Walking Back and Forth in Labelled Transition Systems

Radu Mateescu
Inria Grenoble – Rhône-Alpes / Convecs
http://convecs.inria.fr
Outline

Concurrent systems, LTSs, and Boolean graphs

Forward walks
- MCL – a data-based model checking language
- On-the-fly model checking

Backward walks
- XTL – an executable temporal language
- Graph querying

Future walks
Context

Concurrent systems
- Process algebraic languages (LNT)
- Value-passing communication
- Interleaving semantics (LTSs)
- Branching-time world (adequate with bisimulations)

Explicit-state verification
- Enumeration of individual states and transitions
- Forward and backward exploration
- Diagnostic generation

CADP toolbox
http://cadp.inria.fr
Labeled Transition Systems

Two-place FIFO buffer (two cells in sequence)
Stream of 0/1 messages
FORWARD WALKS
Implicit LTSs
(Open/Caesar environment)

- Represented in comprehension (post function)

Open/Caesar:
- Generic API for implicit LTSs
  - Opaque data type for representing states
  - Initial state function
  - Successor function
- Set of libraries for LTS manipulation
  - Hash tables, stacks, caches, Boolean equation systems, ...

Compilers ➔ Open/Caesar API ➔ Analysis tools
Open/Cæsar Architecture

LOTOS
- caesar.open

LNT
- Int.open

FSP
- fsp.open

LTS
- bcg_open

EXP
- exp.open

... SystemC/TLM
- tlm.open

Open/Cæsar API

On-the-fly analyses
- LTS generation
- interactive simulation
- random execution
- on the fly verification
- steady-state simulation
- test generation

Open/Cæsar libraries

Graphite’2014 - Grenoble, April 5, 2014
Boolean Equation Systems

\[
\begin{align*}
\begin{cases}
  x_1 &= \mu x_2 \lor x_3 \\
  x_2 &= \mu x_3 \lor x_4 \\
  x_3 &= \mu x_2 \land x_7
  \end{cases}
\end{align*}
\]

\[
\begin{align*}
\begin{cases}
  x_4 &= \mu x_5 \lor x_6 \\
  x_5 &= \mu x_8 \lor x_9 \\
  x_6 &= \mu F \\
  x_7 &= \lor x_8 \land x_9 \\
  x_8 &= \lor T \\
  x_9 &= \lor F
  \end{cases}
\end{align*}
\]

no cyclic dependencies between equation blocks (alternation-free)
Particular Blocks

**Acyclic:** no cyclic dependencies between variables

Var. $x_i$ disjunctive (conjunctive): $op_i = \lor (op_i = \land)$

**Disjunctive:**
- 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:** dual definition
Local Resolution

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

Primitive: compute a variable $x_j$ of a block $M_i$
- Resolution routine $R_i$ associated to $M_i$
- $R_i(x_j)$ computes the value of $x_j$ in $M_i$
- Bounded call stack $R_1(x) \rightarrow \ldots \rightarrow R_n(x_k)$
- Coroutine-like style: each $R_i$ keeps its context

Advantages:
- On-the-fly BES construction
- Optimization by specializing $R_i$ w.r.t. particular blocks
Example

\[\begin{align*}
X_1 &= \mu X_2 \lor X_3 \\
X_2 &= \mu X_3 \lor X_4 \\
X_3 &= \mu X_2 \land X_7
\end{align*}\]

\[\begin{align*}
X_4 &= \mu X_5 \lor X_6 \\
X_5 &= \mu X_8 \\
X_6 &= \mu F
\end{align*}\]

\[\begin{align*}
X_7 &= \lor X_8 \land X_9 \\
X_8 &= T \\
X_9 &= F
\end{align*}\]
Local Resolution Algorithms

Represent blocks as *boolean graphs* [Andersen-94]

To \( M = \{ x_j = \_\_ op_j X_j \}_{j \in [1, m]} \) associate \( G = (V, E, L, \mu) \)

