  - *Maximal fixed point* (ν)

    dual of the minimal fixed point operator: *infinite* execution trees going out of a state
- Modal mu-calculus [Kozen-83,Stirling-01]

Modal mu-calculus

(syntax)

φ ::=	tt | ff	boolean constants
|	$\varphi_1 \vee \varphi_2$ | $\neg\varphi_1$	connectors
|	$\langle \alpha \rangle \varphi_1$	possibility
|	$[\alpha] \varphi_1$	necessity
|	X	propositional variable
|	$\mu X . \varphi_1$	minimal fixed point
|	$\nu X . \varphi_1$	maximal fixed point

- Duality: $\nu X . \varphi = \neg \mu X . \neg \varphi [\neg X / X]$

Syntactic restrictions

- Syntactic monotonicity [Kozen-83]
  - Necessary to ensure the existence of fixed points
  - In every formula $\sigma X \,.\, \varphi\,(X)$, where $\sigma \in \{\, \mu, \nu \,\}$, every free occurrence of X in φ falls in the scope of an even number of negations

$$\mu X \,.\, \langle\, a \,\rangle\, X \vee \neg \langle\, b \,\rangle\, X$$

🚫

- Alternation depth 1 [Emerson-Lei-86]
  - Necessary for efficient (linear-time) verification
  - In every formula $\mu X \,.\, \varphi\,(X)$, every maximal subformula $\nu Y \,.\, \varphi'\,(Y)$ of φ is closed

$$\mu X \,.\, \langle\, a \,\rangle\, \nu Y \,.\, ([\, b \,]\, Y \wedge [\, c \,]\, X)$$

🚫

Modal mu-calculus

(semantics)

Let $M = (S, A, T, s_0)$ and $\rho : X \rightarrow 2^S$ a context mapping propositional variables to state sets. Interpretation $[[\ \varphi\ ]] \subseteq S$:

- $[[\ X\ ]]\ \rho = \rho\ (X\ )$
- $[[\ \mu X\ .\ \varphi\ ]]\ \rho = \bigcup_{k \geq 0} \Phi_\rho{}^k\ (\emptyset)$
- $[[\ \nu X\ .\ \varphi\ ]]\ \rho = \bigcap_{k \geq 0} \Phi_\rho{}^k\ (S)$

where $\Phi_\rho : 2^S \rightarrow 2^S$,

$$\Phi_\rho\ (U) = [[\ \varphi\ ]]\ \rho\ [\ U\ /\ X\ ]$$

Minimal fixed point

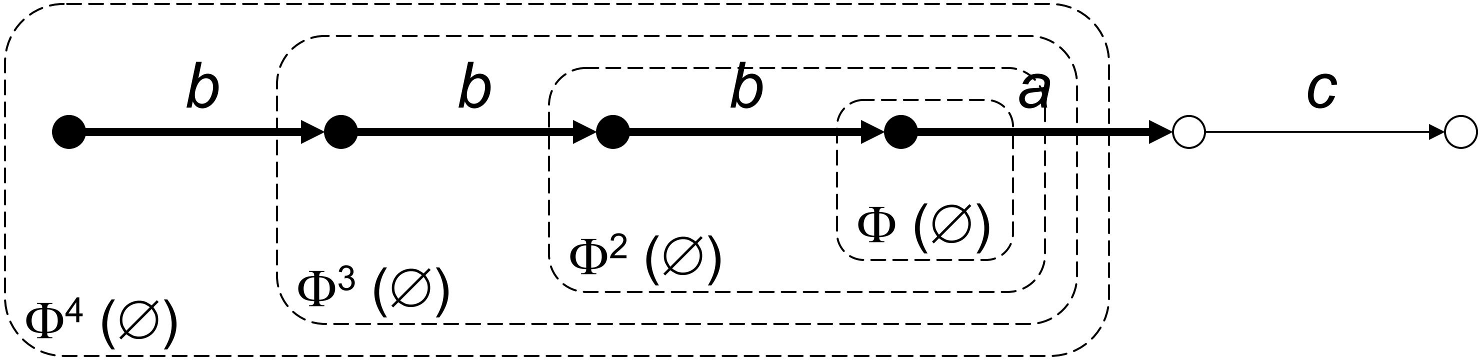
- Potential reachability of an action *a* (existence of a sequence leading to a transition labeled by *a*):

$$\mu X \,.\, \langle\, a \,\rangle\, \mathrm{tt} \vee \langle\, \mathrm{tt} \,\rangle\, X$$

- Associated functional:

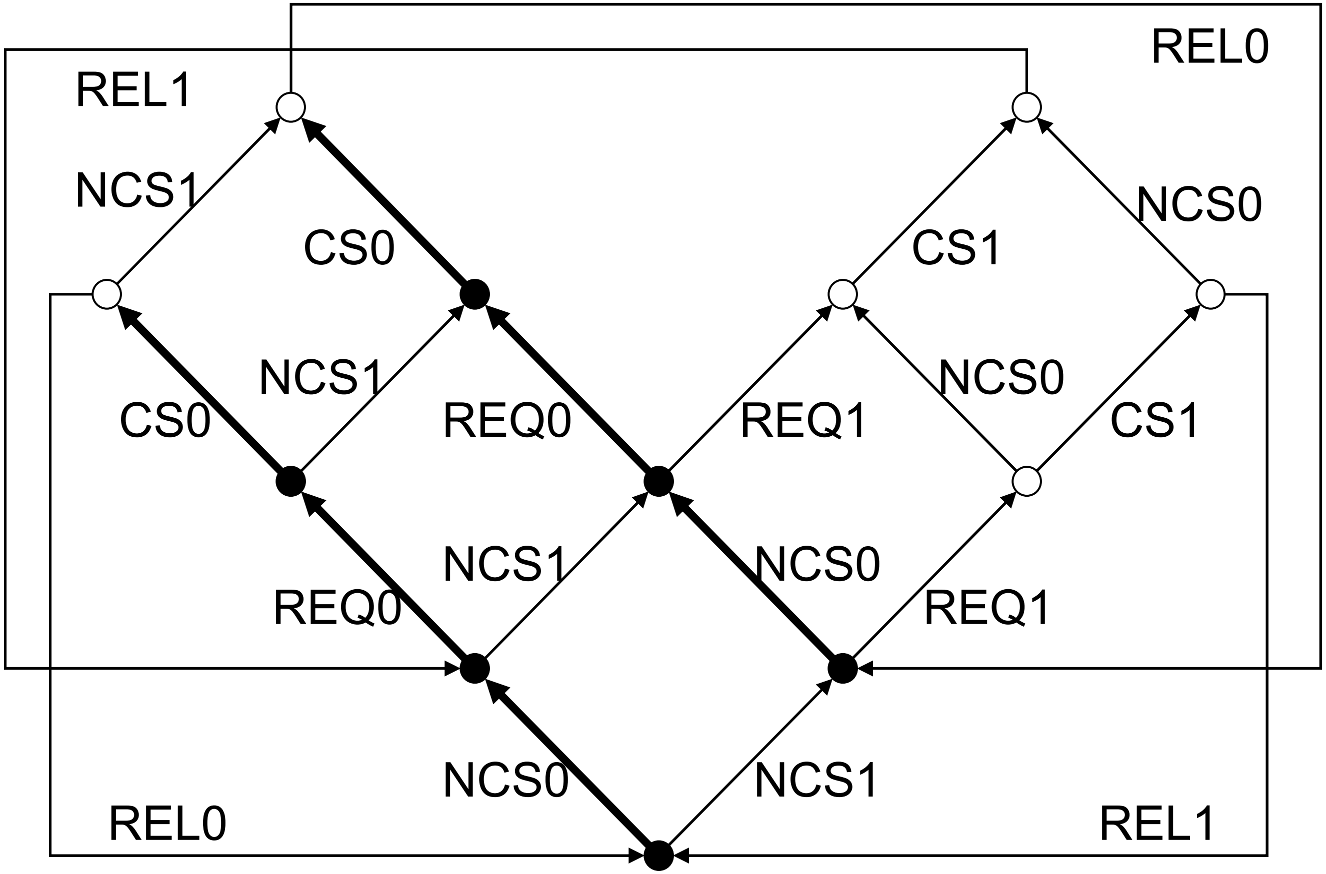
$$\Phi\ (U) = [[\, \langle\, a \,\rangle\, \mathrm{tt} \vee \langle\, \mathrm{tt} \,\rangle\, X \,]]\ \ [\, U \,/\, X \,]$$

- Evaluation on an LTS:



Example

Potential reachability: $\mu X \,.\, \langle CS_0 \rangle tt \vee \langle \neg(REL_1 \vee REL_0) \rangle X$



Maximal fixed point

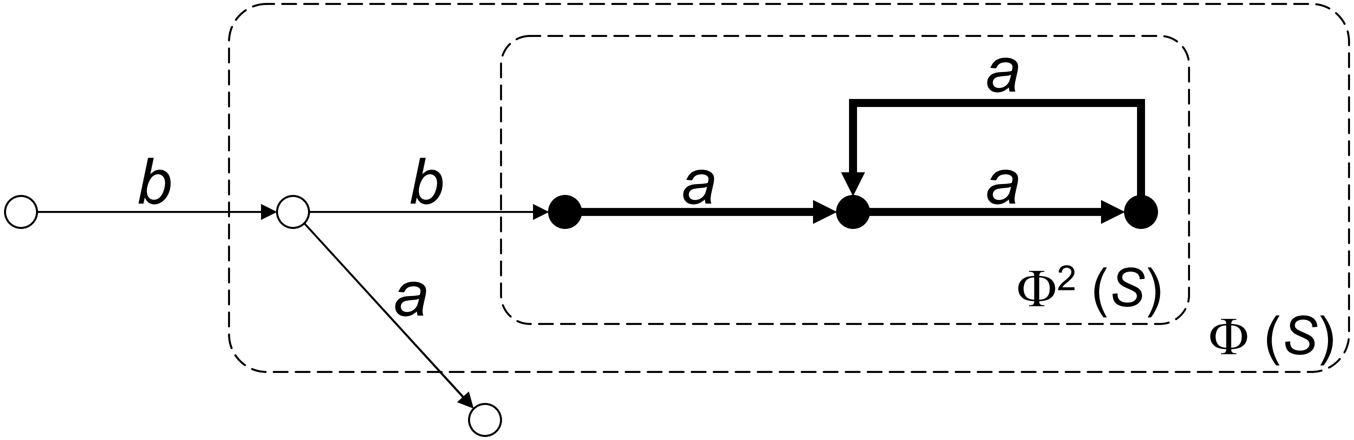
- Infinite repetition of an action *a* (existence of a cycle containing only transitions labeled by *a*):

$$\nu X \,.\, \langle\, a \,\rangle\, X$$

- Associated functional:

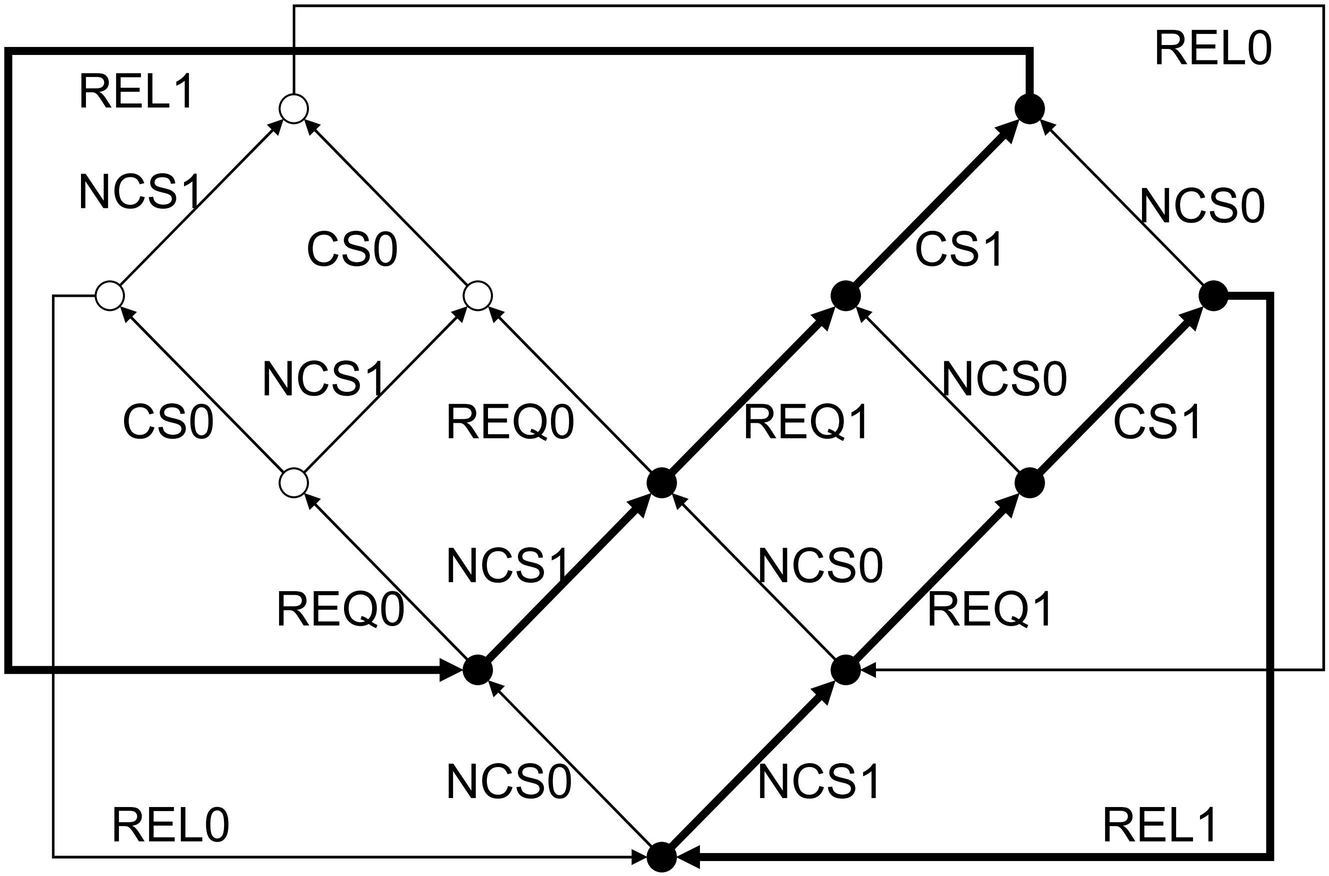
$$\Phi\ (U) = [[\ \langle\, a \,\rangle\, X\ ]]\ \ [\ U\ /\ X\ ]$$

- Evaluation on an LTS:



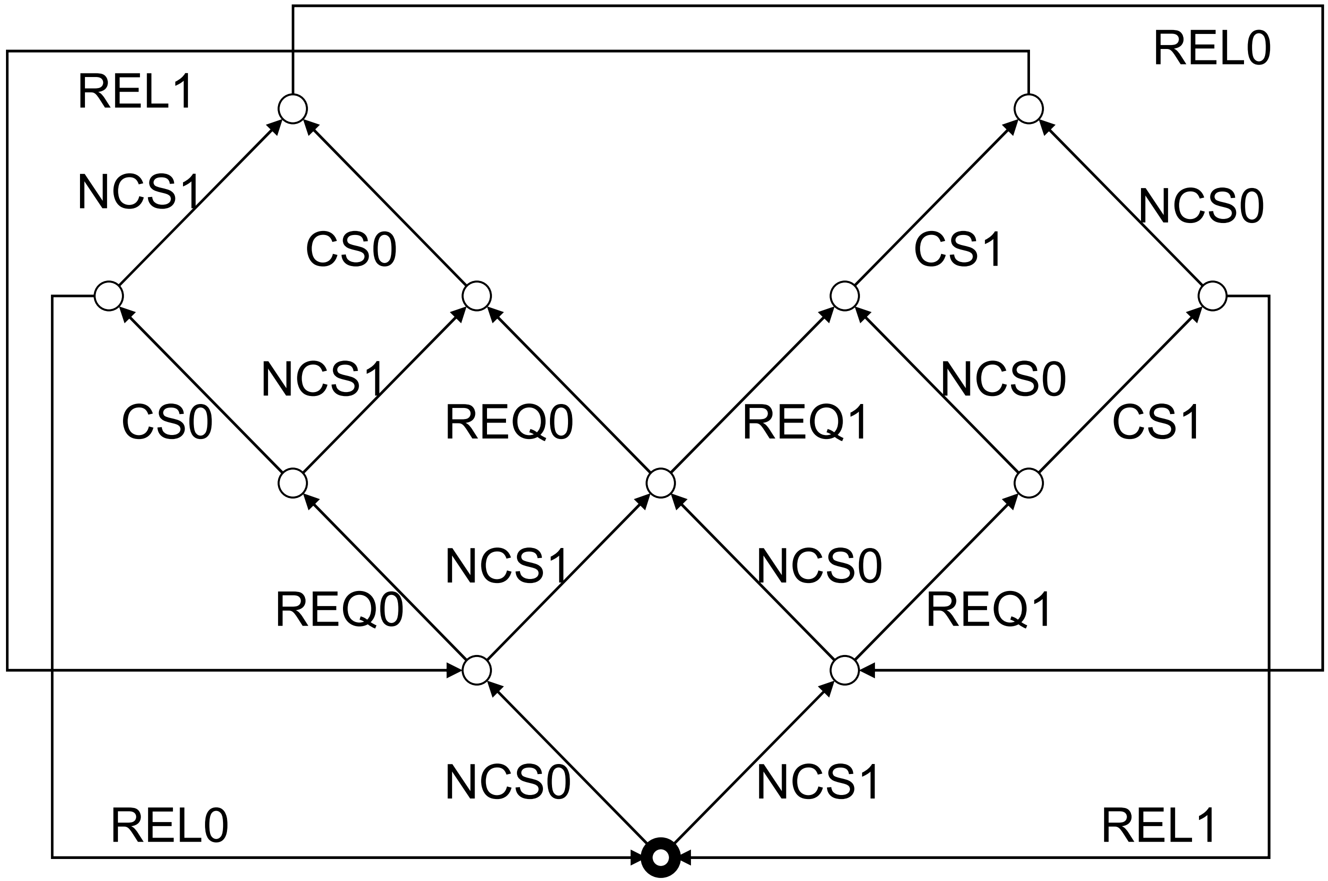
Example

Infinite repetition: $\nu X \,.\, \langle \mathrm{NCS}_1 \vee \mathrm{REQ}_1 \vee \mathrm{CS}_1 \vee \mathrm{REL}_1 \rangle X$



Exercise

Evaluate the formula: $\mu X \,.\, \langle\, CS_0 \,\rangle\, tt \vee ([\, NCS_0 \,]\, ff \wedge \langle\, tt \,\rangle\, X\,)$



Some identities

- Description of (some) ACTL operators:
  - $E\,[\,\varphi_{1\alpha 1}\,U_{\alpha 2}\,\varphi_2\,] = \mu X\,.\,\varphi_1 \wedge (\langle\,\alpha_2\,\rangle\,\varphi_2 \vee \langle\,\alpha_1\,\rangle\,X)$
  - $A\,[\,\varphi_{1\alpha 1}\,U_{\alpha 2}\,\varphi_2\,] = \mu X\,.\,\varphi_1 \wedge \langle\,\mathrm{tt}\,\rangle\,\mathrm{tt} \wedge [\neg(\alpha_1 \vee \alpha_2)\,]\,\mathrm{ff}$
    $\wedge\,[\,\neg\alpha_1 \wedge \alpha_2\,]\,\varphi_2 \wedge [\,\neg\alpha_2\,]\,X \wedge [\,\alpha_1 \wedge \alpha_2\,]\,(\varphi_2 \vee X)$
  - $EF_\alpha\,\varphi = \mu X\,.\,\varphi \vee \langle\,\alpha\,\rangle\,X$
  - $AF_\alpha\,\varphi = \mu X\,.\,\varphi \vee (\langle\,\mathrm{tt}\,\rangle\,\mathrm{tt} \wedge [\,\neg\alpha\,]\,\mathrm{ff} \wedge [\,\alpha\,]\,X)$
- Description of the PDL operators:
  - $\langle\,\beta^*\,\rangle\,\varphi = \mu X\,.\,\varphi \vee \langle\,\beta\,\rangle\,X$
  - $[\,\beta^*\,]\,\varphi = \nu X\,.\,\varphi \wedge [\,\beta\,]\,X$

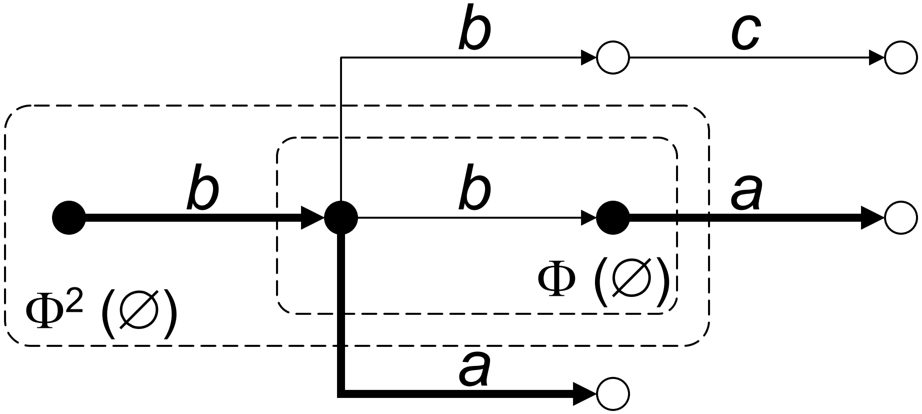
Inevitable reachability

- Inevitable reachability of an action *a*:

  $\text{access}(a) = AF_{tt} \langle a \rangle tt =$

  $\mu X . \langle a \rangle tt \vee (\langle tt \rangle tt \wedge [\, tt \,] X)$
- Associated functional:

  $\Phi(U) = [\![\langle a \rangle tt \vee (\langle tt \rangle tt \wedge [\, tt \,] X)]\!] \; [\, U / X \,]$
- Evaluation on an LTS:



Inevitable execution

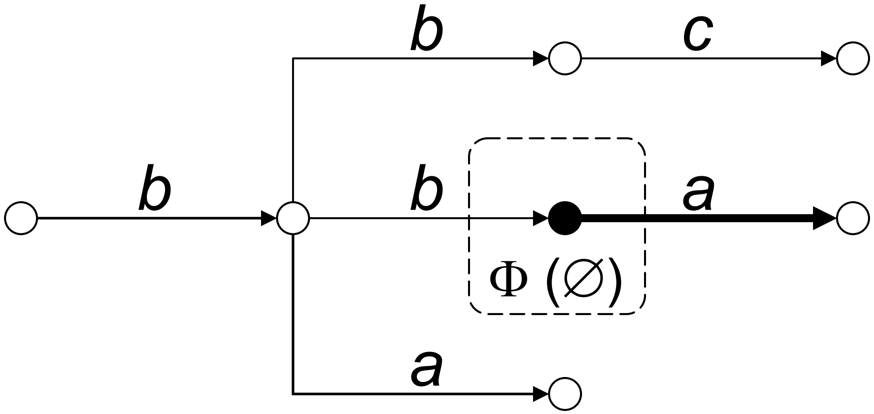
- Inevitable execution of an action *a*:

$$\text{inev}\ (a) = \mu X\ .\ \langle\ tt\ \rangle\ tt \wedge [\ \neg a\ ]\ X$$

- Associated functional:

