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

<table>
<thead>
<tr>
<th></th>
<th>State-based</th>
<th>Action-based</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linear-time</strong></td>
<td>LTL (SPIN tool)</td>
<td>TLA (TLA+ tool)</td>
</tr>
<tr>
<td>(properties about execution sequences)</td>
<td>linear mu-calculus</td>
<td>action-based LTL (LTSA tool)</td>
</tr>
<tr>
<td><strong>Branching-time</strong></td>
<td>CTL (nuSMV tool)</td>
<td>ACTL (JACK tool)</td>
</tr>
<tr>
<td>(properties about execution trees)</td>
<td>CTL*</td>
<td>ACTL*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>modal mu-calculus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(CWB, Concurrency Factory, CADP tools)</td>
</tr>
</tbody>
</table>
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 $M_2$.

$L(M_1) = L(M_2) = \{ \text{money.coffee, money.tea} \}$
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)
- $T \subseteq S \times A \times S$: transition relation
- $s_0 \in S$: initial state

Interpretation of a formula $\varphi$ on $M$:

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

($[[\varphi]]$ defined inductively on the structure of $\varphi$)

An LTS $M$ satisfies a TL formula $\varphi$ ($M \models \varphi$) iff its initial state satisfies $\varphi$:

$$M \models \varphi \iff s_0 \models \varphi \iff s_0 \in [[\varphi]]$$
Running example: mutual exclusion with a semaphore

Description using communicating automata
LTS model
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)

\[ \alpha ::= a \quad \text{atomic proposition \,(a \in A)} \]
\[ | \quad tt \quad \text{constant “true”} \]
\[ | \quad ff \quad \text{constant “false”} \]
\[ | \quad \alpha_1 \lor \alpha_2 \quad \text{disjunction} \]
\[ | \quad \alpha_1 \land \alpha_2 \quad \text{conjunction} \]
\[ | \quad \neg \alpha_1 \quad \text{negation} \]
\[ | \quad \alpha_1 \implies \alpha_2 \quad \text{implication \, (\neg \alpha_1 \lor \alpha_2)} \]
\[ | \quad \alpha_1 \iff \alpha_2 \quad \text{equivalence \,(\alpha_1 \implies \alpha_2 \land \alpha_1 \implies \alpha_2)} \]
Action predicates
(semantics)

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

- $[[ a ]] = \{ a \}$
- $[[ \text{tt} ]] = A$
- $[[ \text{ff} ]] = \emptyset$
- $[[ \alpha_1 \lor \alpha_2 ]] = [[ \alpha_1 ]] \cup [[ \alpha_2 ]]$
- $[[ \alpha_1 \land \alpha_2 ]] = [[ \alpha_1 ]] \cap [[ \alpha_2 ]]$
- $[[ \neg \alpha_1 ]] = A \setminus [[ \alpha_1 ]]$
- $[[ \alpha_1 \Rightarrow \alpha_2 ]] = (A \setminus [[ \alpha_1 ]]) \cup [[ \alpha_2 ]]$
- $[[ \alpha_1 \Leftrightarrow \alpha_2 ]] = ((A \setminus [[ \alpha_1 ]]) \cup [[ \alpha_2 ]]) 
  \cap ((A \setminus [[ \alpha_2 ]]) \cup [[ \alpha_1 ]])$
Examples

\[ A = \{ \text{NCS}_0, \text{NCS}_1, \text{CS}_0, \text{CS}_1, \text{REQ}_0, \text{REQ}_1, \text{REL}_0, \text{REL}_1 \} \]

- \[ [[ \text{tt} ]] = \{ \text{NCS}_0, \text{NCS}_1, \text{CS}_0, \text{CS}_1, \text{REQ}_0, \text{REQ}_1, \text{REL}_0, \text{REL}_1 \} \]
- \[ [[ \text{ff} ]] = \emptyset \]
- \[ [[ \text{NCS}_0 ]] = \{ \text{NCS}_0 \} \]
- \[ [[ \neg \text{NCS}_0 ]] = \{ \text{NCS}_1, \text{CS}_0, \text{CS}_1, \text{REQ}_0, \text{REQ}_1, \text{REL}_0, \text{REL}_1 \} \]
- \[ [[ \text{NCS}_0 \land \neg \text{NCS}_1 ]] = \{ \text{NCS}_0 \} = [[ \text{NCS}_0 ]] \]
- \[ [[ \text{NCS}_0 \lor \text{NCS}_1 ]] = \{ \text{NCS}_0, \text{NCS}_1 \} \]
- \[ [[ (\text{NCS}_0 \lor \text{NCS}_1) \land (\text{NCS}_0 \lor \text{REQ}_0) ]] = \{ \text{NCS}_0 \} \]
- \[ [[ \text{NCS}_0 \land \text{NCS}_1 ]] = \emptyset = [[ \text{ff} ]] \]
- \[ [[ \text{NCS}_0 \lor \neg \text{NCS}_0 ]] = \{ \text{NCS}_0, \text{NCS}_1, \text{CS}_0, \text{CS}_1, \text{REQ}_0, \text{REQ}_1, \text{REL}_0, \text{REL}_1 \} = [[ \text{tt} ]] \]
HML logic
(syntax)

φ ::= tt constant “true”
| ff constant “false”
| φ₁ ∨ φ₂ disjunction
| φ₁ ∧ φ₂ conjunction
| ¬φ₁ negation
| ⟨ α ⟩ φ₁ possibility
| [ α ] φ₁ necessity

Duality: [ α ] φ = ¬⟨ α ⟩ ¬φ
HML logic
(semantics)

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

- $[[ \text{tt} ]] = S$
- $[[ \text{ff} ]] = \emptyset$
- $[[ \varphi_1 \lor \varphi_2 ]] = [[ \varphi_1 ]] \cup [[ \varphi_2 ]]
- $[[ \varphi_1 \land \varphi_2 ]] = [[ \varphi_1 ]] \cap [[ \varphi_2 ]]
- $[[ \neg \varphi_1 ]] = S \setminus [[ \varphi_1 ]]
- $[[ \langle \alpha \rangle \varphi_1 ]] = \{ s \in S \mid \exists (s, a, s') \in T . a \in [[ \alpha ]] \land s' \in [[ \varphi_1 ]] \}$
- $[[ [ \alpha ] \varphi_1 ]] = \{ s \in S \mid \forall (s, a, s') \in T . a \in [[ \alpha ]] \Rightarrow s' \in [[ \varphi_1 ]] \}$
Example (1/4)

Deadlock freedom: \( \langle \text{tt} \rangle \text{tt} \)
Example (2/4)