- \( V = \{ x_1, \ldots, x_m \} \): set of vertices (variables)
- \( E = \{ (x_i, x_j) \mid x_j \in X_i \} \): set of edges (dependencies)
- \( L : V \rightarrow \{ \lor, \land \} \), \( L(x_j) = op_j \): 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 (μ-bloc)

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

\[\text{☩ : } \lor \text{-variables}\]
\[\text{🔺 : } \land \text{-variables}\]
The Caesar_Solve Library

[Mateescu-03,06]

- Generic solver for alt-free BESs
- Part of Open/Caesar
- 16 000 lines of C
- 17 primitives

### Table: Algorithm Analysis

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Type of BES</th>
<th>Strategy</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>general</td>
<td>DFS</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>A1</td>
<td>general</td>
<td>BFS</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>A2</td>
<td>acyclic</td>
<td>DFS</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>A3</td>
<td>disjunctive</td>
<td>DFS</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>A4</td>
<td>conjunctive</td>
<td>DFS</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>A5</td>
<td>general</td>
<td>DFS</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>A6</td>
<td>disjunctive, unique resolution</td>
<td>BFS</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>A7</td>
<td>conjunctive, unique resolution</td>
<td>DFS</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>A8</td>
<td>general</td>
<td>DFS</td>
<td>$O(</td>
<td>V</td>
</tr>
</tbody>
</table>

Diagnostic generation [Mateescu-00]

One prototype distributed algorithm
On-the-fly Model Checking in CADP

- **Evaluator 2.0 [1995-1997]**
  - alternation-free μ-calculus (AFMC)

- **Evaluator 3.0 [1999]**
  - regular alternation-free μ-calculus (RAFMC)
  - New (built-in) BES resolution algorithm
  - Libraries of derived operators

- **Evaluator 3.5 [2005]**
  - RAFMC
  - Several new BES resolution algorithms (Caesar_Solve library)
  - Formula optimizations
  - Improved diagnostics
  - Tau-confluence reduction

- **Evaluator 4.0 [2011]**
  - Model Checking Language (MCL)
  - Data handling
  - Fairness operators
MCL (Model Checking Language)  
[Mateescu-Thivolle-08]

Extension of modal $\mu$-calculus with:
- Regular expressions over sequences [Mateescu-Sighireanu-03]
- Modalities extracting data values from LTS labels
- Parameterized fixed point operators
- Fairness operators (infinite looping)
- Constructs inspired from programming languages

Tool support: **Evaluator 4.0**
- On-the-fly verification of MCL formulas on LTSs
- Diagnostic generation (witnesses & counterexamples)
- Libraries of derived operators (CTL, ACTL, ...) and property patterns [Dwyer-et-al-99]
Mutual Exclusion

[Mateescu-Serwe-12]

Two processes can never execute simultaneously their critical sections.

\[
\begin{align*}
\text{true}^* & . \\
\{ \text{CS !"ENTER" } ?j: \text{Nat} \} . \\
(\text{not} \{ \text{CS !"LEAVE" } !j \})^* . \\
\{ \text{CS !"ENTER" } ?k: \text{Nat where } k \neq j \} \\
\text{false}
\end{align*}
\]

\text{fully parametric MCL formula (depends only on information present on LTS transitions)}
Liveloock Freedom
(two processes [Bar-David-Taubenfeld-03])

There is no cycle in which each process executes an instruction but no one enters its critical section.

\[
\text{true}^* \cdot \{ \text{NCS } ?j: \text{Nat} \} . \\
(\text{not } \{ \text{?any } ?"READ" | ?"WRITE" ... !j \})^* . \\
\{ \text{?any } ?"READ" | ?"WRITE" ... !j \} \\
\text{not} < (\text{not } \{ \text{CS ... } \})^* . \\
\{ ?G: \text{String} ... ?k: \text{Nat} \text{ where } G \not= "CS" \} . \\
(\text{not } \{ \text{CS ... } \})^* . \\
\{ ?G: \text{String} ... !1 - k \text{ where } G \not= "CS" \} \\
> @
\]
Livellock Freedom
(LTS view)