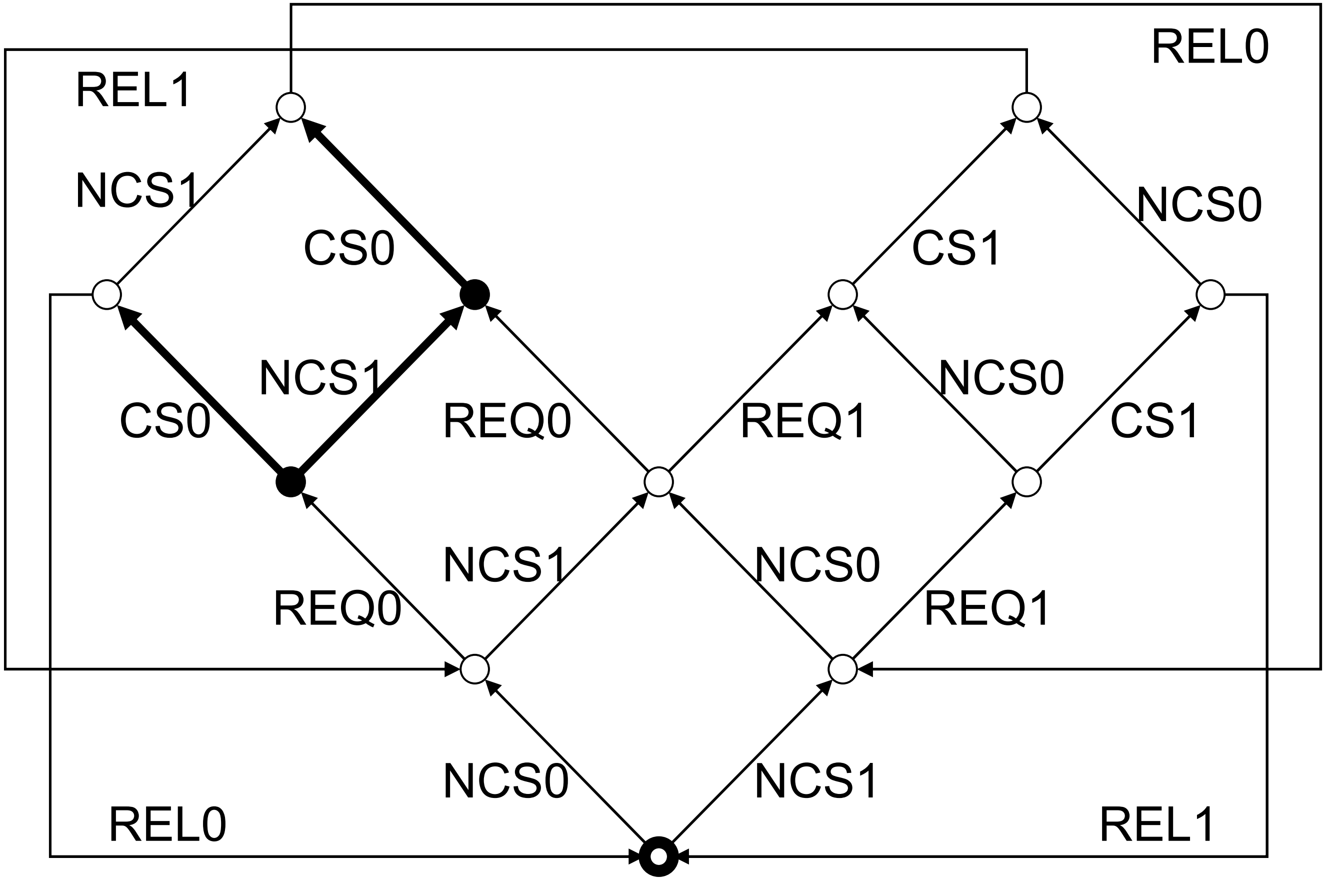
$$\Phi\ (U) = [[\ \langle\ tt\ \rangle\ tt \wedge [\ \neg a\ ]\ X\ ]]\ \ [\ U\ /\ X\ ]$$

- Evaluation on an LTS:



Example

Inevitable execution: $\mu X \,.\, \langle\, tt \,\rangle\, tt \wedge [\, \neg CS_0 \,]\, X$



Fair execution

- Fair execution of an action *a*:

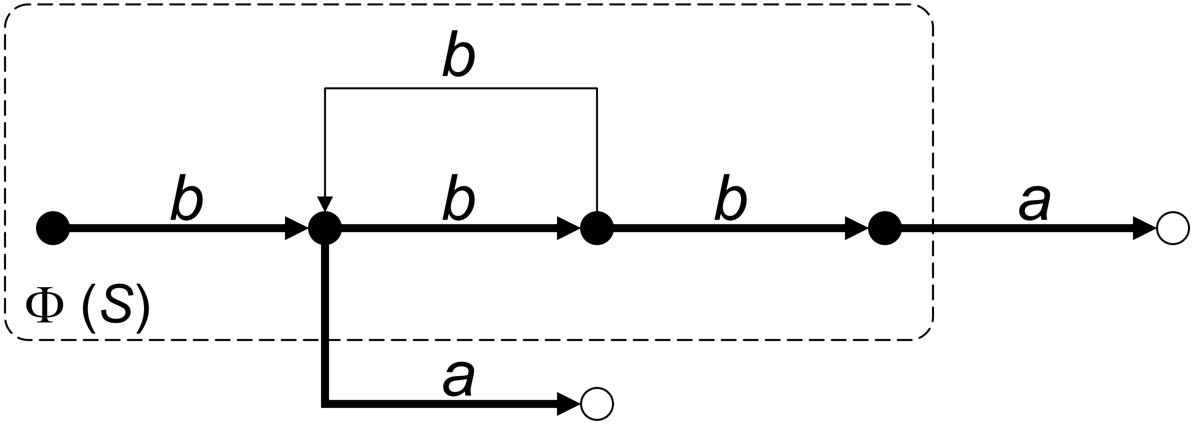
$$\text{fair}(a) = [\,(\neg a)^*\,]\,\langle\, \text{tt}^*.\ a \,\rangle\, \text{tt}$$

$$= \nu X\,.\,\langle\, \text{tt}^*.\ a \,\rangle\, \text{tt} \wedge [\,\neg a\,]\,X$$

- Associated functional:

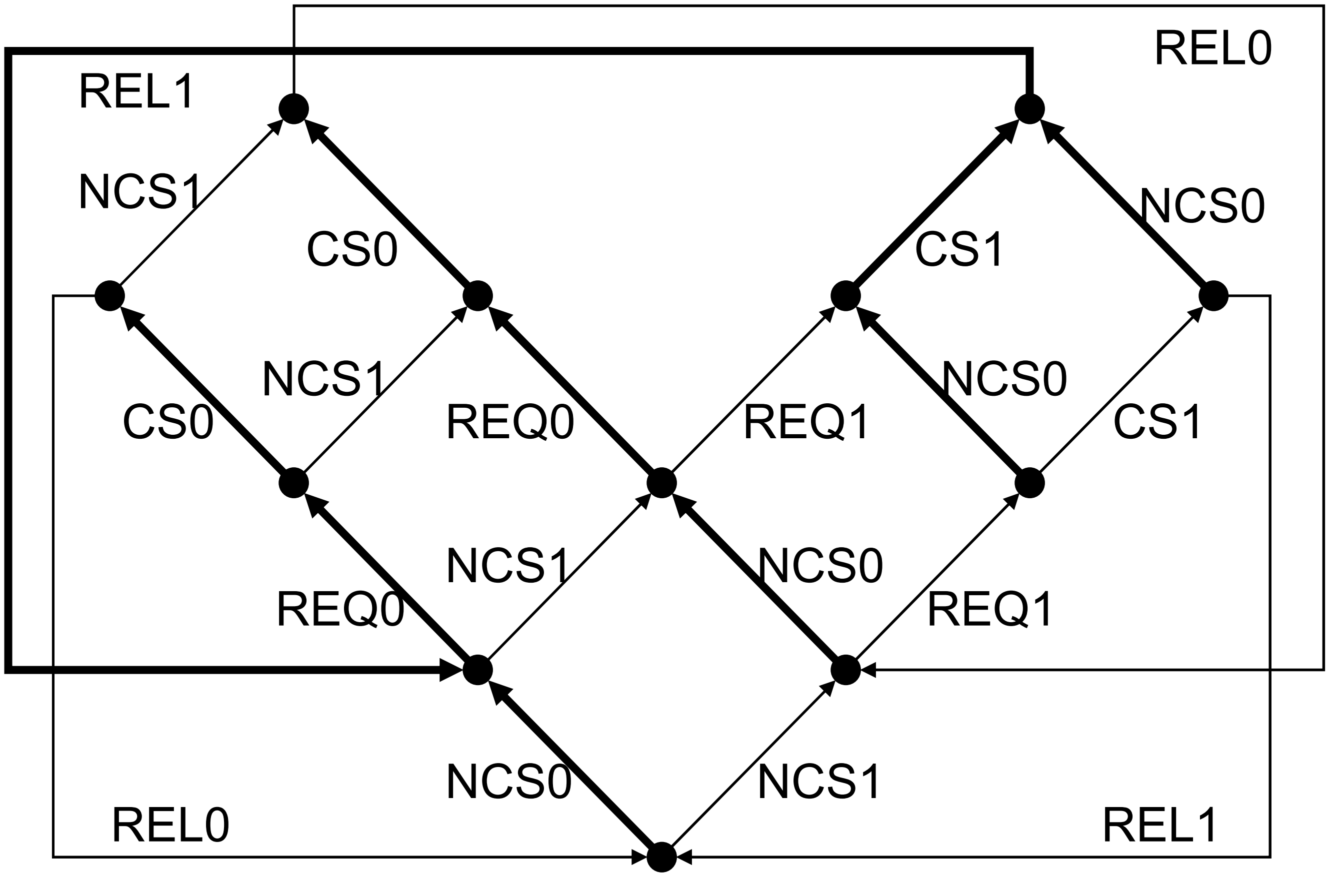
$$\Phi\,(U) = [[\,\langle\, \text{tt}^*.\ a \,\rangle\, \text{tt} \wedge [\,\neg a\,]\,X\,]]\ \ [\,U\,/\,X\,]$$

- Evaluation on an LTS:



Example

Fair execution: $[\ (\neg CS_0)^*\ ]\ \langle\ tt^*.\ CS_0\ \rangle\ tt$



Fixed point logics

(summary)

- They allow to encode virtually all TL proposed in the literature
- Expressive power obtained by *nesting* the fixed point operators:

$$\langle\, (a \,.\, b^*)^* \,.\, c \,\rangle\, \mathrm{tt} =$$

$$\mu X \,.\, \langle\, c \,\rangle\, \mathrm{tt} \vee \langle\, a \,\rangle\, \mu Y \,.\, (X \vee \langle\, b \,\rangle\, Y\,)$$

- Alternation depth of a formula: degree of mutual recursion between μ and ν fixed points

Example of alternation depth 2 formula:

$$\nu X \,.\, \langle\, a^* . \, b \,\rangle\, X \;=\; \nu X \,.\, \mu Y \,.\, \langle\, b \,\rangle\, X \vee \langle\, a \,\rangle\, Y$$

Some verification tools

(for action-based logics)

- CWB (Edinburgh)

  and

- Concurrency Factory (State University of New York)
  - Modal μ-calculus (fixed point operators)

- JACK (University of Pisa, Italy)
  - μ-ACTL (modal μ-calculus combined with ACTL)

- CADP / Evaluator 3.x (INRIA Rhône-Alpes / VASY)
  - Regular alternation-free μ-calculus (PDL modalities and fixed point operators)

Extensions of μ-calculus with data

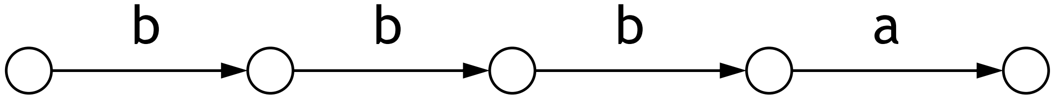
- Temporal logics (ACTL, PDL, ...) and μ-calculi
  - No data manipulation (basic LOTOS, pure CCS, ...)
  - Too low-level operators (complex formulas)

➔ *Extended temporal logics are needed in practice*

- Several μ-calculus extensions with data:
  - For polyadic pi-calculus [Dam-94]
  - For symbolic transition systems [Rathke-Hennessy-96]
  - For μCRL [Groote-Mateescu-99]
  - For full LOTOS [Mateescu-Thivolle-08]

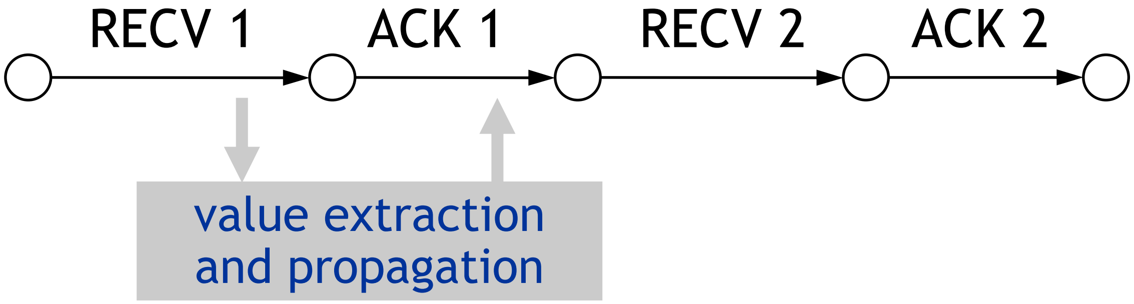
Why to handle data?

- Some properties are cumbersome to express without data (e.g., action counting):



$\langle\, b \,\rangle \langle\, b \,\rangle \langle\, b \,\rangle \langle\, a \,\rangle\ \mathrm{tt}$ or $\langle\, b\,\{3\}\,.\,a \,\rangle\ \mathrm{tt}$?

- LTSs produced from value-passing process algebraic languages (full CCS, LOTOS, ...) contain values on transition labels



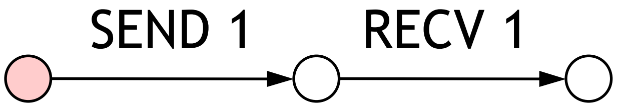
value extraction and propagation

Model Checking Language

- Based on EVALUATOR 3.5 input language
  - standard μ-calculus
  - regular operators
- Data-handling mechanisms
  - data extraction from LTS labels
  - regular operators with counters
  - variable declaration
  - parameterized fixed point operators
  - expressions
- Constructs inspired from programming languages

Parameterized modalities

- Possibility:

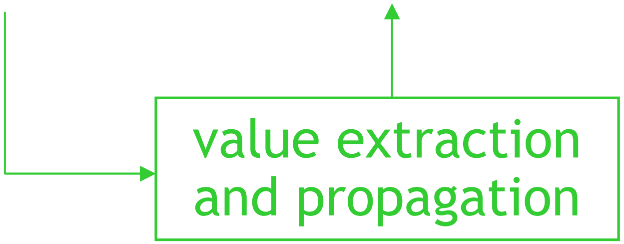


< {SEND ?msg:Nat} > < {RECV !msg} > true


value extraction and propagation

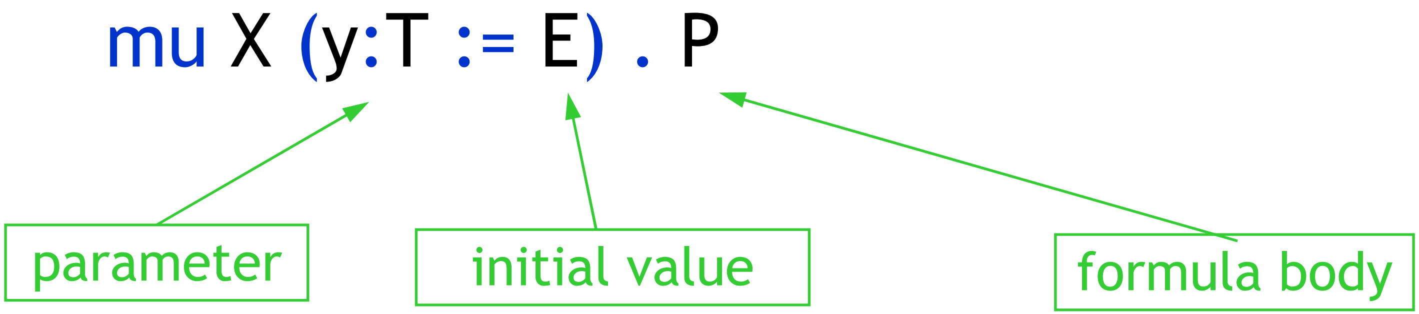
- Necessity:

[{RECV ?msg:Nat}] (msg < 6)


value extraction and propagation

Parameterized fixed points

- (basic) syntax:



mu X (y:T := E) . P

parameter

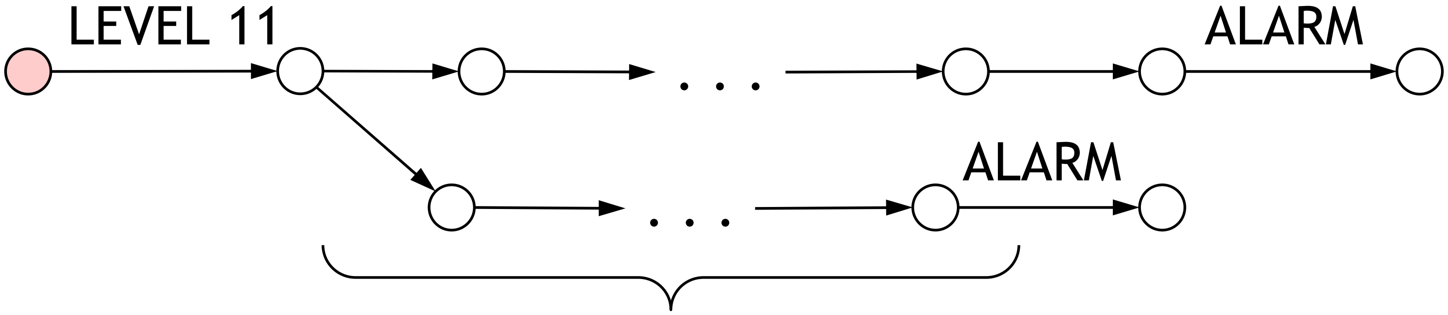
initial value

formula body

  - P contains « calls » X (E’)
  - Allows to perform computations and store intermediate results while exploring the PLTS

Example

- Counting of actions (e.g., clock ticks):



max. 15 transitions
before the alarm

[{LEVEL ?l:Nat where l > 10}]

nu X (c:Nat := 15) .

[not ALARM] (c > 0 and X (c - 1))

Quantifiers

- Existential quantifier:

  exists x:T among { E_1 … E_2 } . P

  limits of the subdomain of T

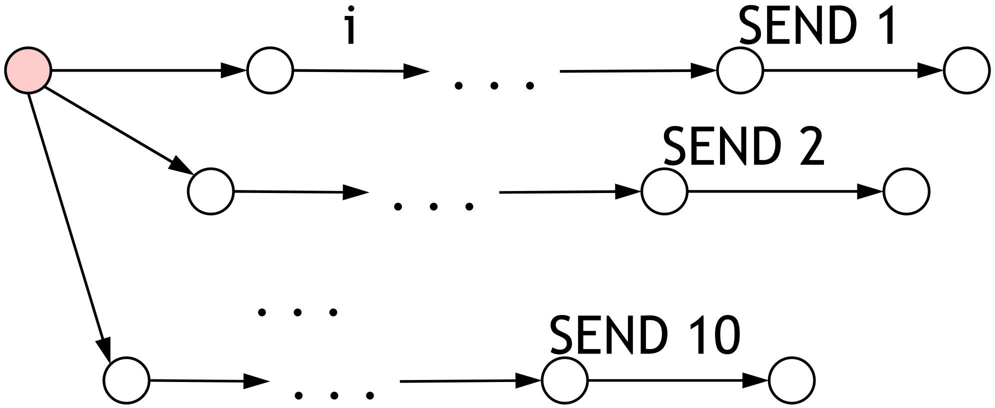
- Universal quantifier:

  forall x:T among { E_1 … E_2 } . P

➔ *shorthands for large disjunctions and conjunctions*

Example

- Broadcast of messages:



forall msg:Nat among { 1 … 10 } .
mu X . (< {SEND !msg} > true or < true > X)

Counting operators

(regular formulas)

R { E }	*repetition E times*
R { E_1 ... }	*repetition at least E_1 times*
R { E_1 ... E_2 }	*repetition between E_1 and E_2 times*

- Some identities:

nil = false *

R * = R { 0 ... }

R + = R { 1 ... }

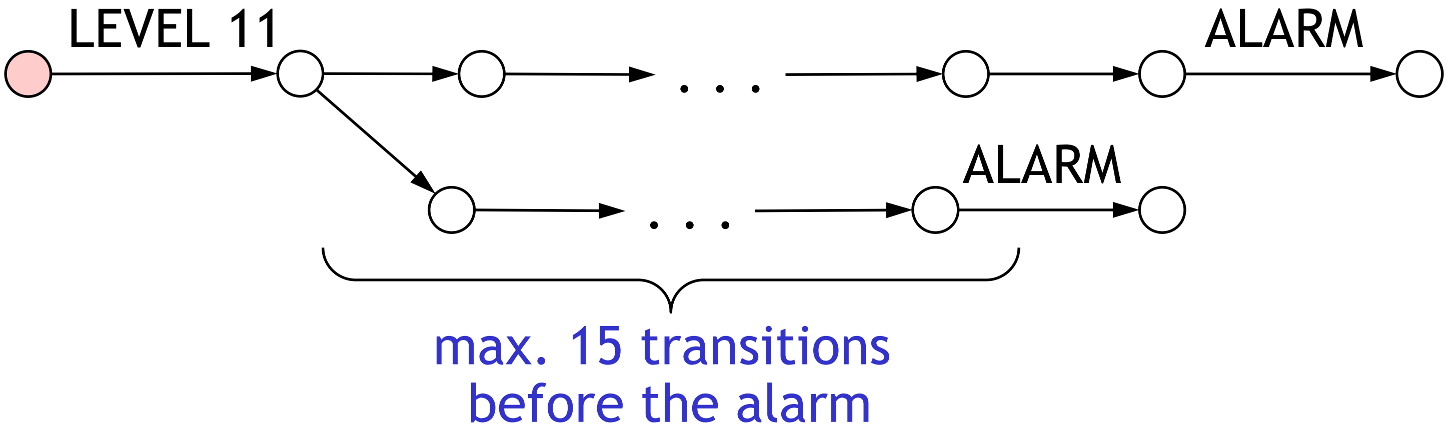
R + = R . R*

R ? = R { 0 ... 1 }

R { E } = R { E ... E }

Example

(action counting revisited)



max. 15 transitions
before the alarm

- Formulation using counting operators:

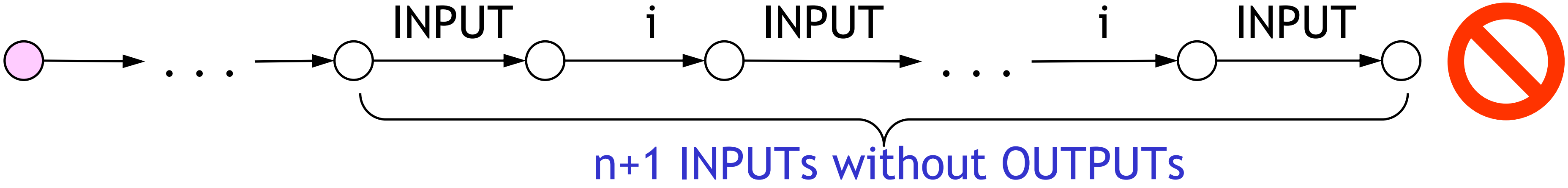
[{LEVEL ?l:Nat where l > 10} . (not ALARM) { 16 }] false