Possible execution of a set of actions: \( \langle CS_0 \lor CS_1 \rangle tt \)
Example (3/4)

Forbidden execution of a set of actions: \[ \text{NCS}_0 \lor \text{NCS}_1 \] 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:**
- \( \langle \alpha \rangle ff = \langle ff \rangle \varphi = ff \)
- \([ \alpha ] tt = [ ff ] \varphi = tt \)

**Distributivity of modalities over \( \lor \) and \( \land \):**
- \( \langle \alpha \rangle \varphi_1 \lor \langle \alpha \rangle \varphi_2 = \langle \alpha \rangle (\varphi_1 \lor \varphi_2) \)
- \( \langle \alpha_1 \rangle \varphi \lor \langle \alpha_2 \rangle \varphi = \langle \alpha_1 \lor \alpha_2 \rangle \varphi \)
- \([ \alpha ] \varphi_1 \land [ \alpha ] \varphi_2 = [ \alpha ] (\varphi_1 \land \varphi_2) \)
- \([ \alpha_1 ] \varphi \land [ \alpha_2 ] \varphi = [ \alpha_1 \lor \alpha_2 ] \varphi \)

**Monotonicity of modalities over \( \varphi \) and \( \alpha \):**
- \(( \varphi_1 \Rightarrow \varphi_2 ) \Rightarrow (\langle \alpha \rangle \varphi_1 \Rightarrow \langle \alpha \rangle \varphi_2 ) \land ([ \alpha ] \varphi_1 \Rightarrow [ \alpha ] \varphi_2 ) \)
- \(( \alpha_1 \Rightarrow \alpha_2 ) \Rightarrow (\langle \alpha_1 \rangle \varphi \Rightarrow \langle \alpha_2 \rangle \varphi ) \land ([ \alpha_2 ] \varphi \Rightarrow [ \alpha_1 ] \varphi ) \)
Characterization of branching

Modal formula distinguishing between $M_1$ and $M_2$:

$$\varphi = [\text{money}] (\langle \text{coffee} \rangle tt \land \langle \text{tea} \rangle tt)$$

$M_1 \models \varphi$ and $M_2 \not\models \varphi$
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 \( \langle tt \rangle \langle tt \rangle \ldots \langle tt \rangle \langle a \rangle tt \))
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)

\( \varphi ::= \) \begin{align*}
& \texttt{tt} \mid \texttt{ff} & \text{boolean constants} \\
& | \varphi_1 \lor \varphi_2 \mid \neg \varphi_1 & \text{connectors} \\
& | E [ \varphi_{1\alpha_1} \cup \varphi_2 ] & \text{potentiality 1} \\
& | E [ \varphi_{1\alpha_1} \cup_{\alpha_2} \varphi_2 ] & \text{potentiality 2} \\
& | A [ \varphi_{1\alpha_1} \cup \varphi_2 ] & \text{inevitability 1} \\
& | A [ \varphi_{1\alpha_1} \cup_{\alpha_2} \varphi_2 ] & \text{inevitability 2}
\end{align*}
ACTL logic
(derived operators)

- $\text{EF}_\alpha \varphi = E\left[\text{tt}_\alpha U \varphi\right]$  \hspace{1cm} \text{basic potentiality}
- $\text{AF}_\alpha \varphi = A\left[\text{tt}_\alpha U \varphi\right]$  \hspace{1cm} \text{basic inevitability}

- $\text{AG}_\alpha \varphi = \neg \text{EF}_\alpha \neg \varphi$  \hspace{1cm} \text{invariance}
- $\text{EG}_\alpha \varphi = \neg \text{AF}_\alpha \neg \varphi$  \hspace{1cm} \text{trajectory}

- $\langle \alpha \rangle \varphi = E\left[\text{tt}_{ff} U_\alpha \varphi\right]$  \hspace{1cm} \text{possibility}
- $[\alpha] \varphi = \neg \langle \alpha \rangle \neg \varphi$  \hspace{1cm} \text{necessity}

\textbf{dualities}
ACTL logic
(semantics - potentiality operators)

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

- $[[ E [ \varphi_1 \alpha \cup \varphi_2 ] ]] = \{ s \in S \mid \exists s (=s_0) \rightarrow a^0 s_1 \rightarrow a^1 s_2 \rightarrow \ldots. \exists k \geq 0. \ \forall 0 \leq i < k. (s_i \in [[ \varphi_1 ]] \land a_i \in [[ \alpha \lor \tau ]] \} \land s_k \in [[ \varphi_2 ]] \}$

- $[[ E [ \varphi_1 \alpha_1 \cup \alpha_2 \varphi_2 ] ]] = \{ s \in S \mid \forall s (=s_0) \rightarrow a^0 s_1 \rightarrow a^1 s_2 \rightarrow \ldots. \exists k \geq 0. \ \forall 0 \leq i < k. (s_i \in [[ \varphi_1 ]] \land a_i \in [[ \alpha_1 \lor \tau ]] \land s_k \in [[ \varphi_1 ]] \land a_k \in [[ \alpha_2 ]] \land s_{k+1} \in [[ \varphi_2 ]] \}$
ACTL logic
(semantics - inevitability operators)

[[ A [ \varphi_1 \alpha U \varphi_2 ] ]]:

[[ A [ \varphi_1 \alpha_1 \cup_{\alpha_2} \varphi_2 ] ]]:
Example (1/4)

Potential reachability: \( \text{EF}_{\neg \text{REL}_1} \langle \text{CS}_0 \rangle \text{tt} \)
Example (2/4)

Inevitable reachability: \( AF_{\neg (REL_0 \lor REL_1)} \langle CS_0 \lor CS_1 \rangle tt \)
Example (3/4)

Invariance: \( AG_{\neg (NCS_0 \lor NCS_1)} \langle NCS_0 \lor NCS_1 \rangle tt \)
Example (4/4)

Trajectory: $EG_{\neg CS_0} [ CS_0 ] ff$
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 $\Rightarrow$ inevitable reachability but the converse does not hold:

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

\[ s \models AF_{tt} \langle a \rangle tt \]
\[ s \not\models A [ tt_{tt} U_a tt ] \]
Safety properties

Informally, safety properties specify that “something bad never happens” during the execution of the system.