NCS !j

V !"OP" !j

unfair cycle

V !"OP" !0

V !"OP" !1

not CS ...

not CS ...

not CS ...

not CS ...
Livellock Freedom
(n processes)

There is no cycle in which each process executes an instruction but no one enters its critical section.

```plaintext
[ true* . { NCS ?j:Nat } .
 (not { ?any ?"READ" | "WRITE" ... !j })* .
 { ?any ?"READ" | "WRITE" ... !j }
] not < for j:Nat from 0 to n − 1 do
 (not { CS ... })* .
 { ?G:String ... !j where G <> "CS" }
end for
> @
```
Livellock Freedom

(n processes – plain MCL formulation)

\[
\begin{align*}
\text{nu } X &. ( [ \{ \text{NCS } ?j:\text{Nat} \} ] \\
\text{nu } Y &. ( [ \{ \text{?any } ?"READ"|"WRITE" ... !j \} ] \\
\text{mu } Z &. \text{nu } U (j:\text{Nat}:=0) . \\
\text{if } j = n - 1 \text{ then } Z \text{ else} \\
\text{nu } W &. \\
&. [ \{ \text{?G:String ... !j where G <> "CS" } \} ] U (j + 1) \\
&. \text{and } [ (\text{not } \{ \text{CS ... } \})* ] \ W \\
\text{end if} \\
&. \text{and } [ \text{not } \{ \text{?any } ?"READ"|"WRITE" ... !j \} ] \ Y) \\
&. \text{and } [ \text{true } ] \ X)
\end{align*}
\]
Starvation Witness

Protocol 3b_p2

[BDT-03]

P₀ overtakes P₁ indefinitely
Bounded Overtaking

How many times a process $P_i$ can be overtaken by another process $P_j$ in accessing the critical section?

< true* . { NCS !i } .
  (not { ?any ?"READ" | "WRITE" ... !i })* .
  { ?any ?"READ" | "WRITE" ... !i } .
  ( (not { CS ?any !i }))* .
    { ?G:String ... !i where G <> "CS" } .
      (not { CS ?any !i })* . { CS !"ENTER" !j }
  ) { overtaking_times }
>
true

regular formula with counting: overtaking degree of $P_i$ by $P_j$  

$P_j$ overtakes $P_i$
Witness of Maximum Overtaking

Dekker’s protocol for two processes (at most 4 overtakes of $P_1$ by $P_0$):
Encoding Classical TL Operators

**CTL (and Action-based CTL):**

\[ E (P_1 U P_2) = \mu X . (P_2 \text{ or } (P_1 \text{ and } < \text{true} > X)) \]

\[ A (P_1 U P_2) = \mu X . (P_2 \text{ or } (P_1 \text{ and } < \text{true} > \text{true and [true] X})) \]

**PDL (Propositional Dynamic Logic):**

\[ < R > P \]
\[ [ R ] P \]
\[ < R > @ \]

Temporal patterns [Dwyer et al]

http://cadp.inria.fr/resources/rafmc.html
Generalized Büchi Automata

**TGBAs** *(Transition-based GBAs)*:
- Accepting cycles must contain *all* kinds of final states
- Target formalism for LTL model checking *[SPOT]*

**MCL encoding of accepting cycles in TGBAs:**

\[
\text{true}^* \cdot \{ F \; !i \}^* \text{true}^* \cdot \{ F \; !i \}^* \cdot \ldots
\]

\[
\text{true}^* \cdot \{ F \; !1 \}^* \cdot \{ F \; !n \}^* \cdot \ldots
\]
Context-Free Properties