Example

(safety of a n-place buffer)

- Formulation using extended regular operators:

  [true* . ((not OUTPUT)* . INPUT) { n + 1 }] false

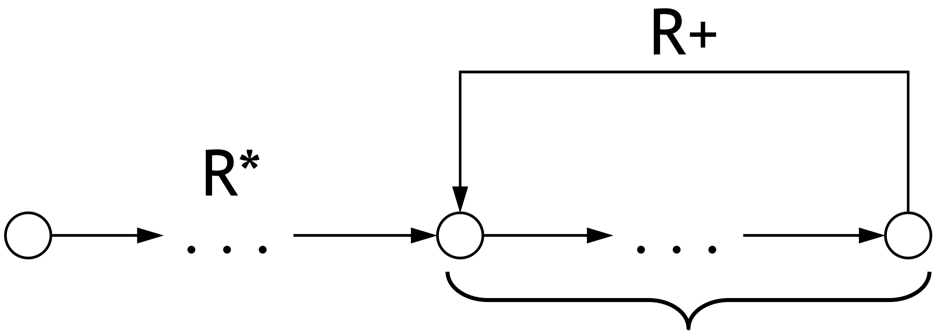


- Formulation using parameterized fixed points:

```
nu X . (nu Y (c:Nat:=0) . (
                  [not OUTPUT] Y (c) and
                  if c = n+1 then [INPUT] false
                        else [INPUT] Y (c+1)
                  end if)
    and [ true ] X)
```

Looping operator (from PDL-delta)

- Δ R operator added to PDL to specify infinite behaviours [Streett-82]
- MCL syntax: < R > @



cycle containing one or more repetitions of R

- Examples:
  - process overtaking

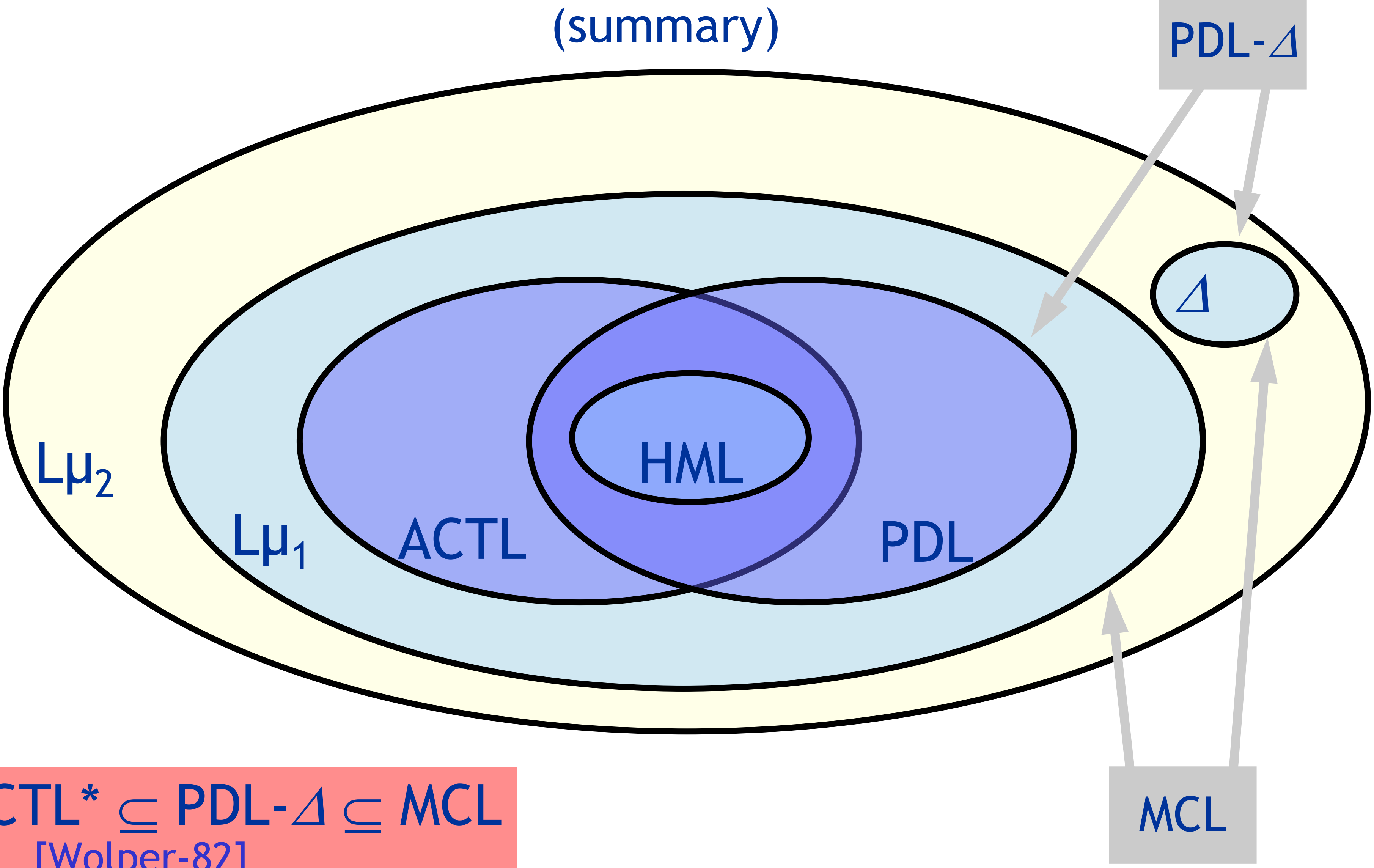
    [REQ_0] < (not GET_0)* . REQ_1 . (not GET_0)* . GET_1 > @
  - Büchi acceptance condition

    < true* . if $P_{accepting}$ then true end if > @

➔ *allows to encode LTL model checking*

Expressiveness

(summary)



CTL* $\subseteq$ PDL-Δ $\subseteq$ MCL

[Wolper-82]

Adequacy with equivalence relations

- A temporal logic L is adequate with an equivalence relation $\approx$ iff for all LTSs M_1 and M_2

$$M_1 \approx M_2 \quad \text{iff} \quad \forall \varphi \in L \,.\, (M_1 \mid= \varphi \;\Leftrightarrow\; M_2 \mid= \varphi)$$

- HML:
  - Adequate with strong bisimulation
  - HMLU (HML with Until): weak bisimulation
- ACTL-X (fragment presented here):
  - Adequate with branching bisimulation
- PDL and modal mu-calculus:
  - Adequate with strong bisimulation
  - Weak mu-calculus: weak bisimulation

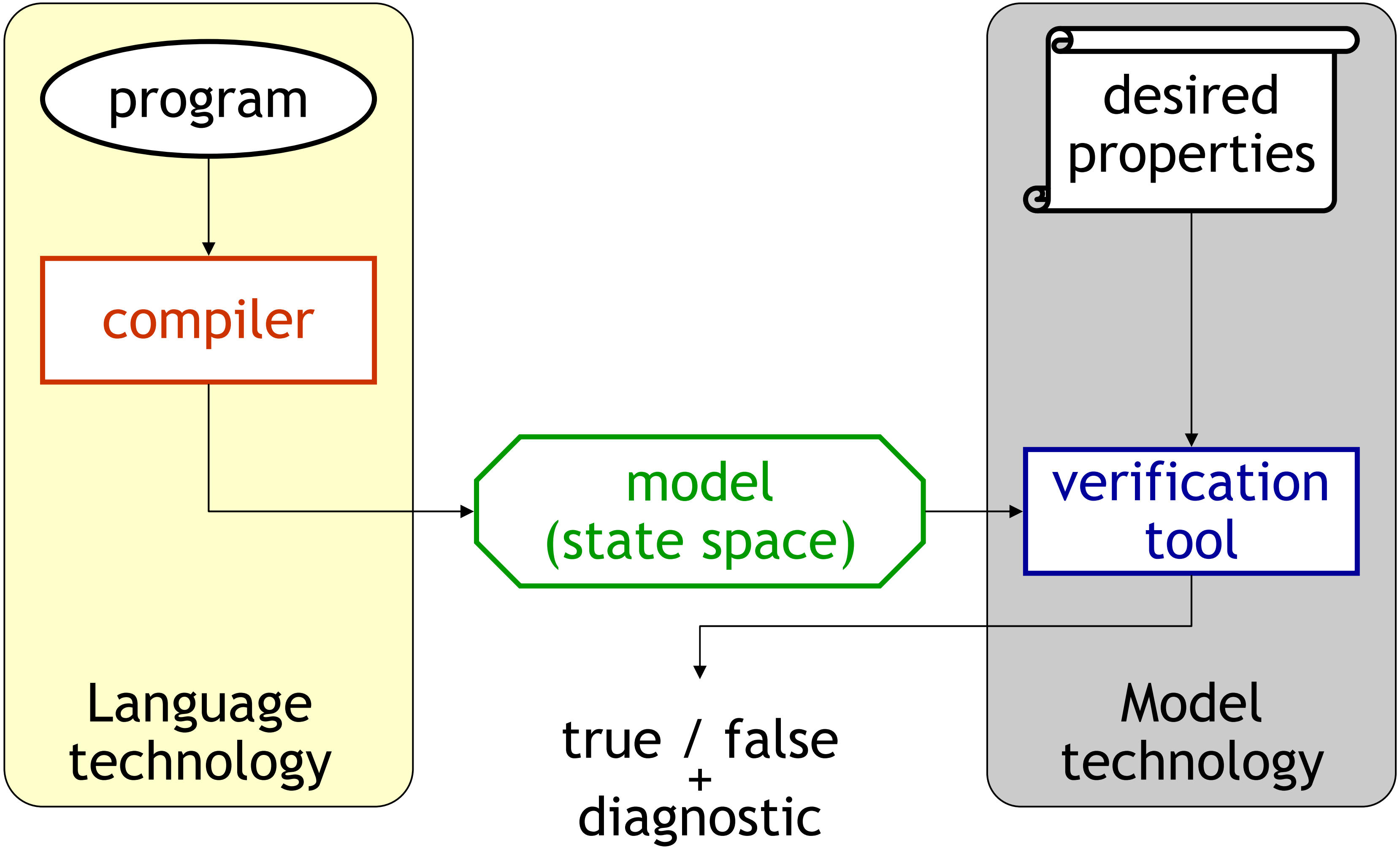
$$\langle\langle\ \rangle\rangle\, \varphi = \langle\, \tau^* \,\rangle\, \varphi$$

$$\langle\langle\, a \,\rangle\rangle\, \varphi = \langle\, \tau^* . \, a \,.\, \tau^* \,\rangle\, \varphi$$

On-the-fly verification

- Principles
- Alternation-free boolean equation systems
- Local resolution algorithms
- Applications:
  - Equivalence checking
  - Model checking
  - Tau-confluence reduction
- Implementation and use

Principle of explicit-state verification



program

compiler

model
(state space)

desired
properties

verification
tool

true / false
+
diagnostic

Language
technology

Model
technology

On-the-fly verification

- Incremental construction of the state space
  - Way of fighting against state explosion
  - Detection of errors in complex systems
- “Traditional” methods:
  - Equivalence checking
  - Model checking
- Solution adopted:
  - Translation of the verification problem into the resolution of a *boolean equation system* (BES)
  - Generation of *diagnostics* (fragments of the state space) explaining the result of verification

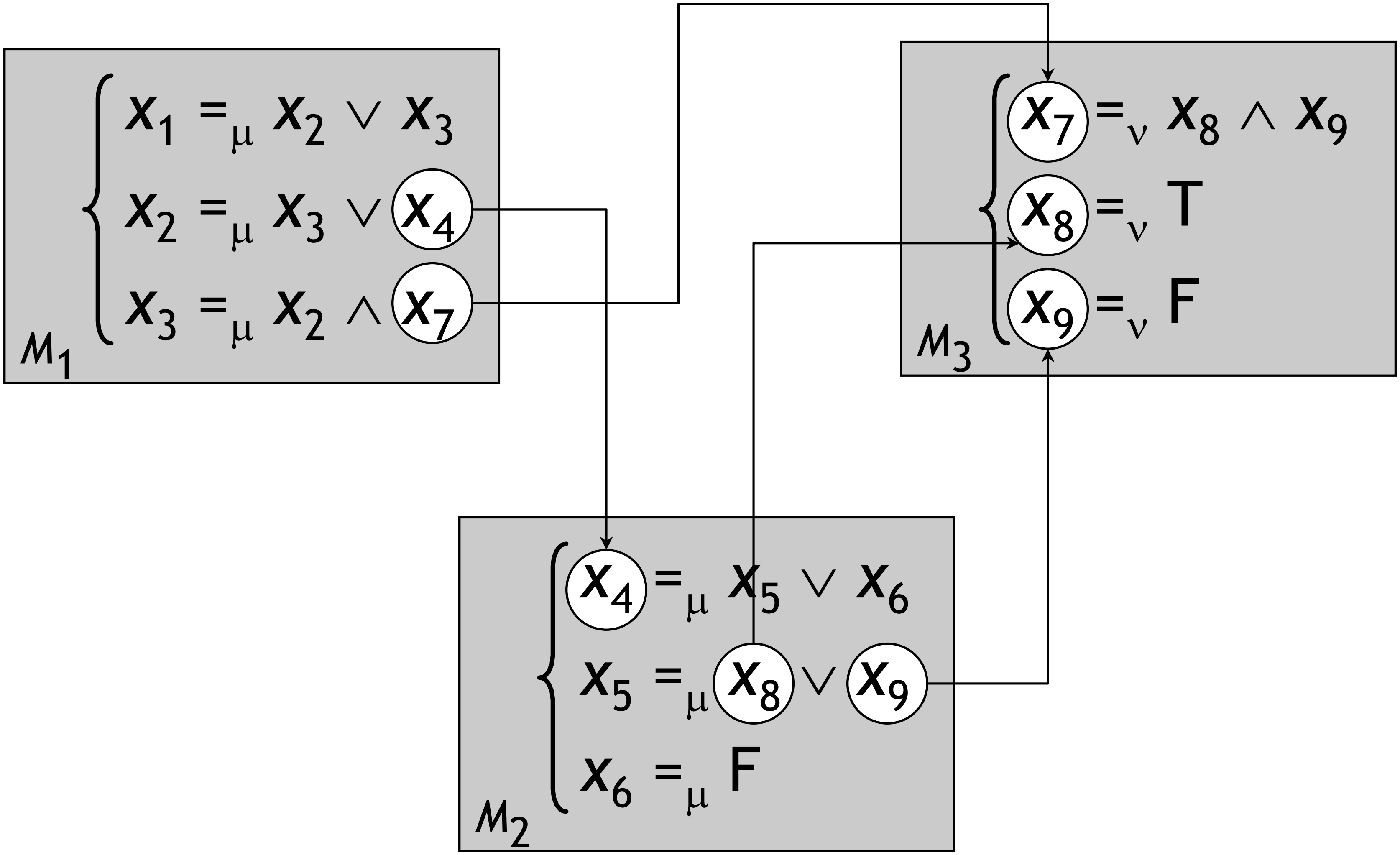
Boolean equation systems

(syntax)

A BES is a tuple $B = (x, M_1, \ldots, M_n)$, where

- $x \in \boldsymbol{X}$: main boolean variable
- $M_i = \{ x_j =\sigma_i \, op_j \, \boldsymbol{X}_j \}_{j \in [1, mi]}$: equation blocks
  - $\sigma_i \in \{ \mu, \nu \}$: fixed point sign of block i
  - $op_j \in \{ \vee, \wedge \}$: operator of equation j
  - $\boldsymbol{X}_j \subseteq \boldsymbol{X}$: variables in the right-hand side of equation j
  - F = $\vee\emptyset$ (empty disjunction), T = $\wedge\emptyset$ (empty conjunction)
  - x_j depends upon x_k iff $x_k \in \boldsymbol{X}_j$
  - M_i depends upon M_l iff a x_j of M_i depends upon a x_k of M_l
  - *Closed* block: does not depend upon other blocks
- *Alternation-free* BES: M_i depends upon $M_{i+1} \ldots M_n$

Example



$$M_1 \quad \begin{cases} x_1 =_\mu x_2 \vee x_3 \\ x_2 =_\mu x_3 \vee \textcircled{x_4} \\ x_3 =_\mu x_2 \wedge \textcircled{x_7} \end{cases}$$

$$M_3 \quad \begin{cases} \textcircled{x_7} =_\nu x_8 \wedge x_9 \\ \textcircled{x_8} =_\nu T \\ \textcircled{x_9} =_\nu F \end{cases}$$

$$M_2 \quad \begin{cases} \textcircled{x_4} =_\mu x_5 \vee x_6 \\ x_5 =_\mu \textcircled{x_8} \vee \textcircled{x_9} \\ x_6 =_\mu F \end{cases}$$

Particular blocks

- *Acyclic* block:
  - No cyclic dependencies between variables of the block
- Var. x_i disjunctive (conjunctive): $op_i = \vee$ ($op_i = \wedge$)
- *Disjunctive* block:
  - contains disjunctive variables
  - and conjunctive variables
    - with a single non constant successor in the block (the last one in the right-hand side of the equation)
    - all other successors are constants or free variables (defined in other blocks)
- *Conjunctive* block: dual definition

Boolean equation systems

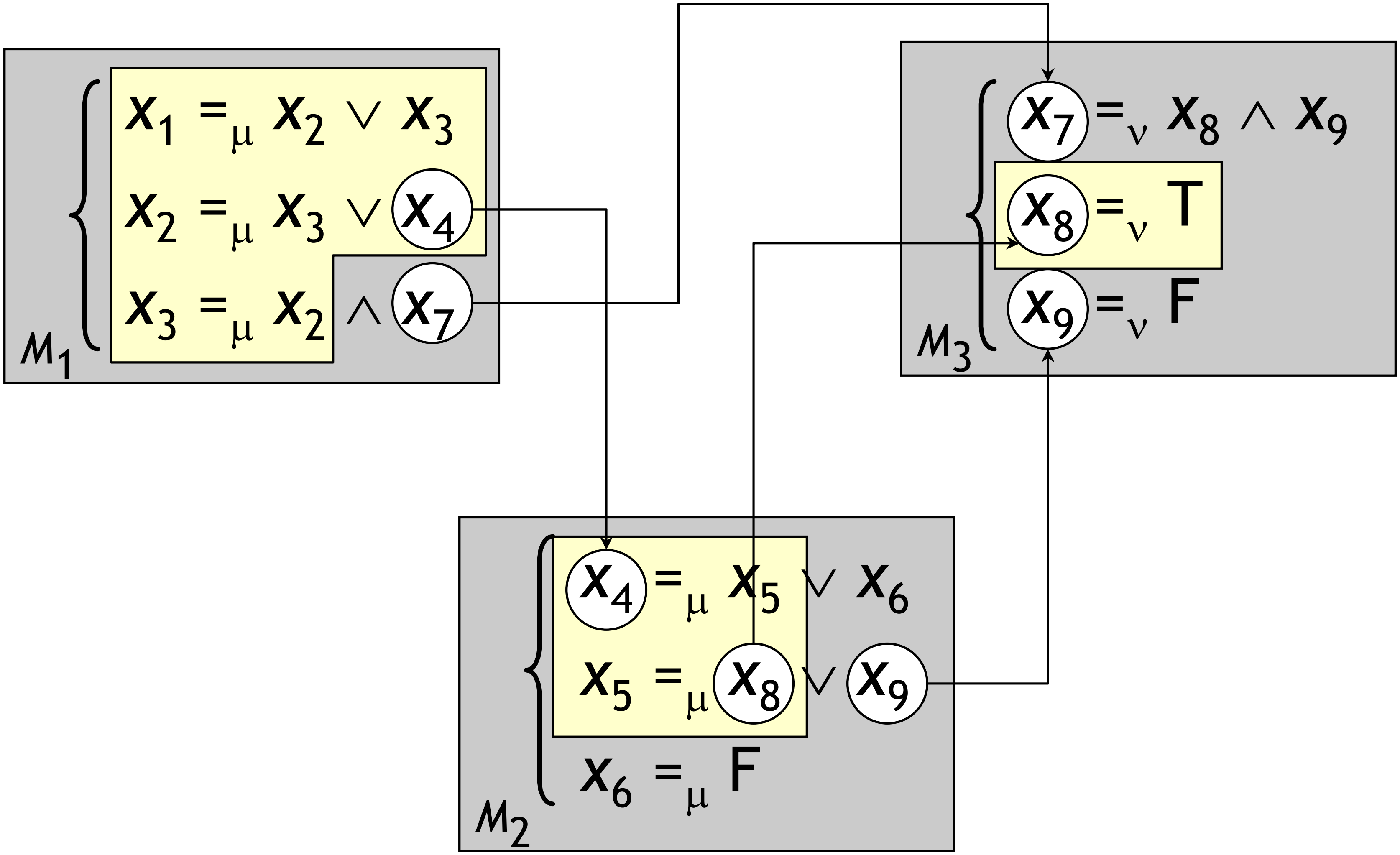
(semantics)

- Context: partial function $\delta : \boldsymbol{X} \rightarrow \text{Bool}$
- Semantics of a boolean formula:
  - $[[\ op\ \{\ x_1, \dots, x_p\ \}\ ]]\ \delta = op\ (\delta\ (x_1), \dots, \delta\ (x_p))$
- Semantics of a block:
  - $[[\ \{\ x_j =\sigma\ op_j\ \boldsymbol{X}_j\ \}_{j\ \in\ [1,\ m]}\ ]]\ \delta = \sigma\Phi_\delta$
  - $\Phi_\delta : \text{Bool}^m \rightarrow \text{Bool}^m$
  - $\Phi_\delta\ (b_1, \dots, b_m) = ([[\ op_j\ \boldsymbol{X}_j\ ]]\ (\delta \oplus [b_1/x_1, \dots, b_m/x_m]))_{j\ \in\ [1,\ m]}$
- Semantics of a BES:
  - $[[\ (x, M_1, \dots, M_n)\ ]] = \delta_1\ (x)$
  - $\delta_n = [[\ M_n\ ]]\ []$ (M_n closed)
  - $\delta_i = ([[\ M_i\ ]]\ \delta_{i+1}) \oplus \delta_{i+1}$ (M_i depends upon $M_{i+1} \dots M_n$)

Local resolution

- Alternation-free BES $B = (x, M_1, \ldots, M_n)$
- Primitive: compute a variable of a block
  - A resolution routine R_i associated to M_i
  - $R_i\ (x_j)$ computes the value of x_j in M_i
  - Evaluation of the rhs of equations + substitution
  - Call stack $R_1\ (x) \rightarrow \ldots \rightarrow R_n\ (x_k)$ bounded by the depth of the dependency graph between blocks
  - “Coroutine-like” style: each R_i must keep its context
- Advantages:
  - Simple resolution routines (a single type of fixed point)
  - Easy to optimize for particular kinds of blocks

Example



M_1
$$\begin{cases} x_1 =_\mu x_2 \vee x_3 \\ x_2 =_\mu x_3 \vee x_4 \\ x_3 =_\mu x_2 \wedge x_7 \end{cases}$$