One way of expressing safety properties:

- Mutual exclusion:
  \[ \neg \langle \text{CS}_0 \rangle \text{EF}_{\neg \text{REL}_0} \langle \text{CS}_1 \rangle \text{tt} \]
  \[= [ \text{CS}_0 ] \text{AG}_{\neg \text{REL}_0} [ \text{CS}_1 ] \text{ff} \]

In ACTL, forbidding a sequence is expressed by combining the \([\alpha] \varphi\) and \(\text{AG}_{\alpha} \varphi\) 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:
  \[ \langle \text{NCS}_0 \rangle \text{EF}_{tt} \langle \text{REQ}_0 \rangle \text{EF}_{tt} \langle \text{REL}_0 \rangle tt \]

- Inevitable access to the critical section:
  \[ \text{A} [ tt_{tt} \cup_{\text{CS}_0} tt ] \]

In ACTL, the existence of a sequence is expressed by combining the \( \langle \alpha \rangle \varphi \) and \( \text{EF}_\alpha \varphi \) 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 $a.b.a.b \ldots a.b$ 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)

\[ \beta ::= \alpha \quad \text{one-step sequence} \]

<table>
<thead>
<tr>
<th>\beta _1 _ \beta _2 \quad \text{concatenation}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\beta _1 _ \beta _2 \quad \text{choice}</td>
</tr>
<tr>
<td>\beta _1 ^* \quad \text{iteration} (\geq 0 \ \text{times})</td>
</tr>
<tr>
<td>\beta _1 ^+ \quad \text{iteration} (\geq 1 \ \text{times})</td>
</tr>
</tbody>
</table>

- Some identities:
  \[ \text{nil} = \text{ff} ^* \]
  \[ \beta ^+ = \beta ^* \]
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)
- $[[\beta_1 \cdot \beta_2]] = [[\beta_1]] \circ [[\beta_2]]$ (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: $NCS_0 \lor CS_0$
Example (2/3)

Alternative sequences: \((REQ_0 \cdot CS_0) \mid (REQ_1 \cdot CS_1)\)
Example (3/3)

Sequences with repetition: \( NCS_0 \cdot (\neg NCS_1)^* \cdot CS_0 \)
PDL logic
(syntax)

\[ \phi ::= \text{tt} \mid \text{ff} \quad \text{boolean constants} \]
\[ \mid \phi_1 \lor \phi_2 \quad \text{disjunction} \]
\[ \mid \phi_1 \land \phi_2 \quad \text{conjunction} \]
\[ \mid \neg \phi_1 \quad \text{negation} \]
\[ \mid \langle \beta \rangle \phi_1 \quad \text{possibility} \]
\[ \mid [\beta] \phi_1 \quad \text{necessity} \]

\textbf{Duality:} \quad [\beta] \phi = \neg \langle \beta \rangle \neg \phi
Let $M = (S, A, T, s_0)$. Interpretation $[[\varphi ]] \subseteq S$:

- $[[ tt ]] = S$
- $[[ ff ]] = \emptyset$
- $[[ \varphi_1 \lor \varphi_2 ]] = [[ \varphi_1 ]] \cup [[ \varphi_2 ]]$
- $[[ \varphi_1 \land \varphi_2 ]] = [[ \varphi_1 ]] \cap [[ \varphi_2 ]]$
- $[[ \neg \varphi_1 ]] = S \setminus [[ \varphi_1 ]]$
- $[[ \langle \beta \rangle \varphi_1 ]] = \{ s \in S \mid \exists s' \in S . \ (s, s') \in [[ \beta ]] \land 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: \( \langle NCS_0 \cdot tt^* \cdot CS_0 \rangle tt \)
Example (2/2)

Mutual exclusion: [ CS₀ . (¬REL₀)* . CS₁ ] ff
Some identities

Distributivity of regular operators over $\langle \rangle$ and $[\hspace{.2em}]$:

- $\langle \beta_1 \cdot \beta_2 \rangle \varphi = \langle \beta_1 \rangle \langle \beta_2 \rangle \varphi$
- $\langle \beta_1 | \beta_2 \rangle \varphi = \langle \beta_1 \rangle \varphi \lor \langle \beta_2 \rangle \varphi$
- $\langle \beta^* \rangle \varphi = \varphi \lor \langle \beta \rangle \langle \beta^* \rangle \varphi$
- $[\beta_1 \cdot \beta_2] \varphi = [\beta_1] [\beta_2] \varphi$
- $[\beta_1 | \beta_2] \varphi = [\beta_1] \varphi \land [\beta_2] \varphi$
- $[\beta^*] \varphi = \varphi \land [\beta] [\beta^*] \varphi$

Potentiality and invariance operators of ACTL:

- $\text{EF}_\alpha \varphi = \langle \alpha^* \rangle \varphi$
- $\text{AG}_\alpha \varphi = [\alpha^*] \varphi$
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

$$s \not\models A \left[ tt_{tt} \cup a tt \right]$$

*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:

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

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: \([ (\neg CS_0)^* ] \langle tt^* \cdot CS_0 \rangle tt\)
Regular logics
(summary)

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

More intuitive description of safety properties:
- Mutual exclusion:
  \[[ \text{CS}_0 \] \ AG\neg_{\text{REL}} \ [ \text{CS}_1 ] \ \text{ff} \ = \ [ \text{CS}_0 \cdot (\neg_{\text{REL}})^* \cdot \text{CS}_1 ] \ \text{ff}\]
  (in ACTL)
  (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** ($\mu$)
  
  “recursive function” defined over the LTS: 
  finite execution trees going out of a state

- **Maximal fixed point** ($\nu$)
  
  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)

\( \varphi ::= \ \text{tt} \ | \ \text{ff} \)  \hspace{1cm} \text{boolean constants}

\( | \ \varphi_1 \lor \varphi_2 \ | \ \neg \varphi_1 \) \hspace{1cm} \text{connectors}

\( | \ \langle \alpha \rangle \varphi_1 \) \hspace{1cm} \text{possibility}

\( | \ [ \alpha ] \varphi_1 \) \hspace{1cm} \text{necessity}

\( | \ \chi \) \hspace{1cm} \text{propositional variable}

\( | \ \mu \chi . \varphi_1 \) \hspace{1cm} \text{minimal fixed point}

\( | \ \nu \chi . \varphi_1 \) \hspace{1cm} \text{maximal fixed point}

\textbf{Duality:} \hspace{1cm} \nu \chi . \varphi = \neg \mu \chi . \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 $\varphi$ falls in the scope of an even number of negations
    \[
    \mu X . \langle a \rangle X \lor \neg \langle b \rangle X
    \]

- **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 $\varphi$ is closed
    \[
    \mu X . \langle a \rangle \nu Y . ([b] Y \land [c] X)
    \]
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

\[
[[ \varphi ]] \subseteq S:
\]

\[
[[ X ]] \rho = \rho (X)
\]

\[
[[ \mu X \cdot \varphi ]] \rho = \bigcup_{k \geq 0} \Phi^k_\rho (\emptyset)
\]

\[
[[ \nu X \cdot \varphi ]] \rho = \bigcap_{k \geq 0} \Phi^k_\rho (S)
\]