Syntax analysis on sequences:

\[
\mu X (c: \text{Nat} := 0). ( \\
  (([ \text{true} ] \text{false}) \text{implies } (c = 0)) \text{ and } \\
  [ "(" ] X (c + 1) \text{ and } \\
  [ ")" ] ((c > 0) \text{ and } X (c - 1)) \\
) \\
\]

Allows to simulate pushdown automata
(store the stack in a parameter)
On-the-Fly Verification Method
(Evaluator 4.0)

LNT specification

On-the-fly activities

Caesar_Solve

Open/Caesar environment

compilation

translation

optimisation

parameterized HMLR

encoding

instantiation & resolution

verdict & diagnostic

Caesar_Solve

MCL formula

specification

parameters

HMLR

BES

translation

optimisation
Example

Every PUT !n is followed by a cycle PUT !m ... GET !m:

\[
\begin{array}{l}
\text{true}^* . \{ \text{PUT ?} n : \text{nat} \} \\
\text{< true}^*. \{ \text{PUT ?} m : \text{nat where } m \not= n \} . \text{true}^*. \{ \text{GET !} m \} > @
\end{array}
\]

\[
\begin{array}{l}
\text{nu } X . ( \lbrack \{ \text{PUT ?} n : \text{nat} \} \rbrack \\
\qquad \text{nu } Y (n' : \text{Nat} := n) . \\
\qquad \text{< true}^*. \{ \text{PUT ?} m : \text{nat where } m \not= n \} . \\
\qquad \text{true}^*. \{ \text{GET !} m \} > Y (n')
\end{array}
\]

and

\[
\begin{array}{l}
\text{true} \} X \}
\end{array}
\]

MCL regular modalities

MCL fixed points
### Translation into HMLR

Let's consider the following expressions:

1. \( \nu X. \{ \{ \text{PUT } ?n: \text{nat} \} \} \)
2. \( \text{and } \{ \text{true } \} X \)
3. \( \{ X = \nu \{ \{ \text{PUT } ?n: \text{nat} \} \} Y (n) \} \)
4. \( \text{and } \{ \text{true } \} X \)

For each of these expressions, we will translate them into HMLR terms.

#### Translation Details

1. **Expression 1:**
   \[
   \nu X. \{ \{ \text{PUT } ?n: \text{nat} \} \}
   \]
   This expression translates to:
   \[
   \nu Y (n': \text{Nat} := n).
   \]
   
2. **Expression 3:**
   \[
   \{ X = \nu \{ \{ \text{PUT } ?n: \text{nat} \} \} Y (n) \}
   \]
   This expression translates to:
   \[
   \{ Y (n': \text{Nat}) = \mu Z (n') \}
   \]
   
3. **Expression 4:**
   \[
   \{ \text{true } \} X
   \]
   This expression translates to:
   \[
   \{ Y (n': \text{Nat}) = \mu Z (n') \}
   \]
   
#### Additional Details

- The translation includes conditional clauses and recursive definitions.
- The use of \( \nu \) (universal quantification) and \( \mu \) (existential quantification) in the expressions.

This translation approach is crucial for understanding the semantics of the original expressions in the context of HMLR.
Translation into BES

\[
\{ X_s = \nu \bigwedge_{s \text{-PUT}} !n \rightarrow s' \ Y_s' (n) \\
\text{and} \\
\bigwedge_{s \text{-a} \rightarrow s'} X_{s'} \}
\]

\[
\{ Y_s (n':\text{Nat}) = \mu Z_s (n') \\
Z_s (n':\text{Nat}) = \mu \bigvee_{s \text{-PUT}} !m \rightarrow s' W_s' (m,n') \\
\text{or} \ igvee_{s \text{-a} \rightarrow s'} Z_s' (n') \\
W_s (m',n':\text{Nat}) = \mu \bigvee_{s \text{-GET}} !m' \rightarrow s' Y_s' (m) \\
\text{or} \ igvee_{s \text{-a} \rightarrow s'} Y_s' (n') \}
\]