M_2
$$\begin{cases} x_4 =_\mu x_5 \vee x_6 \\ x_5 =_\mu x_8 \vee x_9 \\ x_6 =_\mu F \end{cases}$$

M_3
$$\begin{cases} x_7 =_\nu x_8 \wedge x_9 \\ x_8 =_\nu T \\ x_9 =_\nu F \end{cases}$$

Local resolution algorithms

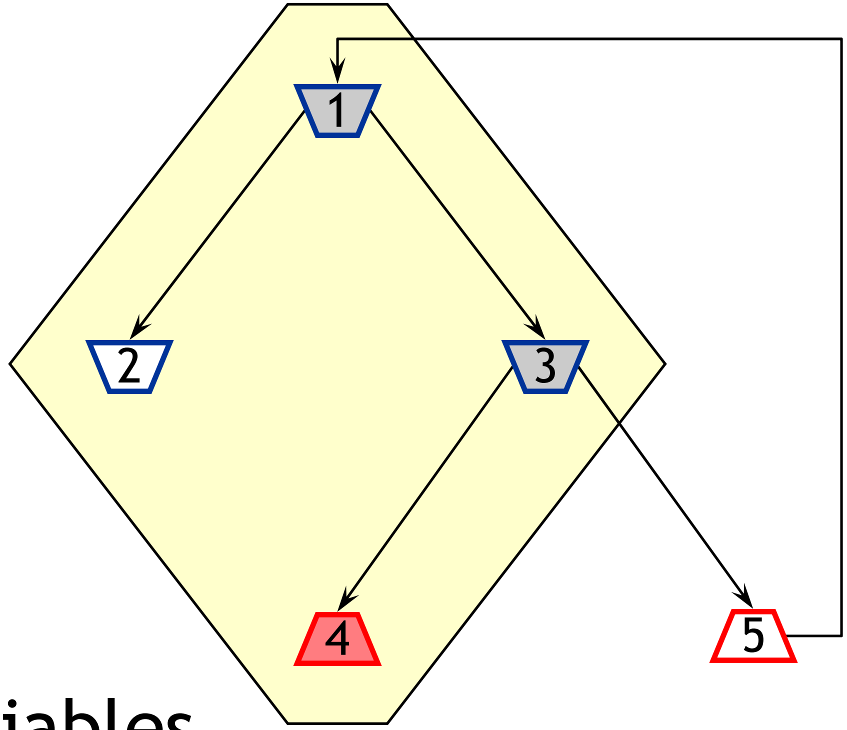
- Representation of blocks as *boolean graphs* [Andersen-94]
- To a block $M = \{ x_j =_\mu op_j\ \boldsymbol{X}_j \}_{j \text{ in } [1, m]}$ we associate the boolean graph $G = (V, E, L, \mu)$, where:
  - $V = \{ x_1, \ldots, x_m \}$: set of vertices (variables)
  - $E = \{ (x_i, x_j) \mid x_j \in \boldsymbol{X}_i \}$: set of edges (dependencies)
  - $L : V \rightarrow \{ \vee, \wedge \}$, $L(x_j) = op_j$: vertex labeling
- Principle of the algorithms:
  - *Forward* exploration of G starting at $x \in V$
  - *Backward* propagation of stable (computed) variables
  - Termination: x is stable or G is completely explored

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

[Mateescu-06]

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*

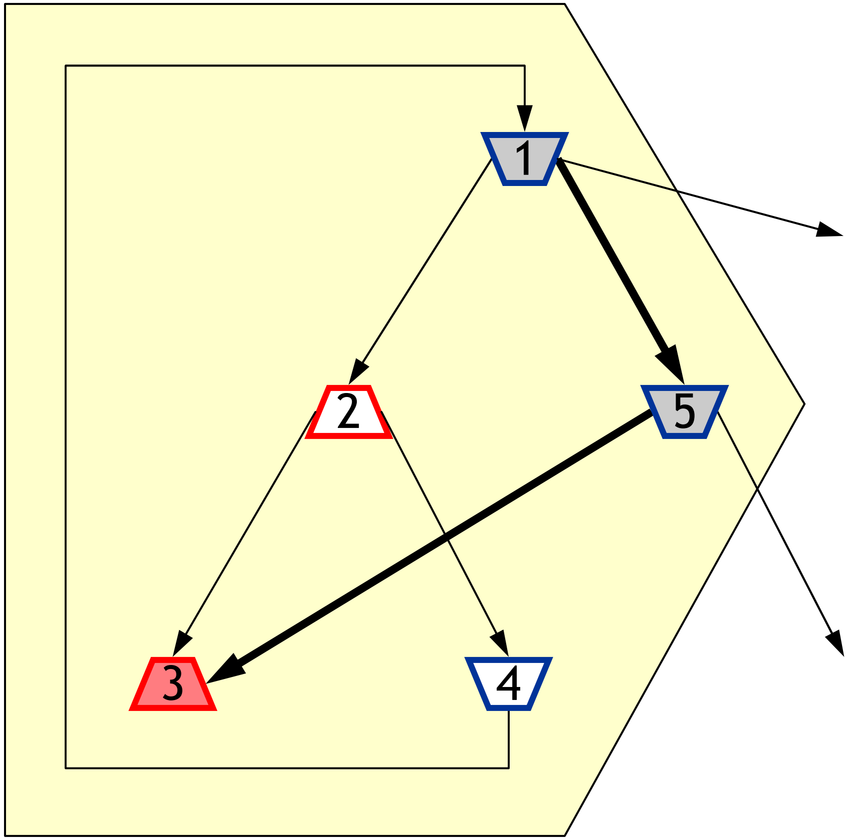
B. While executing *R* (*x*), every variable explored must be « linked » to *x* via unstable variables
   ➔ *graph exploration limited to "useful" variables*

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

Algorithm A0

(general)

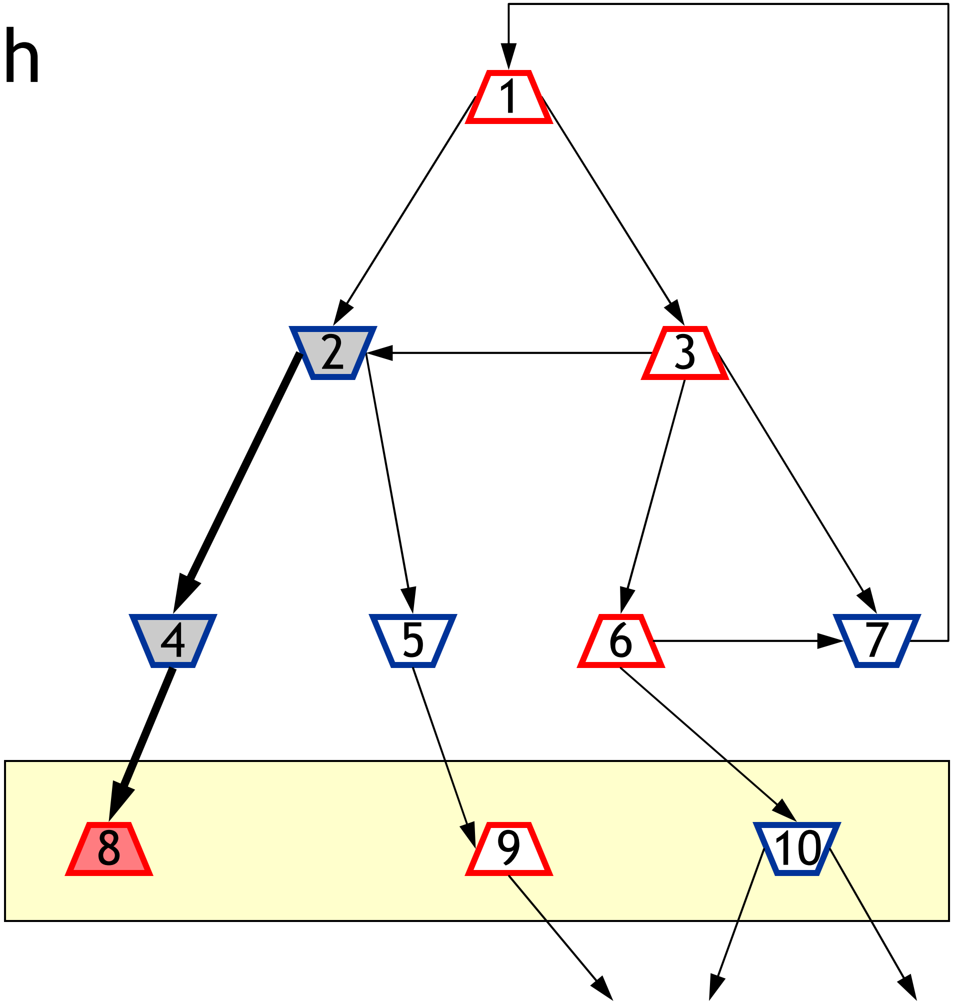
- DFS of the boolean graph
- Satisfies A, B, C
- Memory complexity $O(|V|+|E|)$
- Optimized version of [Andersen-94]
- Developed for model checking regular alternation-free μ-calculus [Mateescu-Sighireanu-00,03]



Algorithm A1

(general)

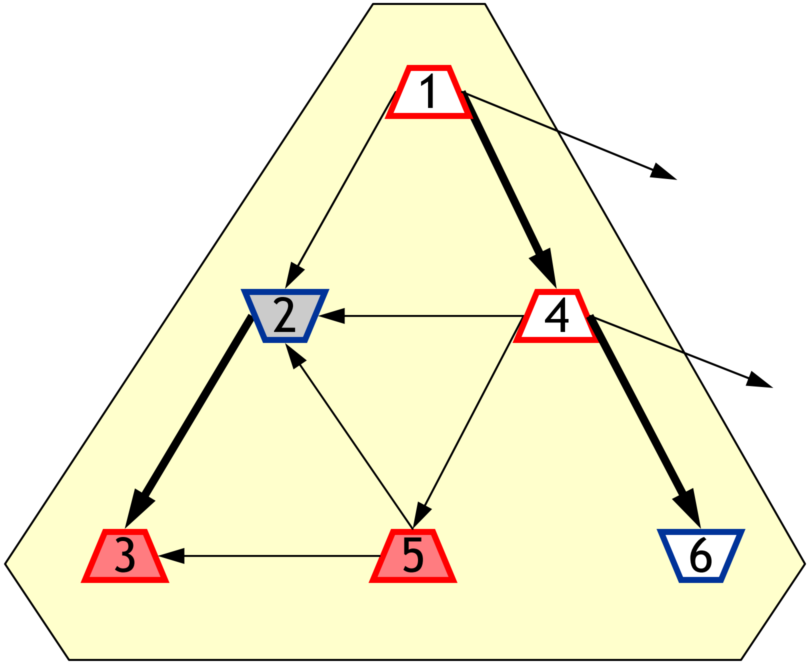
- BFS of the boolean graph
- Satisfies A, C
  (risk of computing
  useless variables)
- Slightly slower than A0
- Memory complexity
  $O(|V|+|E|)$
- Low-depth diagnostics



Algorithm A2

(acyclic)

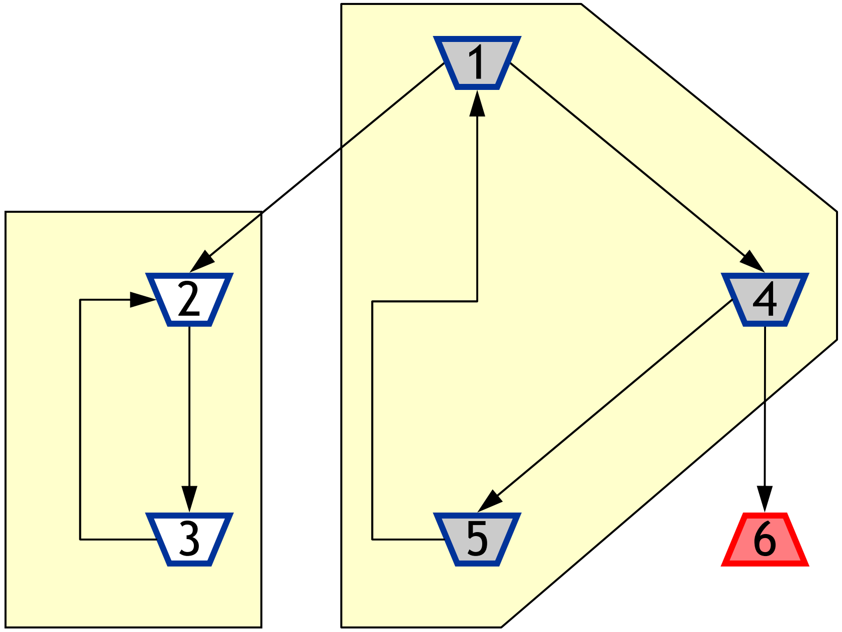
- DFS of the boolean graph
- Back-propagation of stable variables on the DFS stack only
- Satisfies A, B, C
- Avoids storing edges
- Memory complexity *O* (|*V*|)
- Developed for trace-based verification [Mateescu-02]



Algorithm A3 / 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 CTL, ACTL, and PDL



SCC of false variables

SCC of true variables

Resolution algorithms

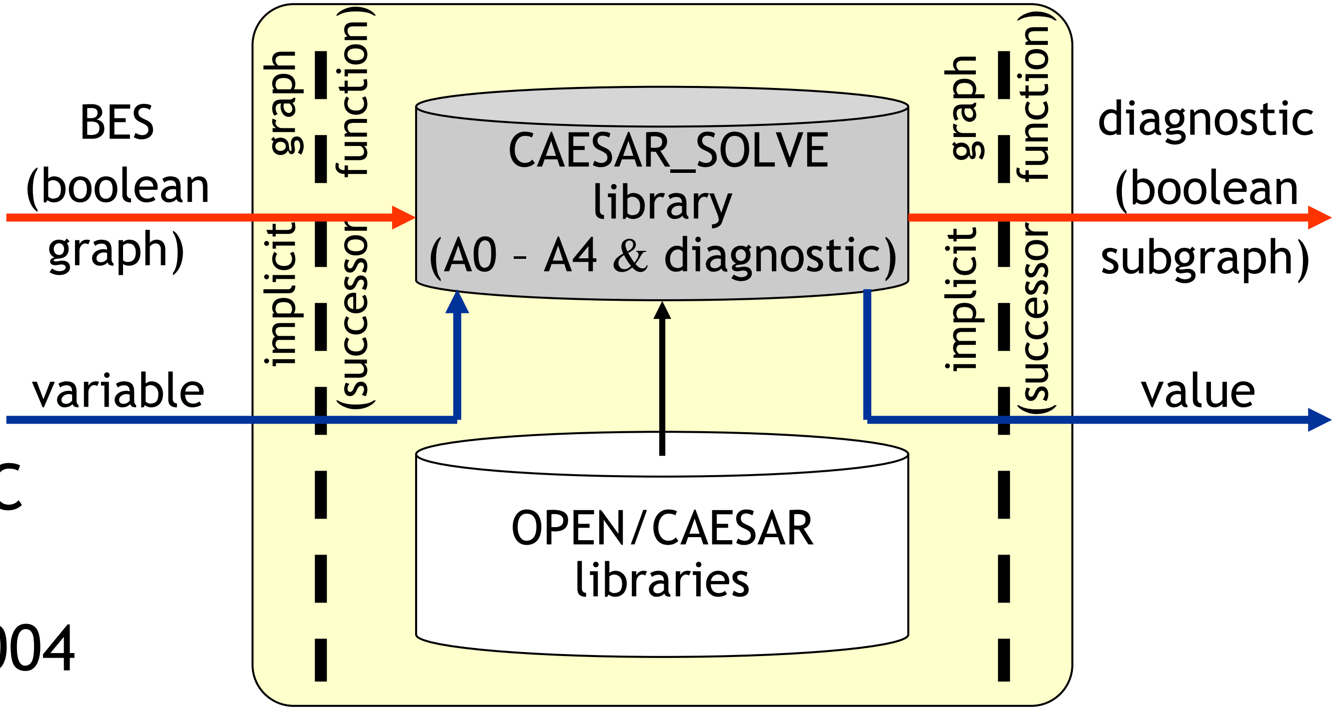
(summary)

- A0 (DFS, general)
  - Satisfies A, B, C
  - Memory complexity $O(|V|+|E|)$
- A1 (BFS, general)
  - Satisfies A, C + « small » diagnostics
  - Memory complexity $O(|V|+|E|)$
- A2 (DFS, acyclic)
  - Satisfies A, B, C
  - Memory complexity $O(|V|)$
- A3/A4 (DFS, disjunctive/conjunctive)
  - Satisfies A, B, C
  - Memory complexity $O(|V|)$

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

Caesar_Solve library of CADP

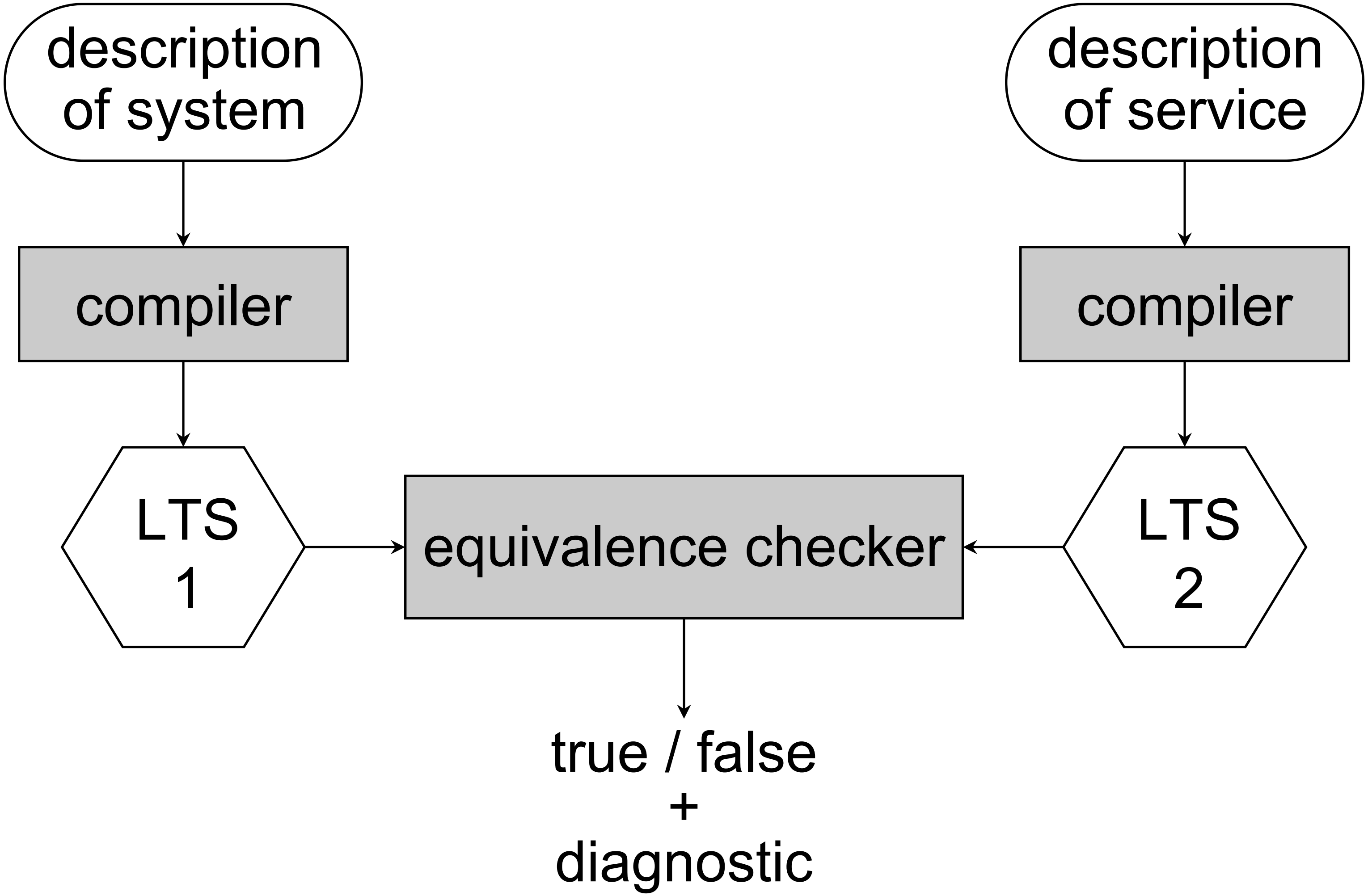
[Mateescu-03,06]



- 15 000 lines of C
- Integrated into CADP in Dec. 2004
- Diagnostic generation features [Mateescu-00]
- Used as verification back-end for Bisimulator, Evaluator 3.5 and 4.0, Reductor 5.0

Equivalence checking

(principle)


description of system

compiler

LTS 1

description of service

compiler

LTS 2

equivalence checker

true / false + diagnostic

Strong equivalence

- $M_1 = (Q_1, A, T_1, q_{01}), M_2 = (Q_2, A, T_2, q_{02})$
  $\approx \subseteq Q_1 \times Q_2$ is the maximal relation s.t. $p \approx q$ iff

  $\forall a \in A. \forall p \rightarrow_a p' \in T_1. \exists q \rightarrow_a q' \in T_2. p' \approx q'$
  and
  $\forall a \in A. \forall q \rightarrow_a q' \in T_2. \exists p \rightarrow_a p' \in T_1. p' \approx q'$

- $M_1 \approx M_2$ iff $q_{01} \approx q_{02}$

Translation to a BES

- Principle: $p \approx q$ iff $X_{p,q}$ is true
- General BES:

$$\left\{\begin{array}{l} \boxed{X_{p,q} =_\nu (\bigwedge_{p \rightarrow_a p'} \bigvee_{q \rightarrow_a q'} X_{p',q'})} \\ \qquad\qquad \wedge \\ \qquad\qquad (\bigwedge_{q \rightarrow_a q'} \bigvee_{p \rightarrow_a p'} X_{p',q'}) \end{array}\right.$$

- Simple BES:

$$\left\{\begin{array}{l} \boxed{\begin{array}{l} X_{p,q} =_\nu (\bigwedge_{p \rightarrow_a p'} Y_{a,p',q}) \\ Y_{a,p',q} =_\nu \bigvee_{q \rightarrow_a q'} X_{p',q'} \end{array}} \wedge (\bigwedge_{q \rightarrow_a q'} Z_{a,p,q'}) \\ Z_{a,p,q'} =_\nu \bigvee_{p \rightarrow_a p'} X_{p',q'} \end{array}\right.$$

$p \leq q$
(preorder)

Tau*.a and safety equivalences

- $M_1 = (Q_1, A_\tau, T_1, q_{01}), M_2 = (Q_2, A_\tau, T_2, q_{02})$
  $A_\tau = A \cup \{ \tau \}$
- Tau*.a equivalence:

$$\begin{cases} X_{p,q} =_\nu (\bigwedge_{p \to_{\tau^*.a} p'} \bigvee_{q \to_{\tau^*.a} q'} X_{p',q'}) \\ \qquad\qquad \wedge \\ \qquad\qquad (\bigwedge_{q \to_{\tau^*.a} q'} \bigvee_{p \to_{\tau^*.a} p'} X_{p',q'}) \end{cases}$$

- Safety equivalence:

$$\begin{cases} X_{p,q} =_\nu Y_{p,q} \wedge Y_{q,p} \\ Y_{p,q} =_\nu \bigwedge_{p \to_{\tau^*.a} p'} \bigvee_{q \to_{\tau^*.a} q'} Y_{p',q'} \end{cases}$$