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 $a$):

$$\mu X . \langle a \rangle tt \lor \langle tt \rangle X$$

Associated functional:

$$\Phi (U) = \left[ \left[ \langle a \rangle tt \lor \langle tt \rangle X \right] \right] \left[ U / X \right]$$

Evaluation on an LTS:

---

VTSA’08 - Max Planck Institute, Saarbrücken
Example

Potential reachability: $\mu X . \langle CS_0 \rangle tt \lor \langle \neg (REL_1 \lor REL_0) \rangle X$
Maximal fixed point

- Infinite repetition of an action $a$ (existence of a cycle containing only transitions labeled by $a$):
  $$ \nu X \cdot \langle a \rangle X $$

- Associated functional:
  $$ \Phi(U) = [[[\langle a \rangle X]]] [U / X] $$

- Evaluation on an LTS:
Example

Infinite repetition: \( \forall X . \langle NCS_1 \lor REQ_1 \lor CS_1 \lor REL_1 \rangle X \)
Exercise

Evaluate the formula: \( \mu X . \langle CS_0 \rangle tt \lor ([ NCS_0 ] ff \land \langle tt \rangle X ) \)
Some identities

Description of (some) ACTL operators:

- \( E[\varphi_{\alpha_1} U \alpha_2 \varphi_2] = \mu X . \varphi_1 \land (\langle \alpha_2 \rangle \varphi_2 \lor \langle \alpha_1 \rangle X) \)

- \( A[\varphi_{\alpha_1} U \alpha_2 \varphi_2] = \mu X . \varphi_1 \land \langle tt \rangle tt \land \lnot(\alpha_1 \lor \alpha_2) \land \lnot\alpha_1 \land \alpha_2 \land \varphi_2 \land \alpha_1 \land \alpha_2 (\varphi_2 \lor X) \)

- \( EF_{\alpha} \varphi = \mu X . \varphi \lor \langle \alpha \rangle X \)

- \( AF_{\alpha} \varphi = \mu X . \varphi \lor (\langle tt \rangle tt \land \lnot\alpha \land \alpha (\varphi_2 \lor X) \)

Description of the PDL operators:

- \( \langle \beta^* \rangle \varphi = \mu X . \varphi \lor \langle \beta \rangle X \)

- \( [\beta^*] \varphi = v X . \varphi \land [\beta] X \)
Inevitable reachability

Inevitable reachability of an action $a$:

\[
\text{access (}a\text{)} = \text{AF}_\text{tt} \langle a \rangle \text{tt} = \\
\mu X . \langle a \rangle \text{tt} \lor (\langle \text{tt} \rangle \text{tt} \land [ \text{tt } ] X )
\]

Associated functional:

\[
\Phi (U) = [[ \langle a \rangle \text{tt} \lor (\langle \text{tt} \rangle \text{tt} \land [ \text{tt } ] X ) ]] [ U / X ]
\]

Evaluation on an LTS:

\[
\Phi^2 (\emptyset) \quad \Phi (\emptyset)
\]
Inevitable execution

Inevitable execution of an action $a$:

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

Associated functional:

$$\Phi \ (U) = [[ \langle tt \rangle tt \land [ \lnot a ] X ]] \ [ U / X ]$$

Evaluation on an LTS:
Example

Inevitable execution: $\mu X . \langle t t \rangle t t \land [ \neg CS_0 ] X$
Fair execution

Fair execution of an action \( a \):

\[
\text{fair } (a) = \left[ (\neg a)^* \right] \langle tt^*. a \rangle tt
= \nu X . \langle tt^*. a \rangle tt \land [ \neg a ] X
\]

Associated functional:

\[
\Phi (U) = [[ \langle tt^*. a \rangle tt \land [ \neg a ] X ]] \ [ U / X ]
\]

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:

\[ \langle (a \cdot b^*)^* \cdot c \rangle tt = \]
\[ \mu X . \langle c \rangle tt \vee \langle a \rangle \mu Y . (X \vee \langle b \rangle Y) \]

**Alternation depth** of a formula: degree of mutual recursion between \( \mu \) and \( \nu \) fixed points

Example of alternation depth 2 formula:

\[ \nu X . \langle a^* \cdot b \rangle X = \nu X . \mu Y . \langle b \rangle X \vee \langle a \rangle Y \]
Some verification tools
(for action-based logics)

- **CWB** (Edinburgh)
  and

- **Concurrency Factory** (State University of New York)
  - Modal $\mu$-calculus (fixed point operators)

- **JACK** (University of Pisa, Italy)
  - $\mu$-ACTL (modal $\mu$-calculus combined with ACTL)

- **CADP / Evaluator 3.x** (INRIA Rhône-Alpes / VASY)
  - Regular alternation-free $\mu$-calculus (PDL modalities and fixed point operators)
Extensions of $\mu$-calculus with data

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

  Extended temporal logics are needed in practice

Several $\mu$-calculus extensions with data:
- For polyadic pi-calculus [Dam-94]
- For symbolic transition systems [Rathke-Hennessy-96]
- For $\mu$CRL [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):

  $\langle b \rangle \langle b \rangle \langle b \rangle \langle a \rangle \text{tt}$  
  or  
  $\langle b \{3\} \cdot a \rangle \text{tt}$  

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

  RECV 1  
  $\downarrow$  
  ACK 1  
  $\downarrow$  
  RECV 2  
  $\downarrow$  
  ACK 2

value extraction and propagation
Model Checking Language

Based on EVALUATOR 3.5 input language
- standard \( \mu \)-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:**

\(< \{\text{SEND } ?\text{msg: Nat}\} > < \{\text{RECV } !\text{msg}\} > \text{true}\)

**Necessity:**

\([ \{\text{RECV } ?\text{msg: Nat}\} ] (\text{msg < 6})\)
Parameterized fixed points

(basic) syntax:

\[
\mu X \ (y:T := E) \ . \ P
\]

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

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

\[
\nu \{ \text{LEVEL ?l:Nat where } l > 10 \} \\
\nu X (c:Nat := 15) . \\
\nu \{ \text{not ALARM } \} (c > 0 \text{ and } X (c - 1))
\]
Quantifiers

- **Existential quantifier:**
  
  \[
  \exists x : T \text{ among } \{ E_1 \ldots E_2 \} . P
  \]

- **Universal quantifier:**
  
  \[
  \forall x : T \text{ among } \{ E_1 \ldots E_2 \} . P
  \]

\[\Rightarrow \text{ shorthands for large disjunctions and conjunctions}\]
Example