Parameterized BES

- Encoding scheme: \( X_s = "s \models X"\) \( Y_s (n') = "s \models Y (n')"\)
- Instantiation into a plain BES
- On-the-fly resolution
On-the-fly BES Resolution

Block $M_1$:
- Conjunctive
- Algo. A4 (DFS, no SCCs detection)

Block $M_2$:
- Disjunctive, marked
- Algo. $A3_{cyc}$ (DFS, detection of marked SCCs)
**Divergence**

Data parameters of infinite types $\rightarrow$ termination of model checking not guaranteed anymore

(Pathological) property:

LTS:

$$\mu X (n:\text{Nat} := 0) \cdot <a > X (n + 1)$$

BES:

$$\begin{align*}
\{ X_s (n:\text{Nat}) = &_{\mu} \text{OR}_{s \rightarrow a s'} X_{s'} (n + 1) \} = \\
\{ X_s (n:\text{Nat}) = &_{\mu} X_s (n + 1) \}
\end{align*}$$

Diagram:

- $X_s(0) \rightarrow X_s(1) \rightarrow X_s(2) \rightarrow \ldots \rightarrow X_s(n) \rightarrow \ldots$
Expressiveness and Complexity
(dataless part of MCL)

- **PDL**
  - Linear-time complexity
  - \( \text{LTS size} \times \text{formula size} \)

- **CTL**
  - Quadratic-time complexity
  - \( (\text{LTS size} \times \text{formula size})^2 \)

- **L\(\mu\)_1**
- **L\(\mu\)_2**

- **MCL**
  - Linear-time complexity
  - \( \text{LTS size} \times \text{formula size} \)
Applications of Evaluator
(http://cadp.inria.fr/case-studies)

- Air traffic control system
- General Packet Radio Service (GPRS)
- SPLICE coordination architecture
- Steam-boiler system
- Distributed locker system
- Truck lifting system
- Dynamic reconfiguration protocol for mobile agents
- Checkpointing algorithms in distributed systems
- AIDA (Automatic In-Flight Data Acquisition) system
- JavaSpaces shared data space architecture
- Needham-Schroeder public key authentication protocol
- Replication features of the Splice architecture
- Positive Acknowledgement Retransmission (PAR) protocol
- Jackal distributed shared memory implementation of Java
- Parallel programs developed using JavaSpaces™
- Agent-based dynamic online auction protocol
- Pragmatic General Multicast (PGM) protocol
- Asynchronous circuit for Data Encryption Standard (DES)
- Chilean electronic invoices system
- Fair payment protocol
- Negotiation among Web services
- Turntable system for drilling products
- Fair non-repudiation protocol
- Fractal components
- Fault-tolerant Erlang programs
- Software components
- Automatic document feeder
- Digital Rights Management (DRM) protocol
- Observational determinism for security protocols
- Verification of distributed shared memory systems
- Asynchronous network-on-chip
- .NET on-line sale application
- Dining cryptographers and electronic voting scheme
- Erlang’s Open Telecom Platform (OTP) library
- IEEE 1394 tree identify protocol
- Dutch Rijnland Internet Election System (RIES)
- Adaptors between evolving components
- Verification and adaptation of WF/.NET components
- Specification and analysis of a Web service for GPS navigation
- Collaboration diagrams
- Verification of a turntable system
- Formal analysis of consensus protocols
- Accelerated heartbeat protocols
- Blitter display
- Trivial file transfer protocol
- Performance evaluation of MPI on CC-NUMA architectures
- Test generation for automotive industry
- Air traffic control subsystem
- Model checking Erlang programs
- Verification of Web service composition
- Systematic correct construction of self-stabilizing systems
- Verification of behavioural properties for group communications
- Analysis of pi-calculus specifications
- Mutual exclusion protocols
- Behavior Analysis of Malware by Rewriting-Based Abstraction
- Safety Verification of Fault-Tolerant Distributed Components
- Verification of Mobile Ad Hoc Networks
- Atomicity Maintenance in EPCReport of ALE
- Rigorous Development of Prompting Dialogs
- Formal Analysis and Co-simulation of a Dynamic Task Dispatcher
- SYNERGY Reconfiguration Protocol
- Self-configuration Protocol for the Cloud
### BES Resolution for Model Checking