Observational and branching equivalences

- Observational equivalence:

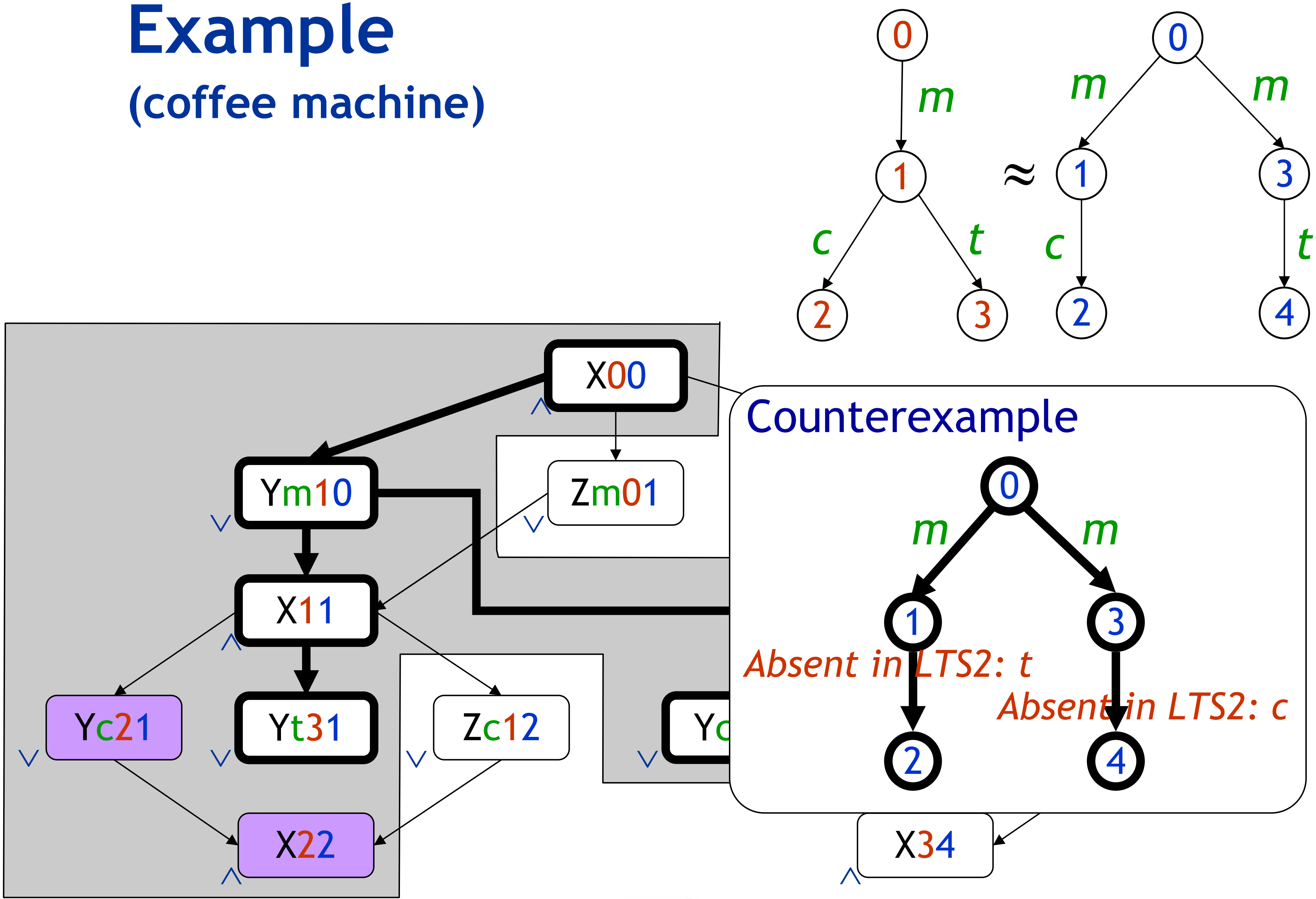
$$\begin{cases} X_{p,q} =_{\nu} (\bigwedge_{p \rightarrow_{\tau} p'} \bigvee_{q \rightarrow_{\tau^*} q'} X_{p',q'}) \wedge (\bigwedge_{p \rightarrow_{a} p'} \bigvee_{q \rightarrow_{\tau^*.a.\tau^*} q'} X_{p',q'}) \\ \qquad\quad \wedge \\ \qquad\quad (\bigwedge_{q \rightarrow_{\tau} q'} \bigvee_{p \rightarrow_{\tau^*} p'} X_{p',q'}) \wedge (\bigwedge_{q \rightarrow_{a} q'} \bigvee_{p \rightarrow_{\tau^*.a.\tau^*} p'} X_{p',q'}) \end{cases}$$

- Branching equivalence:

$$\begin{cases} X_{p,q} =_{\nu} \bigwedge_{p \rightarrow_{b} p'} ((b=\tau \wedge X_{p',q}) \vee \bigvee_{q \rightarrow_{\tau^*} q' \rightarrow_{b} q''} (X_{p,q'} \wedge X_{p',q''}) \\ \qquad\quad \wedge \\ \qquad\quad \bigwedge_{q \rightarrow_{b} q'} ((b=\tau \wedge X_{p,q'}) \vee \bigvee_{p \rightarrow_{\tau^*} p' \rightarrow_{b} p''} (X_{p',q} \wedge X_{p'',q'}) \end{cases}$$

Example

(coffee machine)

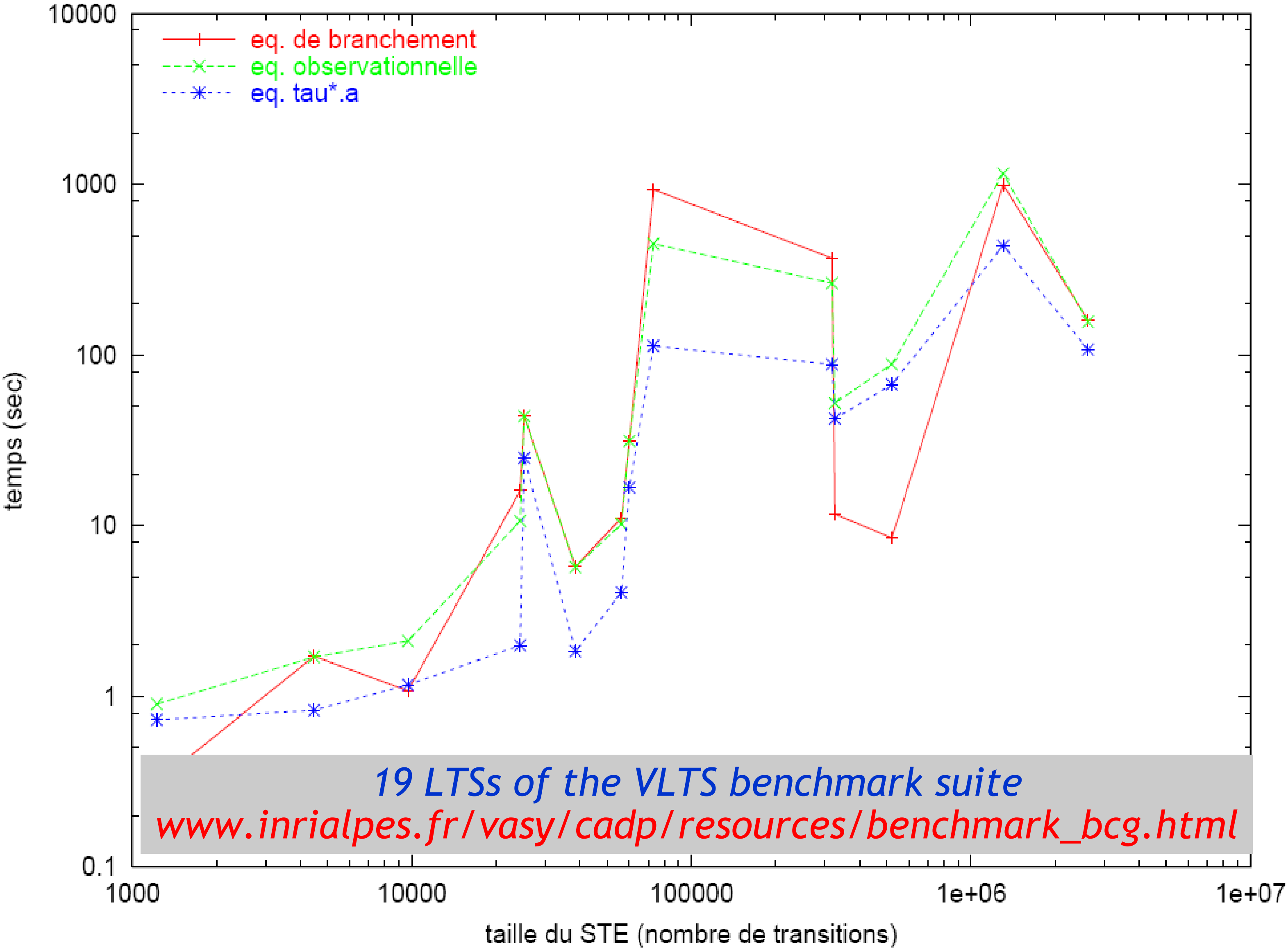


Counterexample

Absent in LTS2: t

Absent in LTS2: c

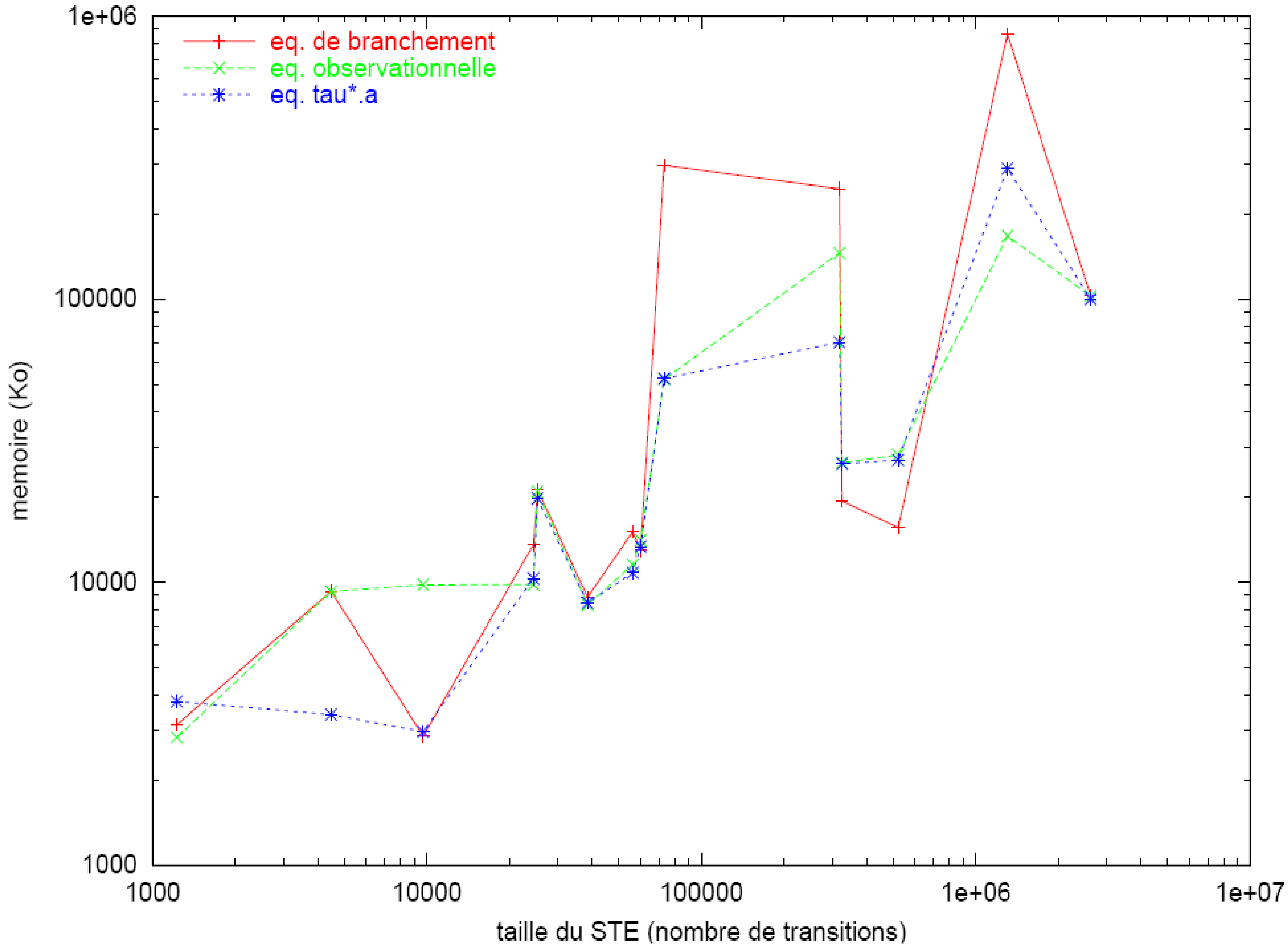
Equivalence checking (time)



19 LTSs of the VLTS benchmark suite

www.inrialpes.fr/vasy/cadp/resources/benchmark_bcg.html

Equivalence checking (memory)



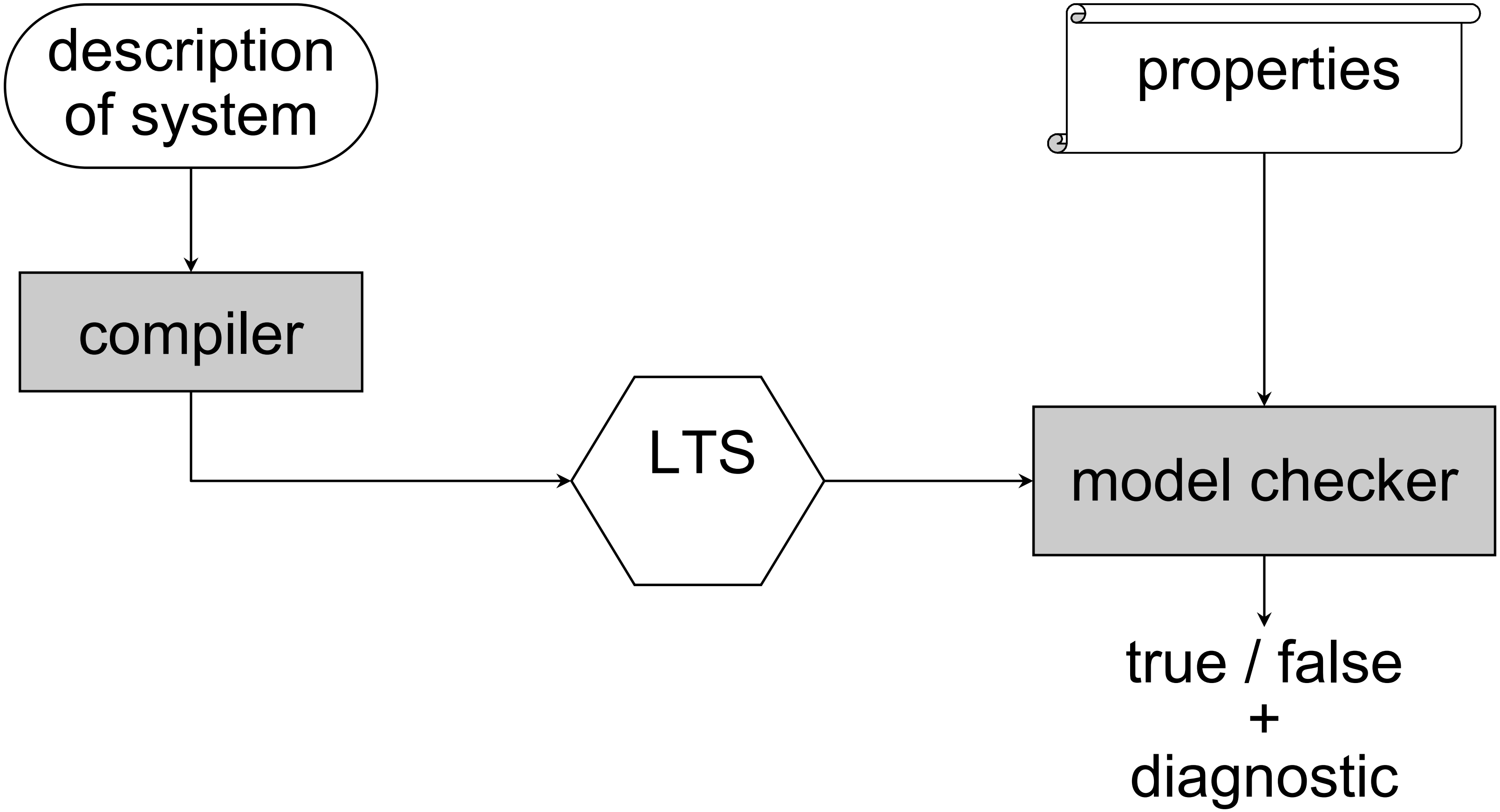
Equivalence checking

(summary)

- *General* boolean graph:
  - All equivalences and their preorders
  - Algorithms A0 and A1 (counterexample depth ↓)
- *Acyclic* boolean graph:
  - Strong equivalence: one LTS acyclic
  - $\tau^*.a$ and safety: one LTS acyclic (τ-circuits allowed)
  - Branching and observational: both LTS acyclic
  - Algorithm A2 (memory ↓)
- *Conjunctive* boolean graph:
  - Strong equivalence: one LTS deterministic
  - Weak equivalences: one LTS deterministic and τ-free
  - Algorithm A4 (memory ↓)

Model checking

(principle)



description of system

compiler

LTS

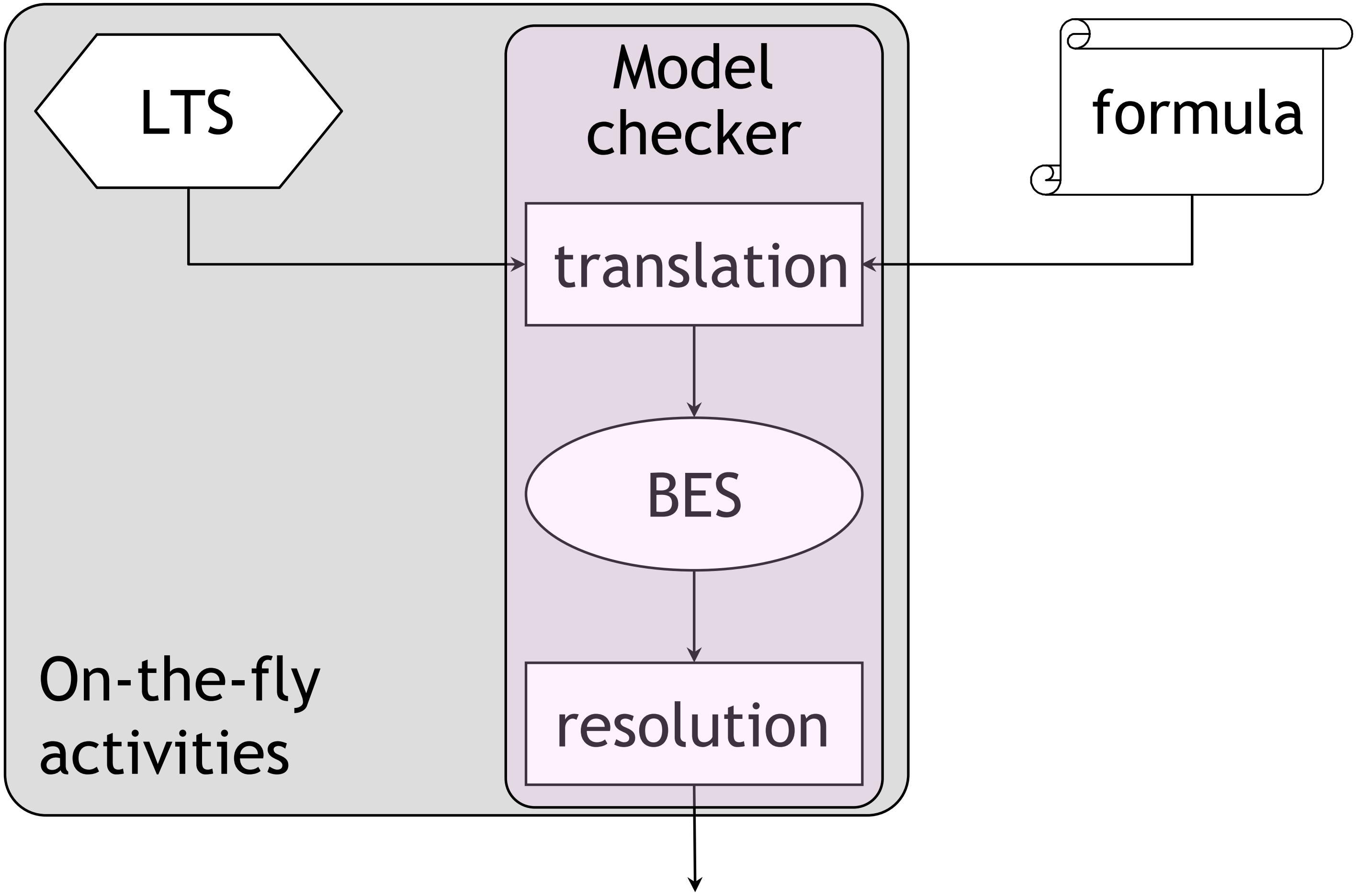
properties

model checker

true / false
+
diagnostic

On-the-fly model checking in CADP

(Evaluator 3.x)