Broadcast of messages:

\[
\text{forall } \text{msg:} \text{Nat among } \{1 \ldots 10\}. \\
\mu \text{X} . (\langle \{\text{SEND} \text{!!msg}\} \rangle > \text{true} \text{ or } \langle \text{true} \rangle > \text{X})
\]
Counting operators
(regular formulas)

\[
\begin{align*}
R \{ E \} & \quad \text{repetition } E \text{ times} \\
R \{ E_1 \ldots \} & \quad \text{repetition at least } E_1 \text{ times} \\
R \{ E_1 \ldots E_2 \} & \quad \text{repetition between } E_1 \text{ and } E_2 \text{ times}
\end{align*}
\]

Some identities:

\[
\begin{align*}
nil & = \text{false} * \\
R * & = R \{ 0 \ldots \} \\
R + & = R \{ 1 \ldots \} \\
R + & = R \{ E \} = R \{ E \ldots E \}
\end{align*}
\]
Example
(action counting revisited)

Formulation using counting operators:

\[
[ \{ \text{LEVEL } ?l: \text{Nat where } l > 10 \} . (\text{not ALARM}) \{ 16 \} ] \text{ false}
\]
Example
(safety of a n-place buffer)

Formulation using extended regular operators:

\[
[ \text{true}^* \cdot ((\text{not \ OUTPUT})^* \cdot \text{INPUT}) \{ n + 1 \} ] \text{ false}
\]

Formulation using parameterized fixed points:

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

n+1 INPUTs without OUTPUTs
Looping operator (from PDL-delta)

\[ \Delta R \] operator added to PDL to specify infinite behaviours [Streett-82]

MCL syntax: < R > @

Examples:
- process overtaking
  \[ [\text{REQ}_0] < (\text{not GET}_0)^* \cdot \text{REQ}_1 \cdot (\text{not GET}_0)^* \cdot \text{GET}_1 > @ \]
- Büchi acceptance condition
  \[ < \text{true}^* \cdot \text{if } P_{\text{accepting}} \text{ then true end if } > @ \]

\[ \Rightarrow \text{allows to encode LTL model checking} \]
Expressiveness
(summary)

CTL* ⊆ PDL-Δ ⊆ 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 $M_2$

$$M_1 \approx M_2 \iff \forall \varphi \in L . (M_1 \models \varphi \iff M_2 \models \varphi)$$

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

\[\langle \langle \varphi \rangle \rangle = \langle \tau^* \rangle \varphi\]
\[\langle \langle a \rangle \rangle \varphi = \langle \tau^* \cdot a \cdot \tau^* \rangle \varphi\]
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
- Language technology

- Model (state space)

- Desired properties
- Verification tool
- Model technology

- True / false + diagnostic
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 $B = (x, M_1, ..., M_n)$, where

- $x \in X$ : main boolean variable
- $M_i = \{ x_j = \sigma_i \ op_j \ X_j \}_{j \in [1, m_i]}$ : equation blocks
  - $\sigma_i \in \{ \mu, \nu \}$ : fixed point sign of block $i$
  - $op_j \in \{ \lor, \land \}$ : operator of equation $j$
  - $X_j \subseteq X$ : variables in the right-hand side of equation $j$
  - $F = \lor\emptyset$ (empty disjunction), $T = \land\emptyset$ (empty conjunction)
  - $x_j$ depends upon $x_k$ iff $x_k \in X_j$
  - $M_i$ 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 $M_{i+1}$ ... $M_n$
Example

\[
\begin{align*}
M_1 & \quad \left\{ \begin{array}{l}
x_1 = \mu \ x_2 \lor x_3 \\
x_2 = \mu \ x_3 \lor x_4 \\
x_3 = \mu \ x_2 \land x_7 \\
\end{array} \right. \\
M_2 & \quad \left\{ \begin{array}{l}
x_4 = \mu \ x_5 \lor x_6 \\
x_5 = \mu \ x_8 \lor x_9 \\
x_6 = \mu \ F \\
\end{array} \right. \\
M_3 & \quad \left\{ \begin{array}{l}
x_7 = v \ x_8 \land x_9 \\
x_8 = v \ T \\
x_9 = v \ F \\
\end{array} \right.
\end{align*}
\]
Particular blocks

- **Acyclic** block:
  - No cyclic dependencies between variables of the block

- **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 \( \delta : X \rightarrow \text{Bool} \)
- Semantics of a boolean formula:
  - \( \llbracket \text{op} \{ x_1, \ldots, x_p \} \rrbracket \ \delta = \text{op} (\delta(x_1), \ldots, \delta(x_p)) \)
- Semantics of a block:
  - \( \llbracket \{ x_j = \sigma \ \text{op}_j X_j \} \_{j \in [1, m]} \rrbracket \ \delta = \sigma \Phi_\delta \)
  - \( \Phi_\delta : \text{Bool}^m \rightarrow \text{Bool}^m \)
  - \( \Phi_\delta(b_1, \ldots, b_m) = (\llbracket \text{op}_j X_j \rrbracket \ (\delta \oplus [b_1/x_1, \ldots, b_m/x_m]))_{j \in [1, m]} \)
- Semantics of a BES:
  - \( \llbracket (x, M_1, \ldots, M_n) \rrbracket = \delta_1(x) \)
  - \( \delta_n = \llbracket M_n \rrbracket [] \) (\( M_n \) closed)
  - \( \delta_i = (\llbracket M_i \rrbracket \ \delta_{i+1}) \oplus \delta_{i+1} \) (\( M_i \) depends upon \( M_{i+1} \ldots M_n \))
Local resolution

Alternation-free BES $B = (x, M_1, ..., M_n)$

Primitives: 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 $R_1 (x) \rightarrow ... \rightarrow R_n (x_k)$ 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

\[
\begin{align*}
X_1 &= \mu X_2 \lor X_3 \\
X_2 &= \mu X_3 \lor X_4 \\
X_3 &= \mu X_2 \land X_7 \\
X_4 &= \mu X_5 \lor X_6 \\
X_5 &= \mu X_8 \lor X_9 \\
X_6 &= \mu F \\
X_7 &= \nu X_8 \land X_9 \\
X_8 &= \nu T \\
X_9 &= \nu F
\end{align*}
\]
Local resolution algorithms

- Representation of blocks as *boolean graphs* [Andersen-94]