LTS

Model checker

formula

translation

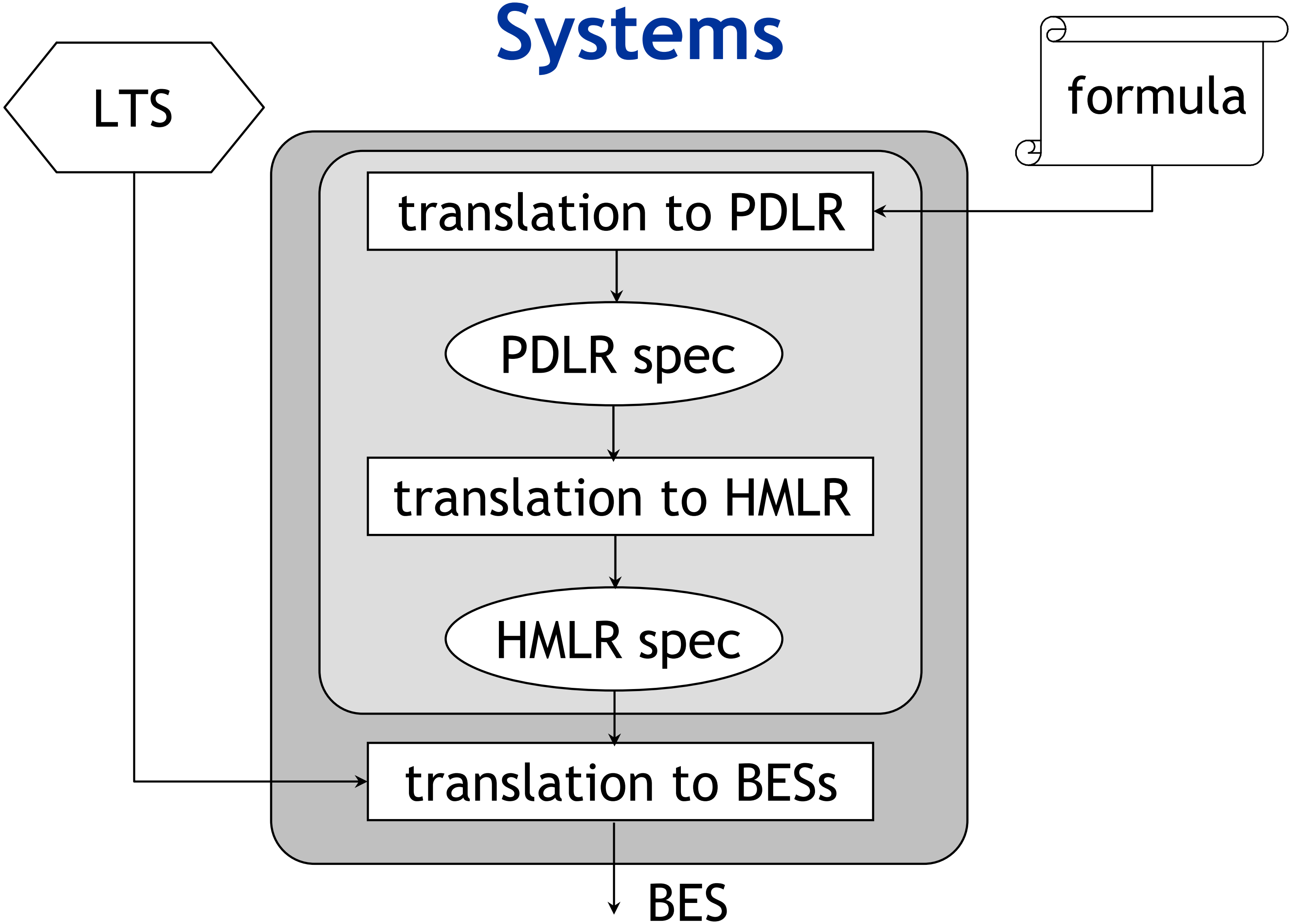
BES

resolution

On-the-fly activities

yes / no + diagnostic

Translation to Boolean Equation Systems



LTS

formula

translation to PDLR

PDLR spec

translation to HMLR

HMLR spec

translation to BESs

BES

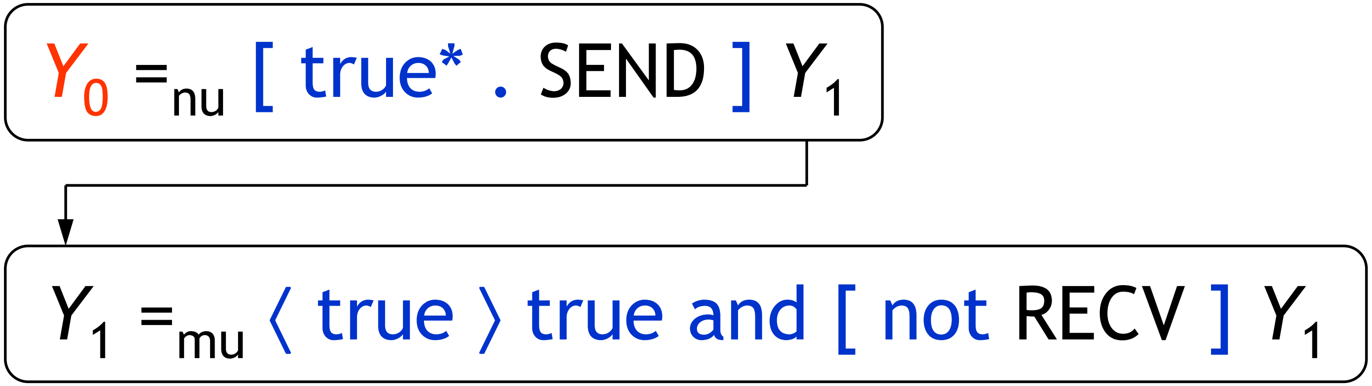
Translation to PDL with recursion

- State formula (expanded):

  nu Y_0 . [true* . SEND]

  mu Y_1 . ⟨ true ⟩ true and [not RECV] Y_1

- PDLR specification [Mateescu-Sighireanu-03]:



$Y_0 =_{nu}$ [true* . SEND] Y_1

$Y_1 =_{mu}$ ⟨ true ⟩ true and [not RECV] Y_1

Simplification

- PDLR specification:

$$Y_0 =_{\text{nu}} [\ \text{true}^* \ .\ \text{SEND}\ ]\ Y_1$$

$$Y_1 =_{\text{mu}} \langle\ \text{true}\ \rangle\ \text{true and}\ [\ \text{not RECV}\ ]\ Y_1$$

- Simple PDLR specification:

$$Y_0 =_{\text{nu}} [\ \text{true}^* \ .\ \text{SEND}\ ]\ Y_1$$

$$Y_1 =_{\text{mu}} Y_2\ \text{and}\ Y_3$$

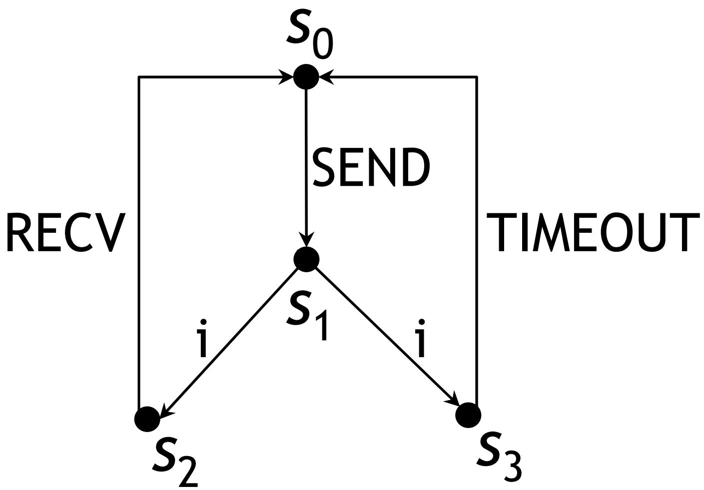
$$Y_2 =_{\text{mu}} \langle\ \text{true}\ \rangle\ \text{true}$$

$$Y_3 =_{\text{mu}} [\ \text{not RECV}\ ]\ Y_1$$

Translation to BESs

$Y_0 =_{nu} Y_4$ and Y_5
$Y_4 =_{nu}$ [SEND] Y_1
$Y_5 =_{nu}$ [true] Y_0

$Y_1 =_{mu} Y_2$ and Y_3
$Y_2 =_{mu}$ ⟨ true ⟩ true
$Y_3 =_{mu}$ [not RECV] Y_1

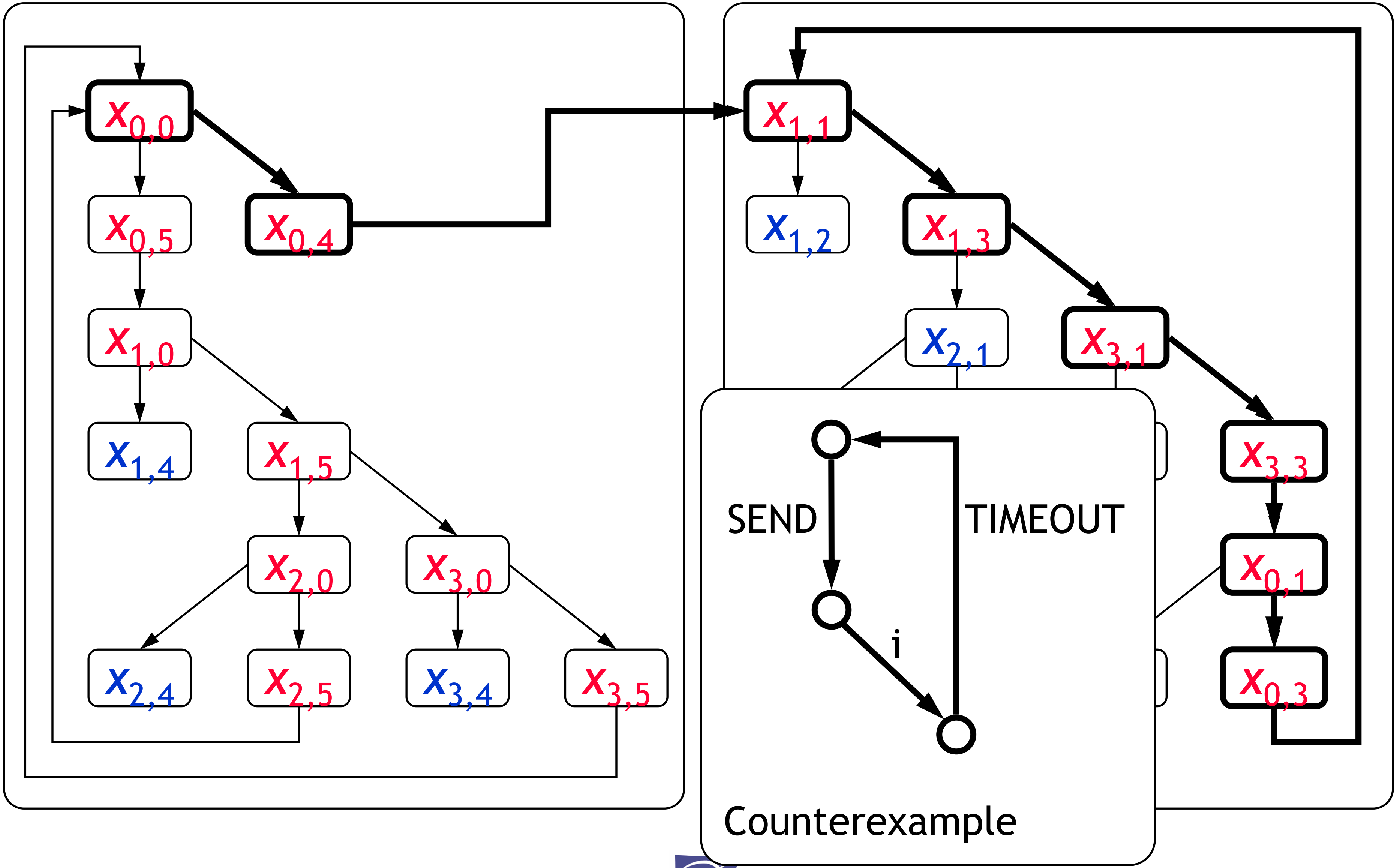


Boolean variables: $x_{i,\,j} \equiv s_i \models Y_j$

$x_{0,0} =_\nu x_{0,4} \wedge x_{0,5}$
$x_{0,4} =_\nu x_{1,1}$
$x_{0,5} =_\nu x_{1,0}$
$x_{1,0} =_\nu x_{1,4} \wedge x_{1,5}$
$x_{1,4} =_\nu$ true
$x_{1,5} =_\nu x_{2,0} \wedge x_{3,0}$
$x_{2,0} =_\nu x_{2,4} \wedge x_{2,5}$
$x_{2,4} =_\nu$ true
$x_{2,5} =_\nu x_{0,0}$
$x_{3,0} =_\nu x_{3,4} \wedge x_{3,5}$
$x_{3,4} =_\nu$ true
$x_{3,5} =_\nu x_{0,0}$

$x_{1,1} =_\mu x_{1,2} \wedge x_{1,3}$
$x_{1,2} =_\mu$ true
$x_{1,3} =_\mu x_{2,1} \wedge x_{3,1}$
$x_{2,1} =_\mu x_{2,2} \wedge x_{2,3}$
$x_{2,2} =_\mu$ true
$x_{2,3} =_\mu$ true
$x_{3,1} =_\mu x_{3,2} \wedge x_{3,3}$
$x_{3,2} =_\mu$ true
$x_{3,3} =_\mu x_{0,1}$
$x_{0,1} =_\mu x_{0,2} \wedge x_{0,3}$
$x_{0,2} =_\mu$ true
$x_{0,3} =_\mu x_{1,1}$

Local BES resolution with diagnostic



Counterexample

Additional operators

- Mechanisms for macro-definition (overloaded) and library inclusion
- Libraries encoding the operators of CTL and ACTL

  $\text{EU}\ (\varphi_1, \varphi_2) = \text{mu}\ Y\ .\ \varphi_2\ \text{or}\ (\varphi_1\ \text{and}\ \langle\ \text{true}\ \rangle\ Y)$

  $\text{EU}\ (\varphi_1, \alpha_1, \alpha_2\ , \varphi_2) = \text{mu}\ Y\ .\ \langle\ \alpha_2\ \rangle\ \varphi_2\ \text{or}\ (\varphi_1\ \text{and}\ \langle\ \alpha_1\ \rangle\ Y)$
- Libraries of high-level property patterns [Dwyer-99]
  - Property classes:
    - Absence, existence, universality, precedence, response
  - Property scopes:
    - Globally, before *a*, after *a*, between *a* and *b*, after *a* until *b*
  - More info:
    - http://www.inrialpes.fr/vasy/cadp/resources

Disjunctive BES

- *Disjunctive* boolean graph:
  - *Potentiality* operator of CTL

    $E\ [\varphi_1\ U\ \varphi_2] = \mu X\ .\ \varphi_2 \lor (\varphi_1 \land \langle\ T\ \rangle\ X)$

    $\{\ X =_\mu \varphi_2 \lor Y\ \ ,\ Y =_\mu \varphi_1 \land Z\ \ ,\ Z =_\mu \langle\ T\ \rangle\ X\ \}$

    $\{\ X_s =_\mu \varphi_{2s} \lor Y_s\ \ ,\ Y_s =_\mu \varphi_{1s} \land Z_s\ \ ,\ Z_s =_\mu \lor_{s \to s'} X_{s'}\ \}$
  - *Possibility* modality of PDL

    $\langle\ (a\ |\ b)^*\ .\ c\ \rangle\ T$

    $\{\ X =_\mu \langle\ c\ \rangle\ T \lor \langle\ a\ \rangle\ X \lor \langle\ b\ \rangle\ X\ \}$

    $\{\ X_s =_\mu (\lor_{s \to c\ s'}\ T) \lor (\lor_{s \to a\ s'}\ X_{s'}) \lor (\lor_{s \to b\ s'}\ X_{s'})\ \}$
- Algorithm A3 (memory ↓)

Linear-time model checking

(looping operator of PDL-delta)

- Translation in mu-calculus of alternation depth 2 [Emerson-Lei-86]:

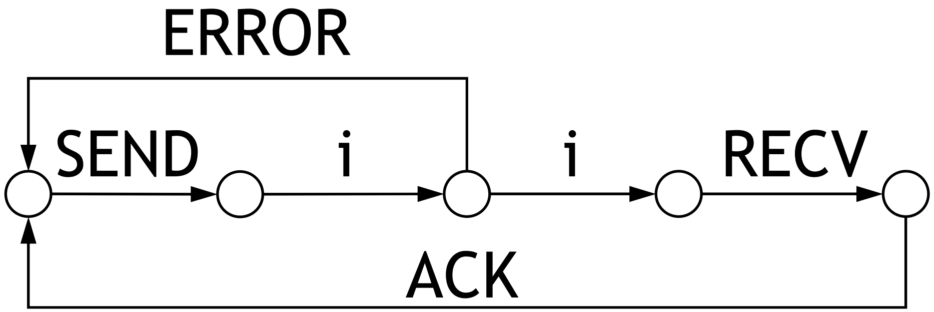
  < R > @ = nu X . < R > X

if R contains *-operators, the formula is of alternation depth 2

- But still checkable in linear-time:
  - Mark LTS states potentially satisfying X
  - Leads to marked variables in the disjunctive BES
  - Computation of boolean SCCs containing marked variables
  - $A3_{cyc}$ algorithm [Mateescu-Thivolle-08]
    - Can serve for LTL model checking
    - Allows linear-time handling of repeated invocations

Model checking of data-based properties

(Evaluator 4.0)



- Every SEND is followed by a RECV after 2 steps:

```
[ true* . SEND ] < true { 2 } . RECV > true =
nu X . ( [ SEND ] mu Y (c:Nat := 2) .
                            if c = 0 then < RECV > true
                                 else < true > Y (c – 1)
                            end if
       and
       [ true ] X )
```

Translation into HMLR

nu X . [SEND]

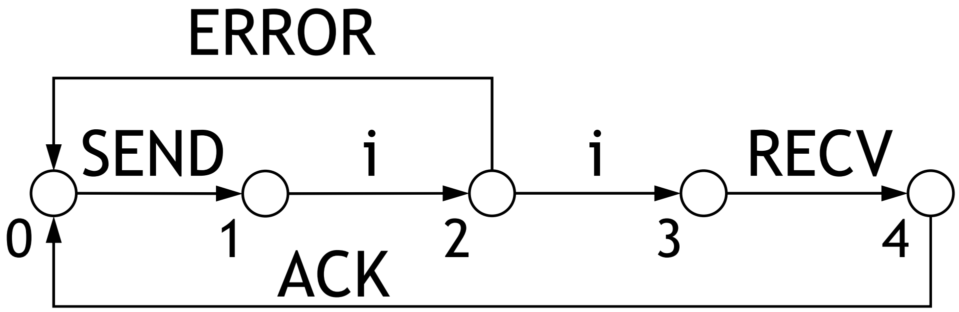
and [true] X

mu Y (c:Nat := 2) .
if c = 0 then < RECV > true
else < true > Y (c - 1)
end if

{ X $=_{nu}$
[SEND] Y (2)
and
[true] X
}

{ Y (c:Nat) $=_{mu}$
if c = 0 then < RECV > true
else < true > Y (c - 1)
end if
}

Translation into BES and resolution



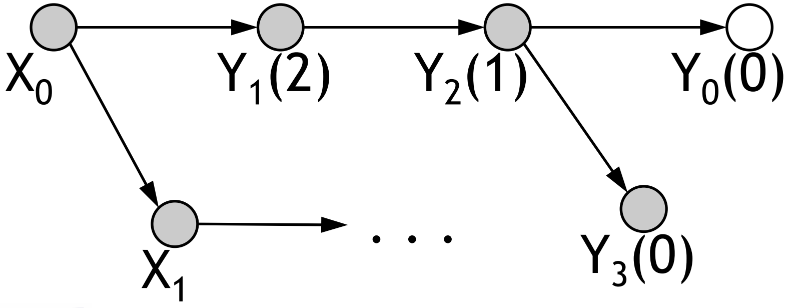
```
{ X =nu
        [ SEND ] Y (2)
        and
        [ true ] X
}
```

```
{ Y (c:Nat) =mu
        if c = 0 then < RECV > true
           else < true > Y (c - 1)
        end if
}
```

- Principle:

$$X_s \ = \ \text{«}\ s\ |= X\ \text{»}$$

$$Y_s\ (c) \ = \text{«}\ s\ |= Y\ (c)\ \text{»}$$

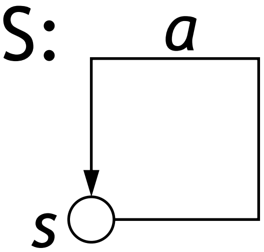


Divergence

- In presence of data parameters of infinite types, termination of model checking is not guaranteed anymore
- (pathological) property:

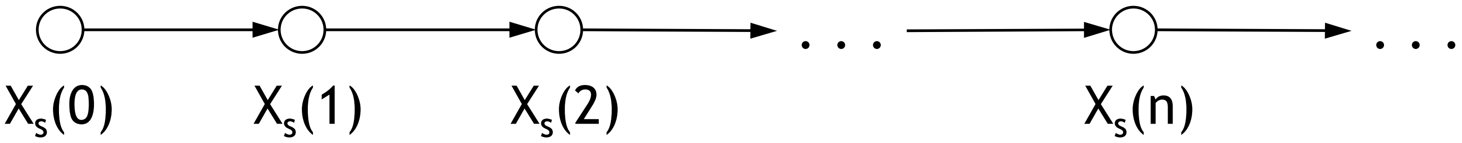
  mu X (n:Nat := 0) . < *a* > X (n + 1)

LTS:



- BES : $\{ X_s\ (n{:}\text{Nat}) =_{mu} \text{OR}_{s \text{ ->a } s'}\ X_{s'}\ (n + 1) \} =$

  $\{ X_s\ (n{:}\text{Nat}) =_{mu} X_s\ (n + 1) \}$



Conjunctive BES

- *Conjunctive* boolean graph:
  - *Inevitability* operator of CTL

    $\mathrm{A}\ [\varphi_1\ \mathrm{U}\ \varphi_2] = \mu X\ .\ \varphi_2 \vee (\varphi_1 \wedge \langle\ \mathrm{T}\ \rangle\ \mathrm{T} \wedge [\ \mathrm{T}\ ]\ X)$

    $\{\ X =_\mu \varphi_2 \vee Y\ \ ,\ Y =_\mu \varphi_1 \wedge Z \wedge [\ \mathrm{T}\ ]\ X\ ,\ Z =_\mu \langle\ \mathrm{T}\ \rangle\ \mathrm{T}\ \}$

    $\{\ X_s =_\mu \varphi_{2s} \vee Y_s\ \ ,\ Y_s =_\mu \varphi_{1s} \wedge Z_s \wedge (\wedge_{s \to s'} X_{s'})\ ,\ Z_s =_\mu \vee_{s \to s'} \mathrm{T}\ \}$
  - *Necessity* modality of PDL

    $[\ (a\ |\ b)^*\ .\ c\ ]\ \mathrm{F}$

    $\{\ X =_\mu [\ c\ ]\ \mathrm{F} \wedge [\ a\ ]\ X \wedge [\ b\ ]\ X\ \}$

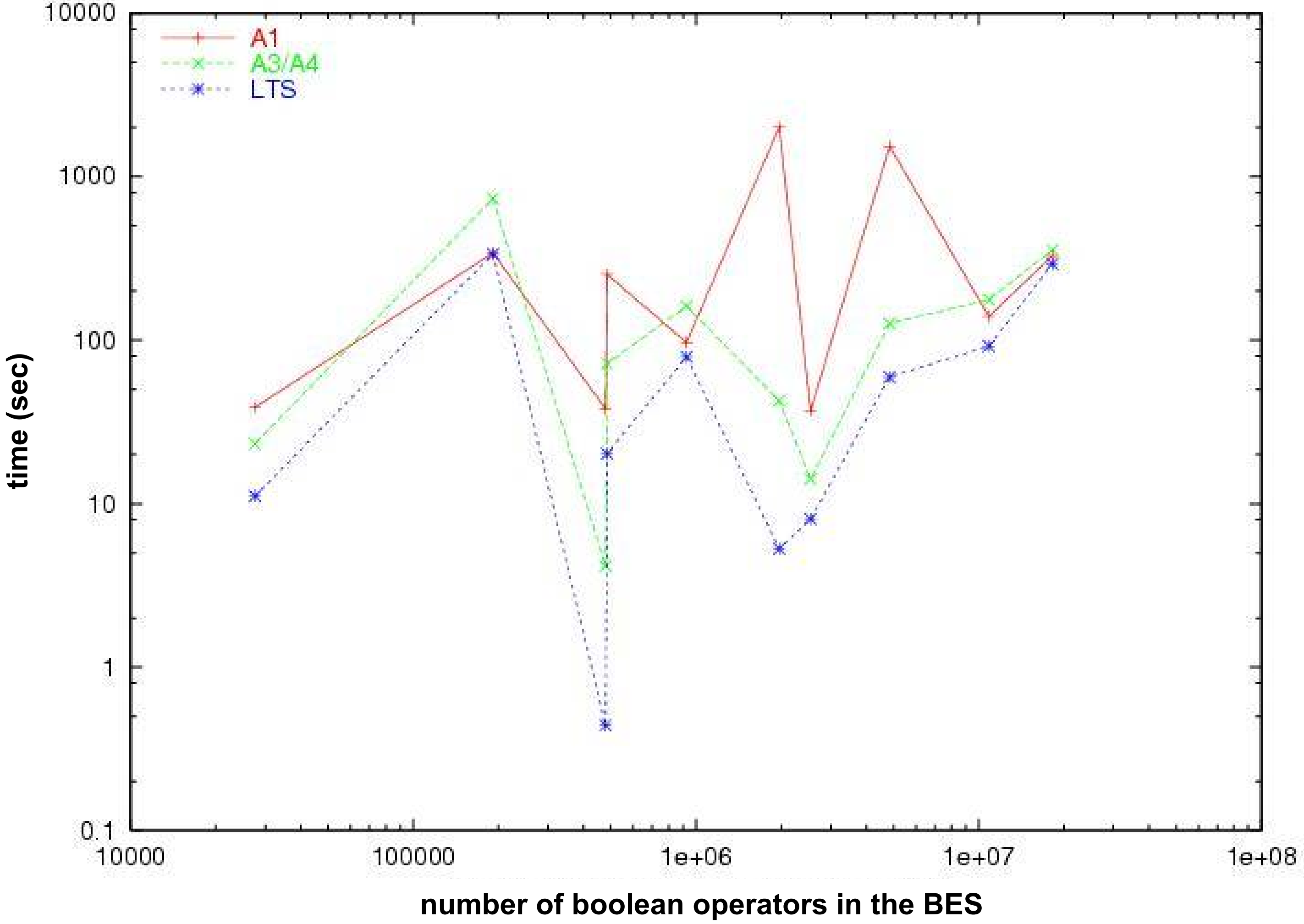
    $\{\ X_s =_\mu (\wedge_{s \to c\ s'}\ \mathrm{F}) \wedge (\wedge_{s \to a\ s'}\ X_{s'}) \wedge (\wedge_{s \to b\ s'}\ X_{s'})\ \}$
- Algorithm A4 (memory ↓)