- To a block $M = \{ x_j =_\mu op_j X_j \}_{j \in [1, m]}$ we associate the boolean graph $G = (V, E, L, \mu)$, where:
  - $V = \{ x_1, \ldots, x_m \}$: set of vertices (variables)
  - $E = \{ (x_i, x_j) \mid x_j \in X_i \}$: set of edges (dependencies)
  - $L : V \to \{ \lor, \land \}$, $L(x_j) = op_j$: 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 ($\mu$-block)

\[
\begin{align*}
    x_1 &= \mu x_2 \lor x_3 \\
    x_2 &= \mu F \\
    x_3 &= \mu x_4 \lor x_5 \\
    x_4 &= \mu T \\
    x_5 &= \mu x_1
\end{align*}
\]

boolean graph

\[\text{\(\bigtriangleup\)} : \lor\text{-variables} \quad \text{\(\bigtriangleup\text{\(\bigtriangleup\)}} : \land\text{-variables} \]
Three effectiveness criteria

[Mateescu-06]

For each resolution routine $R$:

A. The worst-case complexity of a call $R(x)$ must be $O(|V|+|E|)$

$\Rightarrow$ linear-time complexity for the overall BES resolution

B. While executing $R(x)$, every variable explored must be « linked » to $x$ via unstable variables

$\Rightarrow$ graph exploration limited to “useful” variables

C. After termination of $R(x)$, all variables explored must be stable

$\Rightarrow$ 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 $\mu$-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 $O(|V|)$
**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
  - Equivalence checker
    - True / False
    - Diagnostic
- Description of service
  - Compiler
    - LTS 2
Strong equivalence

\[ M_1 = (Q_1, A, T_1, q_{01}), \quad M_2 = (Q_2, A, T_2, q_{02}) \]
\[ \approx \subseteq Q_1 \times Q_2 \text{ is the maximal relation s.t. } p \approx q \text{ iff } \]
\[ \forall a \in A. \forall p \rightarrow a p' \in T_1. \exists q \rightarrow a q' \in T_2. \quad p' \approx q' \]
and
\[ \forall a \in A. \forall q \rightarrow a q' \in T_2. \exists p \rightarrow a p' \in T_1. \quad p' \approx q' \]
\[ M_1 \approx M_2 \text{ iff } q_{01} \approx q_{02} \]
Translation to a BES

- Principle: \( p \approx q \) iff \( X_{p,q} \) is true
- General BES:
  \[
  X_{p,q} = \vee (p \rightarrow a\, p' \vee q \rightarrow a\, q' \wedge X_{p',q'})
  \]
  \[
  (q \rightarrow a\, q' \vee p \rightarrow a\, p' \wedge X_{p',q'})
  \]
- Simple BES:
  \[
  X_{p,q} = \vee (p \rightarrow a\, p' \vee q \rightarrow a\, q' \wedge X_{p',q'})
  \]
  \[
  Y_{a,p',q} = \vee (q \rightarrow a\, q' \wedge X_{p',q'})
  \]
  \[
  Z_{a,p,q'} = \vee (p \rightarrow a\, p' \wedge X_{p',q'})
  \]
  \[
  p \leq q \quad \text{(preorder)}
  \]
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:**

\[
X_{p,q} = \bigvee \left( \bigwedge p \rightarrow^{*}.a \ p' \ \bigvee q \rightarrow^{*}.a \ q' \ X_{p',q'} \right)
\]

**Safety equivalence:**

\[
X_{p,q} = \bigvee \ Y_{p,q} \wedge Y_{q,p}
\]
\[
Y_{p,q} = \bigvee \ \bigwedge p \rightarrow^{*}.a \ p' \ \bigvee q \rightarrow^{*}.a \ q' \ Y_{p',q'}
\]
Observational and branching equivalences

Observational equivalence:

\[
\begin{align*}
X_{p,q} = \nu & \left( \land p \rightarrow \tau p' \lor q \rightarrow \tau^* q' X_{p',q'} \right) \land \\
& \left( \land p \rightarrow a p' \lor q \rightarrow \tau^*.a.\tau^* q' X_{p',q'} \right) \\
& \land \left( \land q \rightarrow \tau q' \lor p \rightarrow \tau^* p' X_{p',q'} \right) \land \\
& \left( \land q \rightarrow a q' \lor p \rightarrow \tau^*.a.\tau^* p' X_{p',q'} \right)
\end{align*}
\]

Branching equivalence:

\[
\begin{align*}
X_{p,q} = \nu & \land p \rightarrow b p'((b=\tau \land X_{p',q}) \lor \land q \rightarrow \tau^* q' \rightarrow b q''(X_{p,q'} \land X_{p',q''})) \\
& \land \land q \rightarrow b q'((b=\tau \land X_{p,q'}) \lor \land p \rightarrow \tau^* p' \rightarrow b p''(X_{p',q} \land X_{p'',q'}))
\end{align*}
\]
Example
(coffee machine)
Equivalence checking (time)

19 LTSs of the VLTS benchmark suite
www.inrialpes.fr/vasy/cadp/resources/benchmark_bcg.html
Equivalence checking (memory)

- eq. de branchement
- eq. observationnelle
- eq. tau.a

Taille du STE (nombre de transitions) vs. mémoire (Ko)
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 ($\tau$-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 $\tau$-free
  - Algorithm A4 (memory ↓)
Model checking
(principle)

description of system

compiler

LTS

properties

model checker

ture / false + diagnostic
On-the-fly model checking in CADP
(Evaluator 3.x)

LTS

Model checker

translation

BES

resolution

yes / no + diagnostic

On-the-fly activities

formula
Translation to Boolean Equation Systems

- LTS
- Translation to PDLR
  - PDLR spec
  - Translation to HMLR
    - HMLR spec
    - Translation to BESs
      - BES

- Formula
Translation to PDL with recursion

State formula (expanded):
\[
\nu Y_0 \cdot [\text{true}^* \cdot \text{SEND}] \\
\mu Y_1 \cdot \langle \text{true} \rangle \text{true and } [\text{not RECV}] Y_1
\]

PDLR specification \cite{Mateescu-Sighireanu-03}:
\[
Y_0 =_{\nu} [\text{true}^* \cdot \text{SEND}] Y_1 \\
Y_1 =_{\mu} \langle \text{true} \rangle \text{true and } [\text{not RECV}] Y_1
\]
Simplification

PDLR specification:

\[ Y_0 = \text{nu} [ \text{true}^* \cdot \text{SEND} ] Y_1 \]

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

Simple PDLR specification:

\[ Y_0 = \text{nu} [ \text{true}^* \cdot \text{SEND} ] Y_1 \]

\[ Y_1 = \text{mu} \]
\[ Y_2 = \text{mu} \langle \text{true} \rangle \text{true} \]
\[ Y_3 = \text{mu} [ \text{not RECV } ] Y_1 \]
Translation to BESs