Acyclic BES

- *Acyclic* boolean graph:
  - *Acyclic* LTS and *guarded* formulas [Mateescu-02]
- Handling of CTL (and ACTL) operators:
  - $E\,[\varphi_1\ U\ \varphi_2] = \mu X\,.\,\varphi_2 \vee (\varphi_1 \wedge \langle\, T \,\rangle\, X)$
  - $A\,[\varphi_1\ U\ \varphi_2] = \mu X\,.\,\varphi_2 \vee (\varphi_1 \wedge \langle\, T \,\rangle\, T \wedge [\, T \,]\, X)$
- Handling of full mu-calculus
  - Translation to guarded form
  - Conversion from maximal to minimal fixed points [Mateescu-02]
- Algorithm A2 (memory ↓)

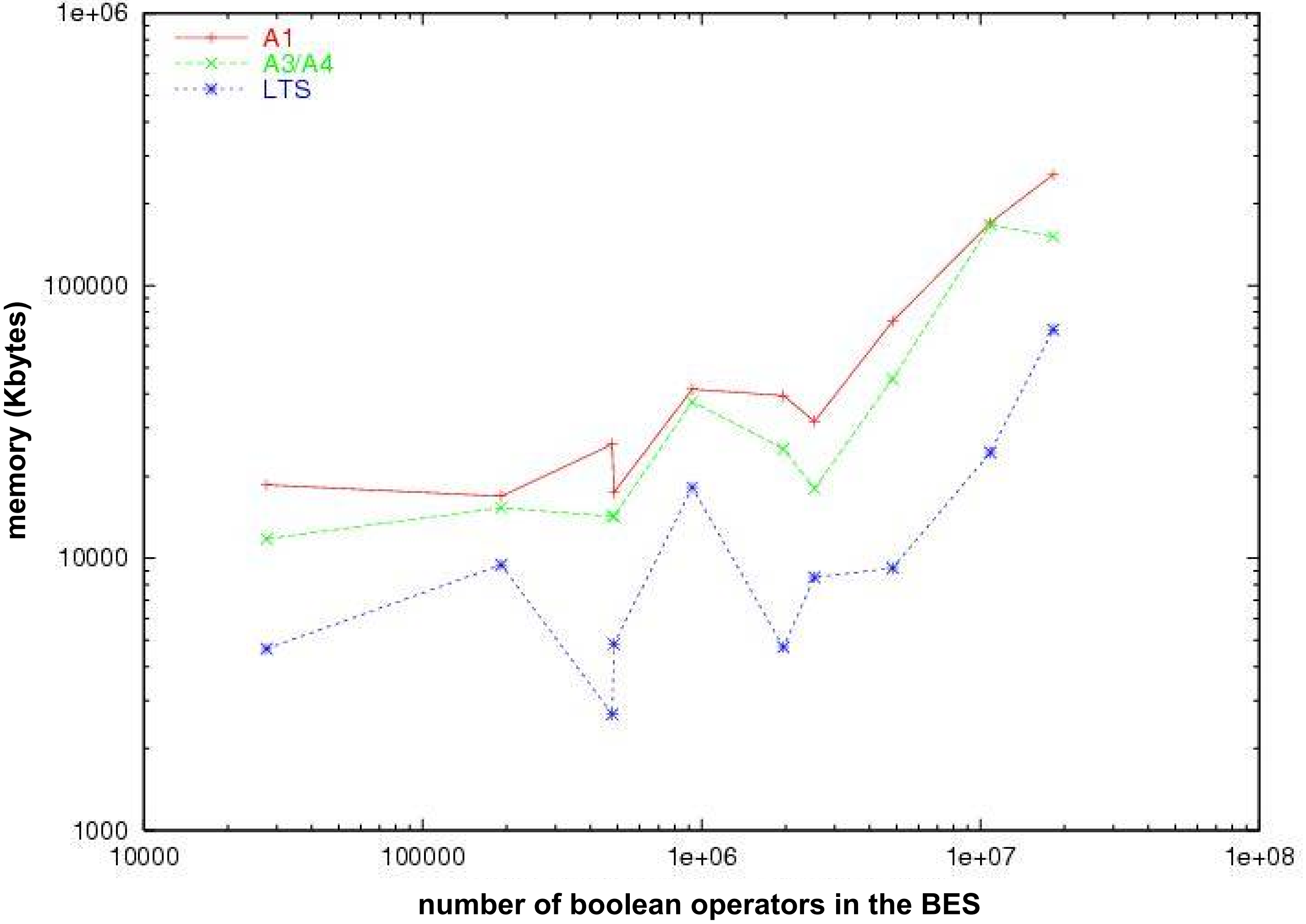
Algorithm A1 vs. A3/A4

(execution time – CADP demos)



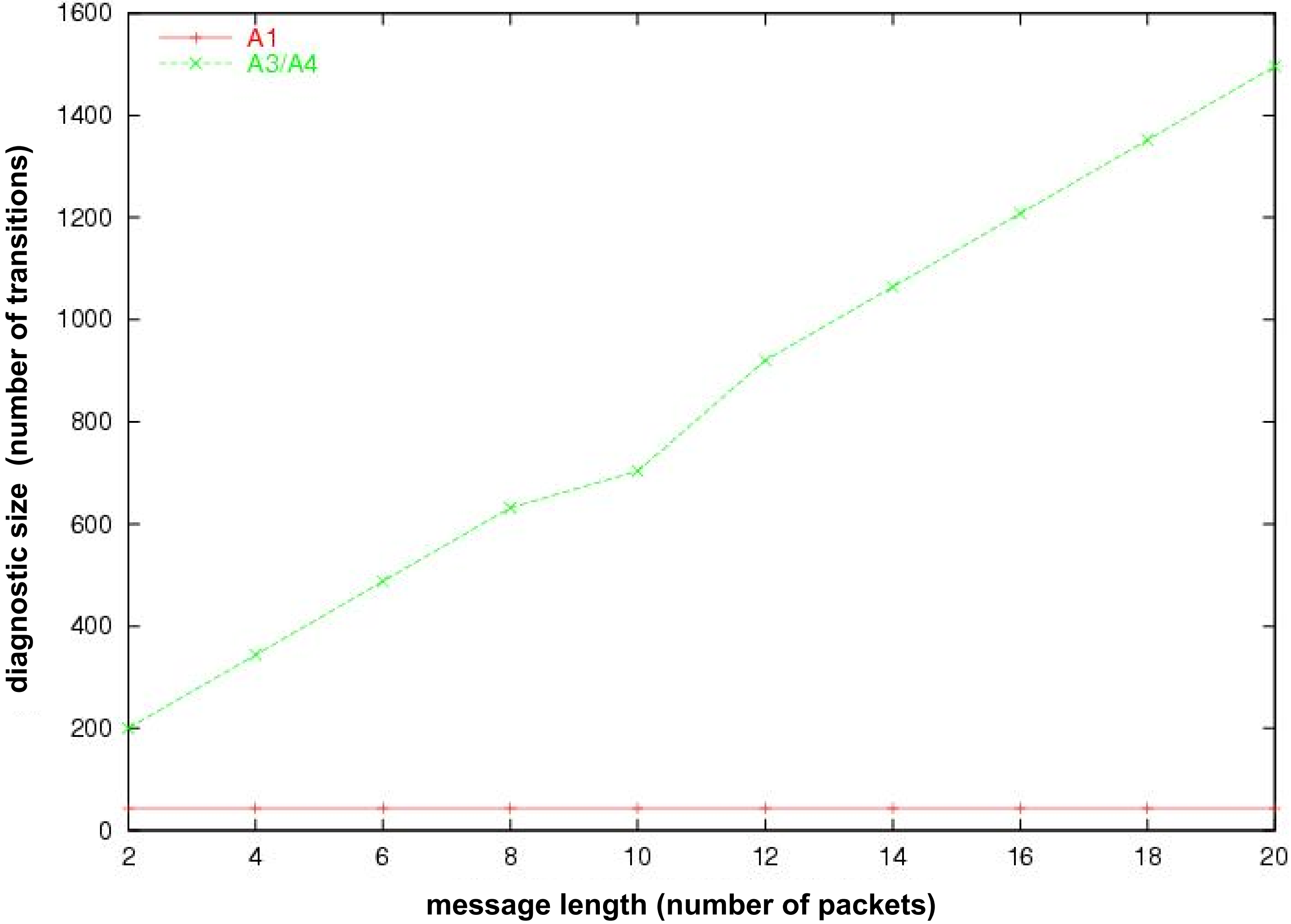
Algorithm A1 vs. A3/A4

(memory consumption – CADP demos)



Algorithm A1 vs. A3/A4

(diagnostic size – BRP protocol)



Model checking

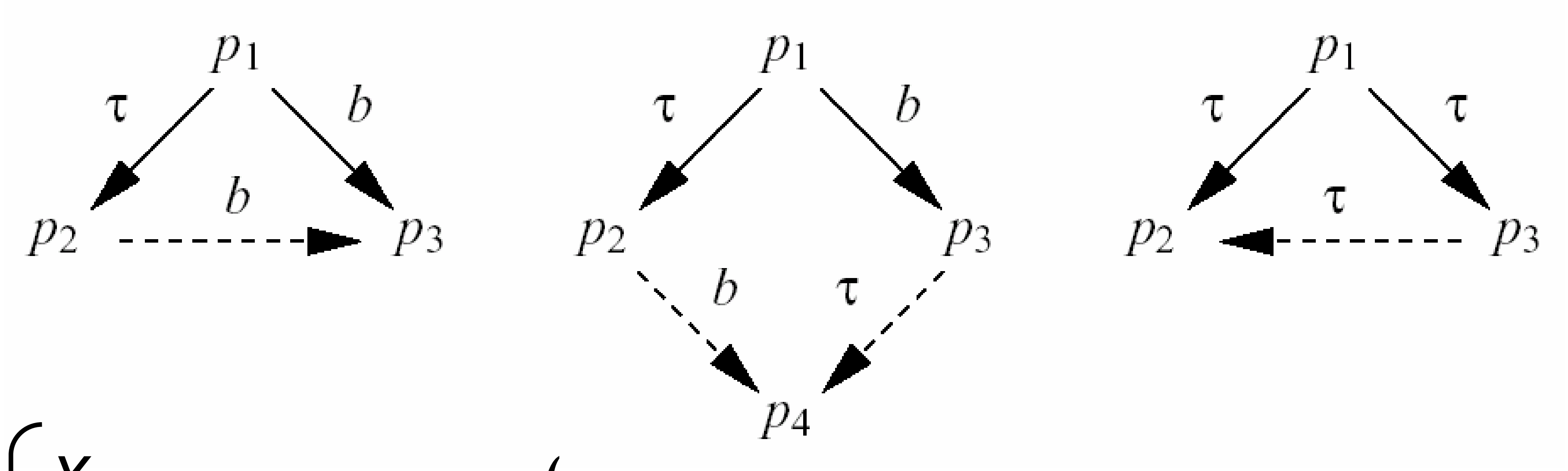
(summary)

- *General* boolean graph:
  - Any LTS and any alternation-free μ-calculus formula
  - Algorithms A0 and A1 (diagnostic depth ↓)
- *Acyclic* boolean graph:
  - Acyclic LTS and guarded formula (CTL, ACTL)
  - Acyclic LTS and μ-calculus formula (via reduction)
  - Algorithm A2 (memory ↓)
- *Disjunctive/conjunctive* boolean graph:
  - Any LTS and any formula of CTL, ACTL, PDL
  - Algorithm A3/A4 (memory ↓)
  - Matches the best local algorithms dedicated to CTL [Vergauwen-Lewi-93]

Partial order reduction

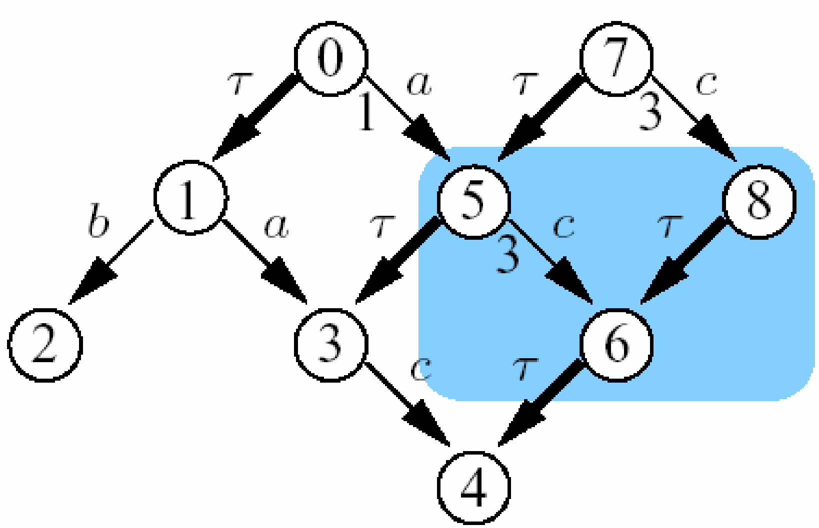
- *τ-confluence* [Groote-vandePol-00]
  - Form of partial-order reduction defined on LTSs
  - Preserves branching bisimulation
- Principle
  - Detection of τ-confluent transitions
  - Elimination of “neighbour” transitions (*τ-prioritisation*)
- On-the-fly LTS reduction
  - Direct approach [Blom-vandePol-02]
  - BES-based approach [Pace-Lang-Mateescu-03]
    - Define τ-confluence in terms of a BES
    - Detect τ-confluent transitions by locally solving the BES
    - Apply τ-prioritisation and compression on sequences

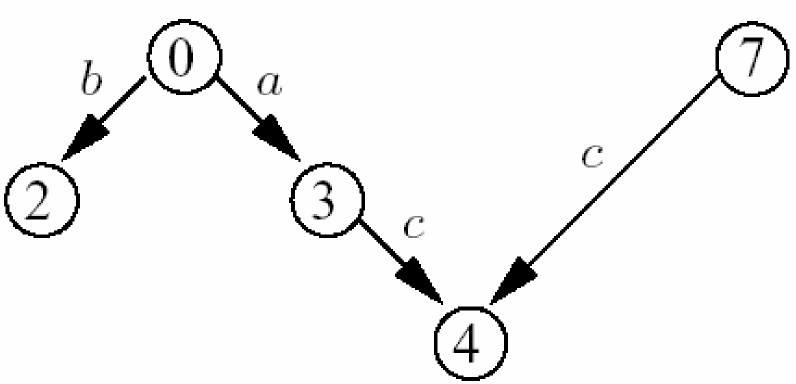
Translation to a BES



$$\left\{ \begin{array}{l} X_{p1,p2} =_{\nu} \wedge_{p1 \rightarrow b\ p3} (\\ \qquad\qquad p2 \rightarrow b\ p3 \vee \\ \qquad\qquad \vee_{p2 \rightarrow b\ p4,\ p3 \rightarrow \tau\ p4} X_{p3,p4} \vee \\ \qquad\qquad ((b = \tau) \wedge \vee_{p3 \rightarrow \tau\ p2} X_{p3,p2}) \\ \qquad) \end{array} \right.$$

Tau-prioritisation and compression





Original LTS

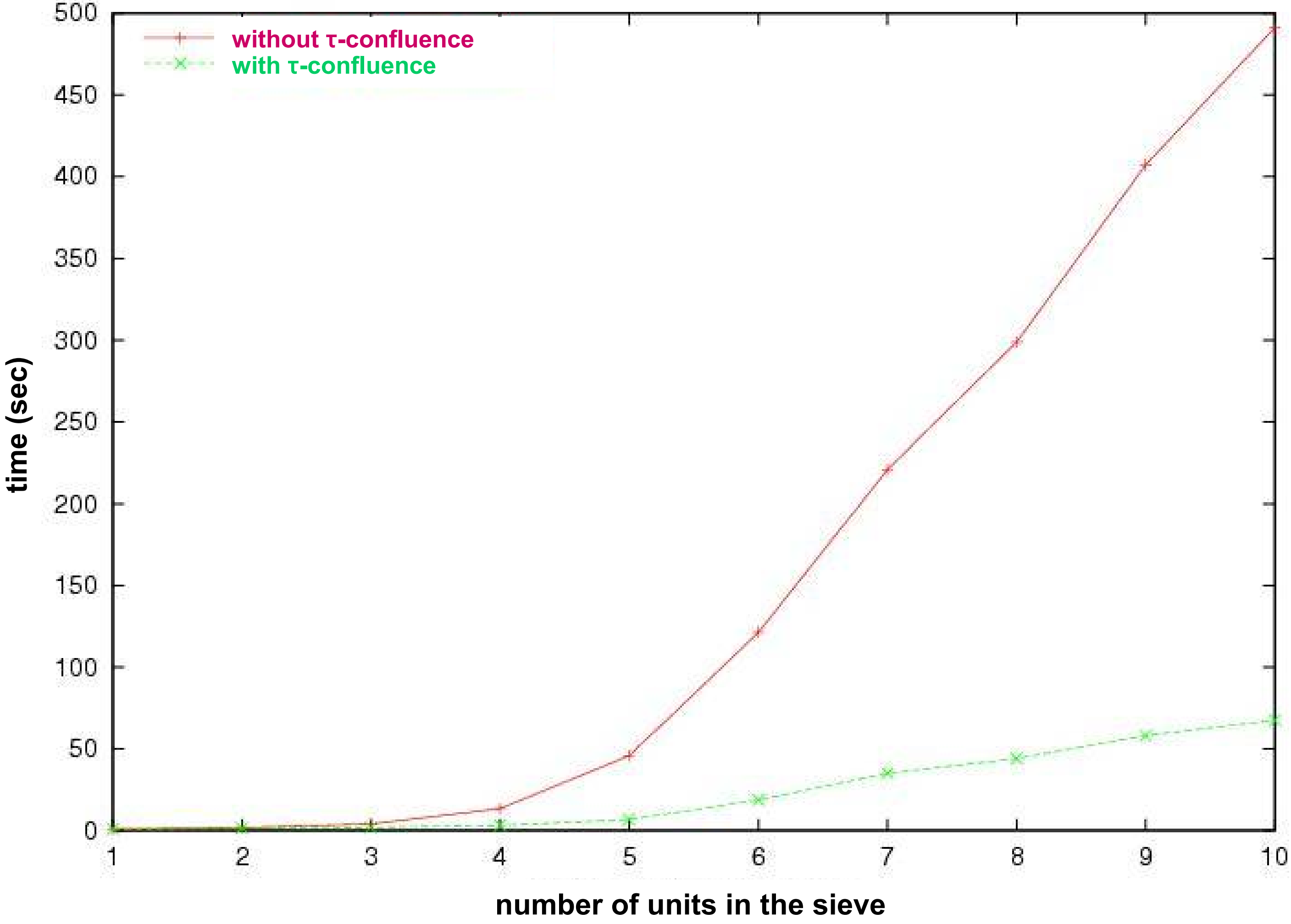
(exploration from s_0 and s_7)

Reduced LTS

- In practice: reductions of a factor 10^2 – 10^3 [Mateescu-05]

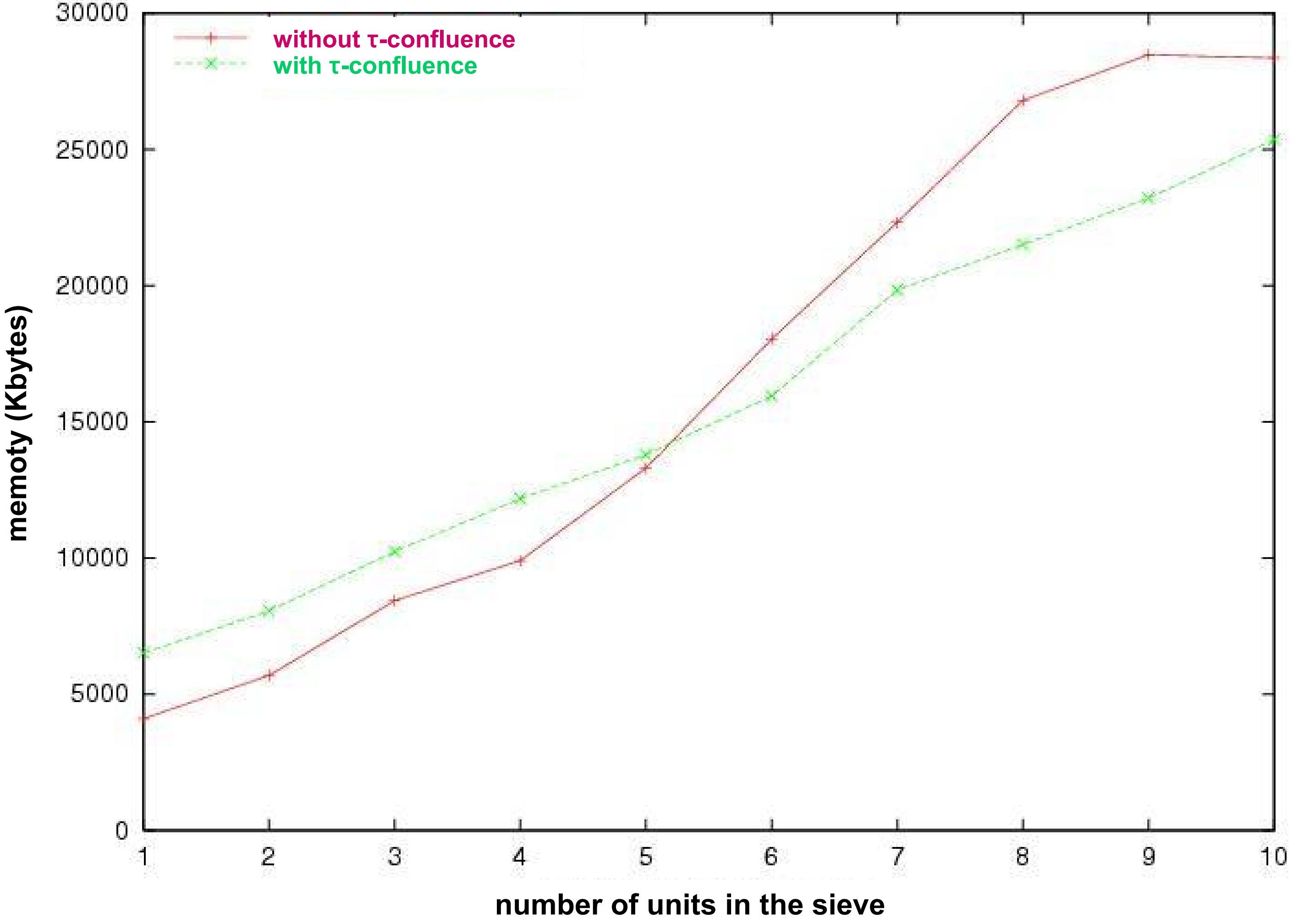
Model checking using A3/A4

(effect of τ-confluence reduction – time – Erathostene's sieve)



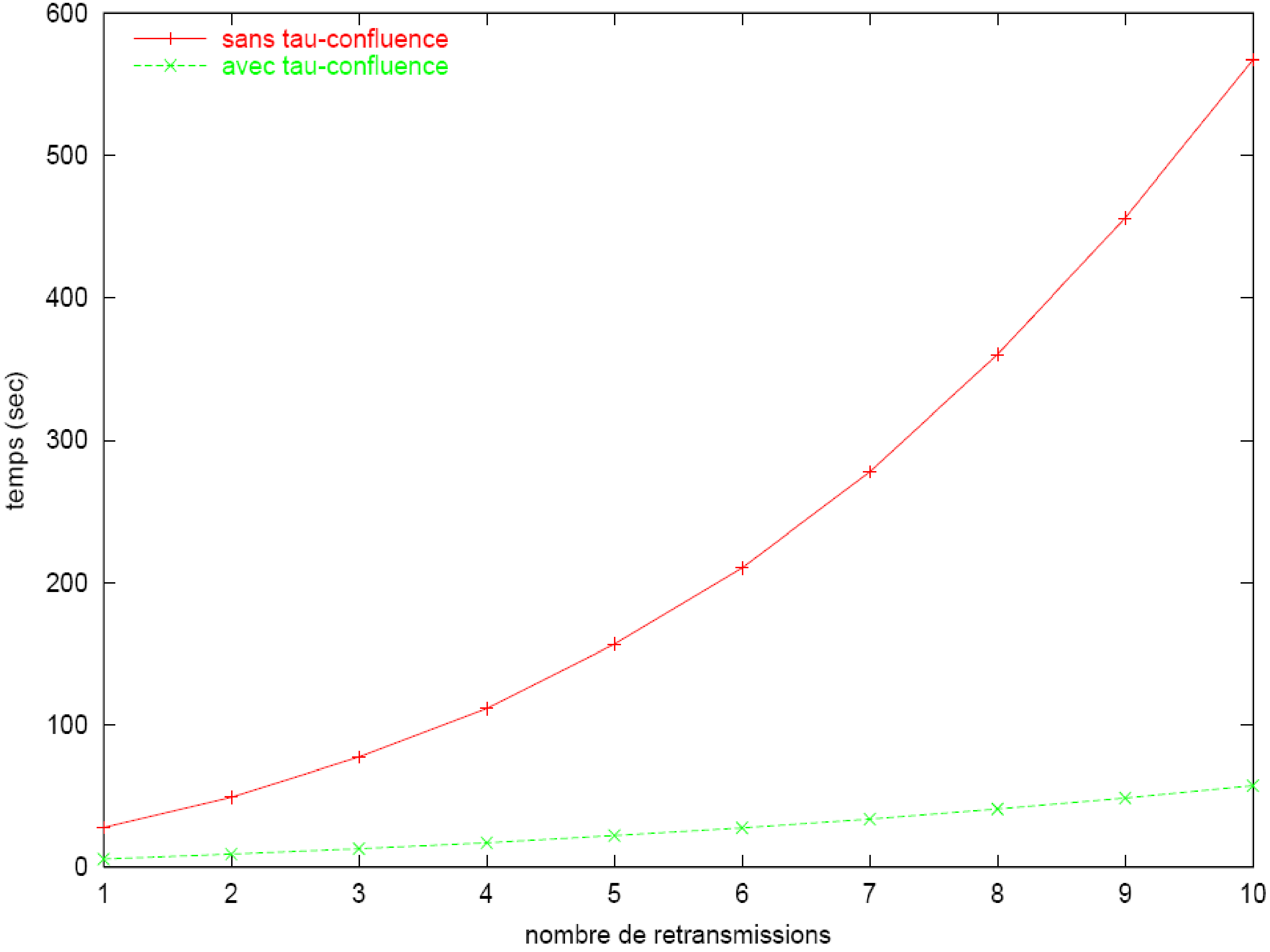
Model checking using A3/A4

(effect of τ-confluence reduction – memory – Erathostene's sieve)



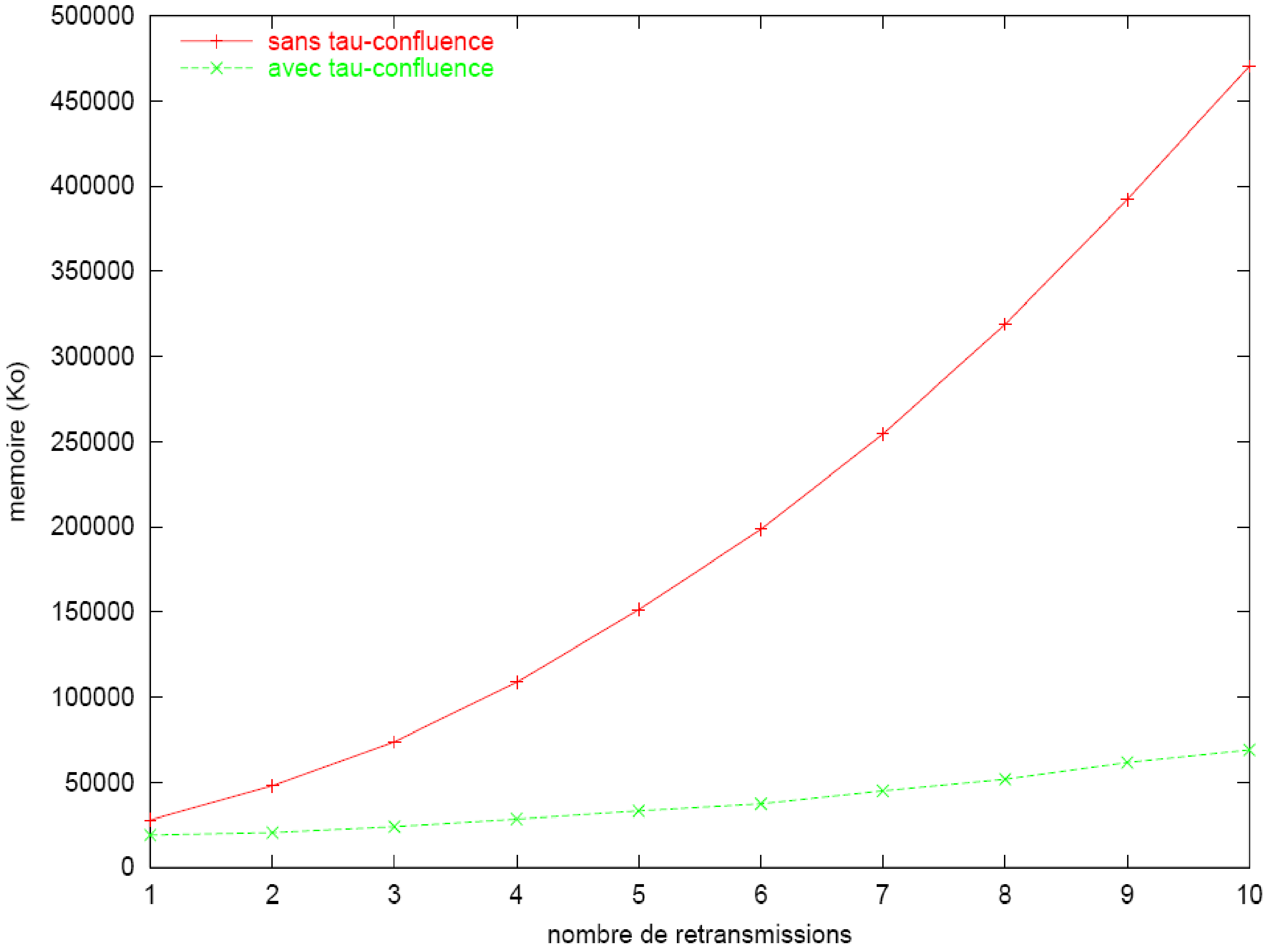
Checking branching bisimulation

(effect of τ-confluence reduction – time – BRP protocol)



Checking branching bisimulation

(effect of τ-confluence reduction – memory – BRP protocol)



On-the-fly verification

(summary)

Already available:

- Generic Caesar_Solve library [Mateescu-03,06]
- 9 local BES resolution algorithms (A8 added in 2008)
- Diagnostic generation features
- Applications: Bisimulator, Evaluator 3.5, Reductor 5.0

Ongoing:

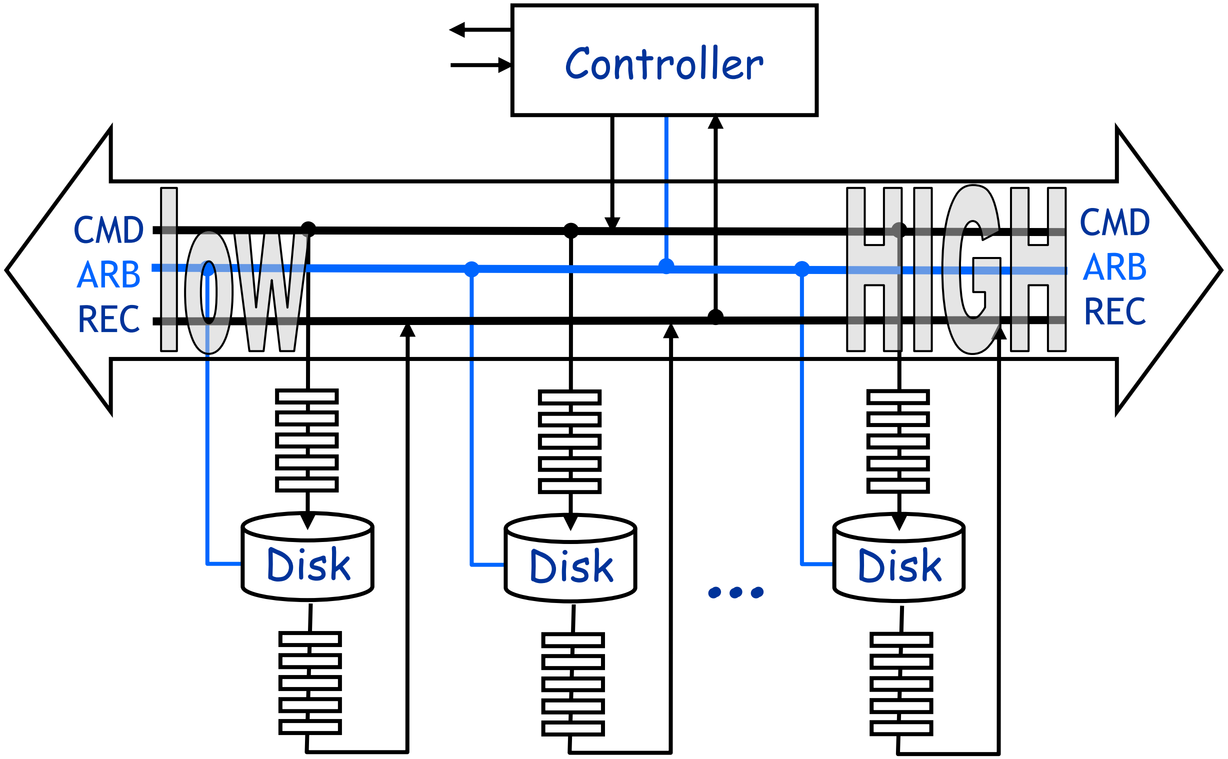
- Distributed BES resolution algorithms on clusters of machines [Joubert-Mateescu-04,05,06]
- New applications
  - Test generation
  - Software adaptation
  - Discrete controller synthesis

Case study

- SCSI-2 bus arbitration protocol
- Description in LOTOS
- Specification of properties in TL
- Verification using Evaluator 3.5 and 4.0
- Interpretation of diagnostics

SCSI-2 bus arbitration protocol

- Prioritized arbitration mechanism, based on static IDs on bus (devices numbered from 0 to n – 1)
- Fairness problem (starvation of low-priority disks)



Architecture of the system

```
(
    DISK [ARB, CMD, REC] (0, 0)
    |[ARB]|
    DISK [ARB, CMD, REC] (1, 0)
    |[ARB]|
    ...
    |[ARB]|
    DISK [ARB, CMD, REC] (6, 0)
)
|[ARB, CMD, REC]|
CONTROLLER [ARB, CMD, REC] (NC, ZERO)
```

8-ary rendezvous
on gate ARB

binary rendezvous
on gates CMD, REC

Synchronization constraints

(bus arbitration policy)

- Synchronizations on gate ARB:

  ARB ?r0, …,r7:Bool [C (r0, …, r7, n)] ; …

  where:
  - r0, …, r7 = values of the electric signals on the bus
  - n = index of the current device

- Two particular cases for guard condition C:
  - P (r0, …, r7, n): device n does not ask the bus
  - A (r0, …, r7, n): device n asks and obtains access to bus

Guard conditions

- Predicate P (r0, …, r7, n) = $\neg r_n$

  P (r0, …, r7, 0) = not (r0)

  P (r0, …, r7, 1) = not (r1)

  …

  P (r0, …, r7, 7) = not (r7)

- Predicate A (r0, …, r7, n) = $r_n \wedge \forall i \in [n+1, 7] . \neg r_i$

  A (r0, …, r7, 0) = r0 and not (r1 or … or r7)

  A (r0, …, r7, 1) = r1 and not (r2 or … or r7)

  …

  A (r0, …, r7, 7) = r7

Controller process

```
process Controller [ARB, CMD, REC] (C:Contents) : noexit :=
    (* communicate with disk N *)
    choice N:Nat []
        [(N >= 0) and (N <= 6)] ->
                Controller2 [ARB, CMD, REC] (C, N)
    []
    (* does not request the bus *)
    ARB ?r0, ..., r7:Bool [P (r0, ..., r7, 7)];
        Controller [ARB, CMD, REC] (C)
endproc
```

Controller process

```
process Controller2 [ARB, CMD, REC] (C:Contents, N:Nat) :
noexit :=
   [not_full (C, N)] ->
        (* request and obtain the bus *)
        ARB ?r0, ..., r7:Bool [A (r0, ..., r7, 7)];
                CMD !N; (* send a command *)
                        Controller [ARB, CMD, REC] (incr (C, N))
   []
   REC !N;     (* receive an acknowledgement *)
        Controller [ARB, CMD, REC] (decr (C, N))
endproc
```

Disk process

```
process DISK [ARB, CMD, REC] (N, L:Nat) : noexit :=
   CMD !N;  DISK [ARB,CMD,REC] (N, L+1)
   []
   [L > 0] -> (
        ARB ?r0, ..., r7:Bool [A (r0, ..., r7, N)];
                REC !N;  DISK [ARB, CMD, REC] (N, L-1)
        []
        ARB ?r0, ..., r7:Bool [not (A (r0, ..., r7, N)) and
                                  not (P (r0, ..., r7, N))];
                DISK [ARB, CMD, REC] (N, L)
   )
   []
   [L = 0] -> ARB ?r0, ..., r7:Bool [P (r0, ..., r7, N)];
                        DISK [ARB, CMD, REC] (N, L)
endproc
```

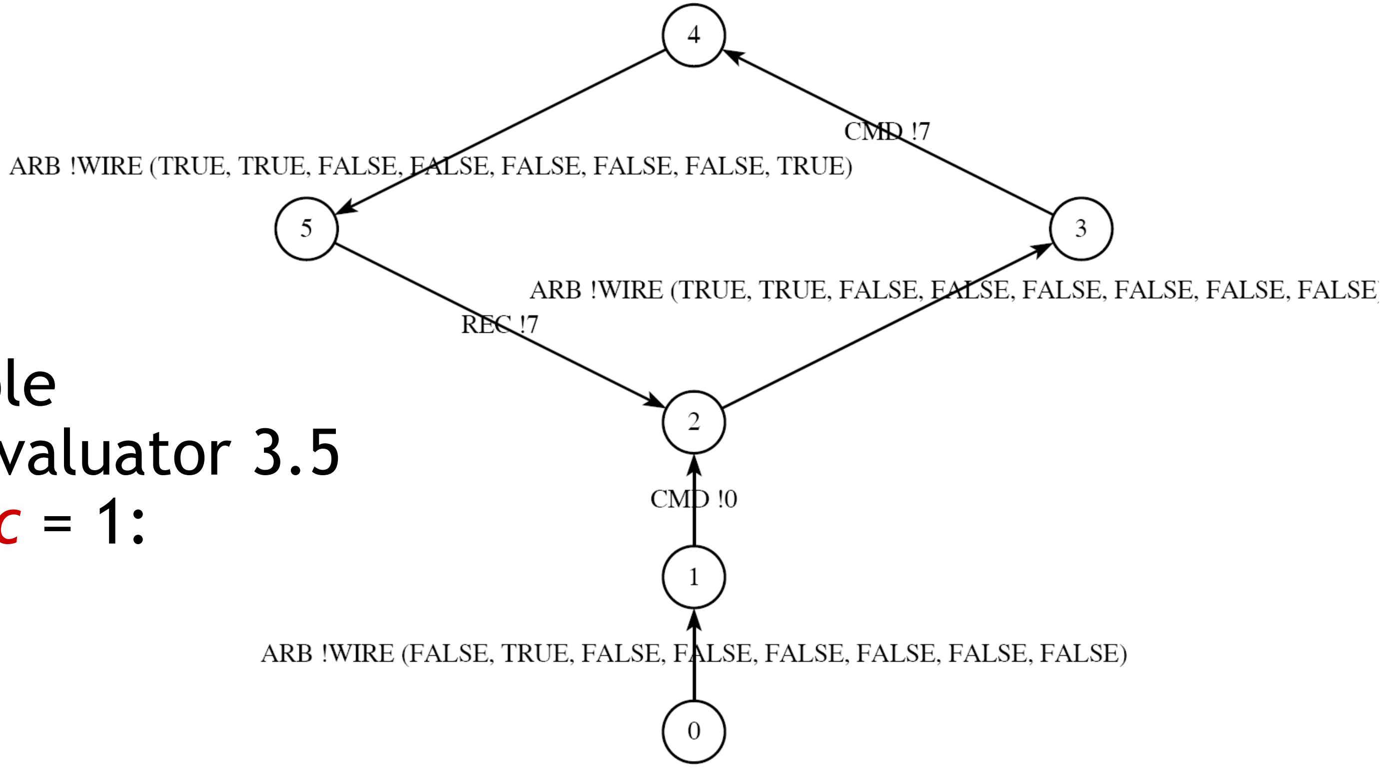
Absence of starvation property

(PDL+ACTL formulation)

"Every time a disk i *receives a command from the controller, it will be able to gain access to the bus in order to send the corresponding acknowledgement"*

$$[\ \mathrm{true}^{*}\ .\ \mathrm{cmd}_{i}\ ]\ \mathrm{A}\ [\ \mathrm{true}_{\mathrm{true}}\ \mathrm{U}_{\mathrm{rec}i}\ \mathrm{true}\ ]$$

- Property fails for *i* < *nc*
- Counterexample produced by Evaluator 3.5 for *i* = 0 and *nc* = 1:



Starvation property

(MCL formulation)

"Every time a disk i with priority lower than the controller nc receives a command, its access to the bus can be continuously preempted by any other disk j with higher priority"

```
[ true*. {cmd ?i:Nat where i < nc} ]
        forall j:Nat among { i + 1 ... n − 1 } .
                (j <> nc) implies
                < (not {rec !i})*. {cmd !j} .
                   (not {rec !i})*. {rec !j} > @
```

Safety property

(MCL formulation)

“The difference between the number of commands received and reconnections sent by a disk i varies between 0 and 8 (the size of the buffers associated to disks)”

```
forall i:Nat among { 0 … n – 1 } .
        nu Y (c:Nat:=0) . (
                [ {cmd !i} ] ((c < 8) and Y (c + 1))
                and
                [ {rec !i} ] ((c > 0) and Y (c − 1))
                and
                [ not ({cmd !i} or {rec !i}) ] Y (c)
        )
```

Safety property

(standard mu-calculus formulation)

```
nu CMD_REC_0 . (
    [ CMD_i ] nu CMD_REC_1 . (
      [ CMD_i ] nu CMD_REC_2 . (
        [ CMD_i ] nu CMD_REC_3 . (
          [ CMD_i ] nu CMD_REC_4 . (
            [ CMD_i ] nu CMD_REC_5 . (
              [ CMD_i ] nu CMD_REC_6 . (
                [ CMD_i ] nu CMD_REC_7 . (
                  [ CMD_i ] nu CMD_REC_8 . (
                    [ CMD_i ] false
                    and
                    [ REC_i ] CMD_REC_7
                    and
                    [ not ((CMD_i) or (REC_i)) ] CMD_REC_8
                  )
                  and
                  [ REC_i ] CMD_REC_6
                  and
                  [ not ((CMD_i) or (REC_i)) ] CMD_REC_7
                )
                and
                [ REC_i ] CMD_REC_5
                and
                [ not ((CMD_i) or (REC_i)) ] CMD_REC_6
              )
              and
              [ REC_i ] CMD_REC_4
              and
              [ not ((CMD_i) or (REC_i)) ] CMD_REC_5
            )
            and
            [ REC_i ] CMD_REC_3
            and
            [ not ((CMD_i) or (REC_i)) ] CMD_REC_4
          )
          and
          [ REC_i ] CMD_REC_2
          and
          [ not ((CMD_i) or (REC_i)) ] CMD_REC_3
        )
        and
        [ REC_i ] CMD_REC_1
        and
        [ not ((CMD_i) or (REC_i)) ] CMD_REC_2
      )
      and
      [ REC_i ] CMD_REC_0
      and
      [ not ((CMD_i) or (REC_i)) ] CMD_REC_1
    )
    and
    [ REC_i ] false
    and
    [ not ((CMD_i) or (REC_i)) ] CMD_REC_0
  )
```

Discussion and perspectives

- Model-based verification techniques:
  - Bug hunting, useful in early stages of the design process
  - Confronted with (very) large models
  - Temporal logics extended with data (XTL, Evaluator 4.0)
  - Machinery for on-the-fly verification (Open/Caesar)

- Perspectives:
  - Parallel and distributed algorithms
    - State space construction
    - BES resolution
  - New applications
    - Analysis of genetic regulatory networks