Boolean variables: \( x_{i,j} \equiv s_j \models Y_j \)

\[
Y_0 = _\text{nu} \ Y_4 \text{ and } Y_5 \\
Y_4 = _\text{nu} \ [ \text{SEND} ] \ Y_1 \\
Y_5 = _\text{nu} \ [ \text{true} ] \ Y_0 \\
Y_1 = _\text{mu} \ Y_2 \text{ and } Y_3 \\
Y_2 = _\text{mu} \ ( \text{true} ) \ \text{true} \\
Y_3 = _\text{mu} \ [ \text{not RECV} ] \ Y_1 \\
\]

\[
X_{0,0} = _\text{v} \ X_{0,4} \land X_{0,5} \\
X_{0,4} = _\text{v} \ X_{1,1} \\
X_{0,5} = _\text{v} \ X_{1,0} \\
X_{1,0} = _\text{v} \ X_{1,4} \land X_{1,5} \\
X_{1,4} = _\text{v} \ \text{true} \\
X_{1,5} = _\text{v} \ X_{2,0} \land X_{3,0} \\
X_{2,0} = _\text{v} \ X_{2,4} \land X_{2,5} \\
X_{2,4} = _\text{v} \ \text{true} \\
X_{2,5} = _\text{v} \ X_{0,0} \\
X_{3,0} = _\text{v} \ X_{3,4} \land X_{3,5} \\
X_{3,4} = _\text{v} \ \text{true} \\
X_{3,5} = _\text{v} \ X_{0,0} \\
X_{1,1} = _\text{\mu} \ X_{1,2} \land X_{1,3} \\
X_{1,2} = _\text{\mu} \ \text{true} \\
X_{1,3} = _\text{\mu} \ X_{2,1} \land X_{3,1} \\
X_{2,1} = _\text{\mu} \ X_{2,2} \land X_{2,3} \\
X_{2,2} = _\text{\mu} \ \text{true} \\
X_{2,3} = _\text{\mu} \ \text{true} \\
X_{3,1} = _\text{\mu} \ X_{3,2} \land X_{3,3} \\
X_{3,2} = _\text{\mu} \ \text{true} \\
X_{3,3} = _\text{\mu} \ X_{0,1} \\
X_{0,1} = _\text{\mu} \ X_{0,2} \land X_{0,3} \\
X_{0,2} = _\text{\mu} \ \text{true} \\
X_{0,3} = _\text{\mu} \ X_{1,1} \\
\]
Local BES resolution with diagnostic

Counterexample

SEND

TIMEOUT

\[ x_{0,0} \rightarrow x_{0,4} \rightarrow x_{0,0} \rightarrow x_{1,0} \rightarrow x_{1,5} \rightarrow x_{2,0} \rightarrow x_{3,0} \rightarrow x_{3,5} \]

\[ x_{1,1} \rightarrow x_{1,2} \rightarrow x_{1,3} \rightarrow x_{2,1} \rightarrow x_{3,1} \rightarrow x_{3,3} \rightarrow x_{0,1} \rightarrow x_{0,3} \]
**Additional operators**

- **Mechanisms for macro-definition (overloaded) and library inclusion**

- **Libraries encoding the operators of CTL and ACTL**

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

    \[
    \text{EU} (\varphi_1, \alpha_1, \alpha_2, \varphi_2) = \mu Y . \langle \alpha_2 \rangle \varphi_2 \text{ or } (\varphi_1 \text{ and } \langle \alpha_1 \rangle Y)
    \]

- **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](http://www.inrialpes.fr/vasy/cadp/resources)
**Disjunctive BES**

*Disjunctive* boolean graph:

- **Potentiality** operator of CTL

\[
E \left[ \varphi_1 \mathbin{U} \varphi_2 \right] = \mu X \cdot \varphi_2 \lor (\varphi_1 \land \langle T \rangle X)
\]

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

\[
\{ X_s = \mu \varphi_{2s} \lor Y_s, \ Y_s = \mu \varphi_{1s} \land Z_s, \ Z_s = \mu \lor_{s \rightarrow s'} X_{s'} \}
\]

- **Possibility** modality of PDL

\[
\langle (a \mid b)^* \cdot c \rangle \text{T}
\]

\[
\{ X = \mu \langle c \rangle \text{T} \lor \langle a \rangle X \lor \langle b \rangle X \}
\]

\[
\{ X_s = \mu (\lor_{s \rightarrow c} s', T) \lor (\lor_{s \rightarrow a} s', X_{s'}), \lor (\lor_{s \rightarrow b} s', X_{s'}) \}
\]

- Algorithm **A3** (memory ↓)
Linear-time model checking
(looping operator of PDL-delta)

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

\[ < R > @ = \nu X . < R > X \]

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
- \( A_{3cyc} \) algorithm [Mateescu-Thivolle-08]
  - Can serve for LTL model checking
  - Allows linear-time handling of repeated invocations

If \( R \) contains \(*\)-operators, the formula is of alternation depth 2.
Model checking of data-based properties (Evaluator 4.0)

Every SEND is followed by a RECV after 2 steps:

\[ [ \text{true}^* \cdot \text{SEND} ] < \text{true} \{ 2 \} \cdot \text{RECV} > \text{true} = \]
\[ \nu X \cdot ( [ \text{SEND} ] \mu Y (c:\text{Nat} := 2) \cdot \]
\[ \text{if } c = 0 \text{ then } < \text{RECV} > \text{true} \]
\[ \text{else } < \text{true} > Y (c - 1) \]
\[ \text{end if} \]

and

\[ [ \text{true} ] X ) \]
Translation into HMLR

\( \text{nu } X \cdot [ \text{SEND } ] \)

and [ true ] X

\{ X =_{\text{nu}} \)

[ SEND ] Y (2)

and

[ true ] X

\}
Translation into BES and resolution

\[
\begin{align*}
\{ & X = \nu \left( \left[ \text{SEND} \right] Y (2) \right)
\quad \text{and}
\quad \left[ \text{true} \right] X \\
\} & \quad \text{for } c = 0 \text{ then } \langle \text{RECV} \rangle \text{ true}
\end{align*}
\]

Principle:

\[
X_s = \langle s \models X \rangle
\quad \text{and}
\quad Y_s (c) = \langle s \models Y (c) \rangle
\]

\[
\begin{align*}
\{ & Y (c: \text{Nat}) = \mu \left( \text{if } c = 0 \text{ then } \langle \text{RECV} \rangle \text{ true} \\
& \quad \text{else } \langle \text{true} \rangle \text{ Y} (c - 1) \text{ end if} \\
\} & \quad \text{for } c = 0 \text{ then } \langle \text{ERROR} \rangle \text{ true}
\end{align*}
\]
Divergence

In presence of data parameters of infinite types, termination of model checking is not guaranteed anymore

(pathological) property:

\[ \mu X (n: \text{Nat} := 0) . < a > X (n + 1) \]

BES:

\[ \{ X_s (n: \text{Nat}) =_{\mu} \text{OR} s \to a s', X_s, (n + 1) \} = \{ X_s (n: \text{Nat}) =_{\mu} X_s (n + 1) \} \]


Conjunctive BES

- **Conjunctive** boolean graph:
  - **Inevitability** operator of CTL
    \[
    A \left[ \varphi_1 \cup \varphi_2 \right] = \mu X . \varphi_2 \lor (\varphi_1 \land \langle T \rangle T \land [ T ] X)
    \]
    \[
    \{ X = \mu \varphi_2 \lor Y , Y = \mu \varphi_1 \land Z \land [ T ] X , Z = \mu \langle T \rangle T \}
    \]
    \[
    \{ X_s = \mu \varphi_{2s} \lor Y_s , Y_s = \mu \varphi_{1s} \land Z_s \land (\langle s \Rightarrow s' \rangle \varphi_{s'}) , Z_s = \mu \lor \varphi_{s'} , T \}
    \]
  - **Necessity** modality of PDL
    \[
    \left[ (a \mid b)^* . c \right] F
    \]
    \[
    \{ X = \mu \left[ c \right] F \land \left[ a \right] X \land \left[ b \right] X \}
    \]
    \[
    \{ X_s = \mu (\langle s \Rightarrow c \rangle s' , F) \land (\langle s \Rightarrow a \rangle s' , X_{s'}) \land (\langle s \Rightarrow b \rangle s' , X_{s'}) \}
    \]

- **Algorithm A4** (memory ↓)
Acyclic BES

*Acyclic* boolean graph:
- *Acyclic* LTS and *guarded* formulas [Mateescu-02]

Handling of CTL (and ACTL) operators:
- $E [\varphi_1 U \varphi_2] = \mu X . \varphi_2 \lor (\varphi_1 \land \langle T \rangle X)$
- $A [\varphi_1 U \varphi_2] = \mu X . \varphi_2 \lor (\varphi_1 \land \langle T \rangle T \land [T] X)$

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

number of boolean operators in the BES

time (sec)
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 $\mu$-calculus formula
  - Algorithms $A_0$ and $A_1$ (diagnostic depth $\downarrow$)

- **Acyclic** boolean graph:
  - Acyclic LTS and guarded formula (CTL, ACTL)
  - Acyclic LTS and $\mu$-calculus formula (via reduction)
  - Algorithm $A_2$ (memory $\downarrow$)

- **Disjunctive/conjunctive** boolean graph:
  - Any LTS and any formula of CTL, ACTL, PDL
  - Algorithm $A_3/A_4$ (memory $\downarrow$)
  - 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

\[ X_{p1,p2} = \nu \land p_1 \rightarrow_b p_3 \]

\[ p_2 \rightarrow_b p_3 \lor \]

\[ \lor p_2 \rightarrow_b p_4, p_3 \rightarrow_{\tau} p_4 X_{p3,p4} \lor \]

\[ (((b = \tau) \land \lor p_3 \rightarrow_{\tau} p_2 X_{p3,p2}) \]
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 \( \tau \)-confluence reduction - time - Erathostene’s sieve)

![Graph showing the effect of \( \tau \)-confluence reduction on model checking time]

- Without \( \tau \)-confluence
- With \( \tau \)-confluence
Model checking using A3/A4
(effect of τ-confluence reduction - memory - Erathostene’s sieve)
Checking branching bisimulation

(effect of \(\tau\)-confluence reduction - time - BRP protocol)
Checking branching bisimulation
(effect of $\tau$-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
Synchronizations on gate ARB:

\[
\text{ARB } ?r_0, \ldots, r_7: \text{Bool} \ [C (r_0, \ldots, r_7, n)] \ ; \ldots
\]

where:
- \( r_0, \ldots, r_7 = \text{values of the electric signals on the bus} \)
- \( n = \text{index of the current device} \)

Two particular cases for guard condition \( C \):
- \( \text{P} (r_0, \ldots, r_7, n): \text{device } n \text{ does not ask the bus} \)
- \( \text{A} (r_0, \ldots, r_7, n): \text{device } n \text{ asks and obtains access to bus} \)
Guard conditions

Predicate \( P (r_0, \ldots, r_7, n) = \neg r_n \)
- \( P (r_0, \ldots, r_7, 0) = \text{not} \ (r_0) \)
- \( P (r_0, \ldots, r_7, 1) = \text{not} \ (r_1) \)

... 
- \( P (r_0, \ldots, r_7, 7) = \text{not} \ (r_7) \)

Predicate \( A (r_0, \ldots, r_7, n) = r_n \land \forall i \in [n+1, 7] . \neg r_i \)
- \( A (r_0, \ldots, r_7, 0) = r_0 \text{ and not} \ (r_1 \text{ or ... or } r_7) \)
- \( A (r_0, \ldots, r_7, 1) = r_1 \text{ and not} \ (r_2 \text{ or ... or } r_7) \)

... 
- \( A (r_0, \ldots, r_7, 7) = r_7 \)
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”

\[
[ \text{true}^* \cdot \text{cmd}_i ] \mathcal{A} [ \text{true}_{\text{true}} \mathcal{U}_{\text{rec}_i} \text{true} ]
\]

- Property fails for \( i < \text{nc} \)
- Counterexample produced by Evaluator 3.5 for \( i = 0 \) and \( \text{nc} = 1 \):
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”

\[
\begin{align*}
\text{[ true}. \{ \text{cmd} \ ?i: \text{Nat where } i < nc \} \] \\
\forall j: \text{Nat among } \{ i + 1 \ldots n - 1 \}. \\
(j <> nc) \implies \\
< (\neg \{ \text{rec } !i \})*. \{ \text{cmd } !j \} . \\
(\neg \{ \text{rec } !i \})*. \{ \text{rec } !j \} > @
\end{align*}
\]
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)”

\[
\text{forall } i : \text{Nat among } \{ 0 \ldots n - 1 \} . \quad \text{nu } Y (c : \text{Nat}:=0) . ( \\
\quad [ \{ \text{cmd } !i \} ] ((c < 8) \text{ and } Y (c + 1)) \text{ and } \\
\quad [ \{ \text{rec } !i \} ] ((c > 0) \text{ and } Y (c - 1)) \text{ and } \\
\quad [ \text{not } (\{ \text{cmd } !i \} \text{ or } \{ \text{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 ] 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
)
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