<table>
<thead>
<tr>
<th>Condition</th>
<th>Algorithm</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_{\mu_1}$ input formula -dfs</td>
<td>A0</td>
<td>stores LTS transitions</td>
</tr>
<tr>
<td>PDL or (A)CTL input formula -dfs</td>
<td>A3, A4</td>
<td>stores only LTS states</td>
</tr>
<tr>
<td>$L_{\mu_1}$ input formula -bfs</td>
<td>A1</td>
<td>small diagnostics \ stores LTS transitions</td>
</tr>
<tr>
<td>PDL or (A)CTL input formula -bfs</td>
<td>A6, A7</td>
<td>small diagnostics \ stores only LTS states</td>
</tr>
<tr>
<td>-acyclic</td>
<td>A2</td>
<td>stores only LTS states</td>
</tr>
</tbody>
</table>

- On-the-fly linear-time model checking procedures
- No storage of LTS transitions for usual operators
### BES Resolution for Equivalence Checking and Reduction

**(Bisimulator, Reductor)**

<table>
<thead>
<tr>
<th>Equivalence relation</th>
<th>Condition</th>
<th>Algorithm</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Branching</td>
<td>LTSs nondeterministic -dfs</td>
<td>A0</td>
<td>stores LTSs transitions</td>
</tr>
<tr>
<td>Weak Branching</td>
<td>1 LTS deterministic and τ-free -dfs</td>
<td>A4</td>
<td>stores only LTSs states</td>
</tr>
<tr>
<td>Tau*.a Safety</td>
<td>LTSs nondeterministic -bfs</td>
<td>A1</td>
<td>small diagnostics stores LTSs transitions</td>
</tr>
<tr>
<td>Trace Weak trace</td>
<td>1 LTS deterministic and τ-free -bfs</td>
<td>A7</td>
<td>small diagnostics stores only LTSs states</td>
</tr>
<tr>
<td></td>
<td>-acyclic</td>
<td>A2</td>
<td>stores only LTSs states</td>
</tr>
<tr>
<td>Tau-confluence</td>
<td>always</td>
<td>A5, A8</td>
<td>small average complexity</td>
</tr>
</tbody>
</table>
BACKWARD WALKS
Explicit LTSs
(BCG file format)

Represented in extension (set of transitions)

BCG (*Binary-Coded Graphs*):

- Compact file format for storing LTSs
- Set of APIs and libraries (create/read/explore)
- Set of tools
  - `bcg_info`: extract info from a BCG file
  - `bcg_io`: convert BCG from and to other formats
  - `bcg_labels`: hide and/or rename labels
  - `bcg_draw, bcg_edit`: visualize LTSs
  - `bcg_graph`: generation of particular BCG graphs
  - `bcg_open`: connection to Open/Cæsar applications
XTL (eXecutable Temporal Language)  
Mateescu-Garavel-98

- Functional-like language for LTS exploration
  - Explicit manipulation of states/labels/edges (+sets)
  - Handling of data values contained in labels
- Definition of temporal logic operators using fixed point computations (CTL, ACTL, mu-calculus)
- Non-standard operators (nondeterminism, cycles, ...)
- Import of external C types and functions
- Macro-definitions and reusable libraries

Model checker built on top of the BCG libraries
Walking Backward

< A > P diamond modality of HML (Hennessy-Milner Logic):

\[
\text{def } \text{Dia} (A: \text{labelset}, P: \text{stateset}) : \text{stateset} = \\
\{ \text{S: state where} \\
\exists \text{T: edge among} \text{ out (S) in} \\
(\text{label (T) among A}) \text{ and } (\text{target (T) among P}) \\
\text{end}_\exists \\
\} \\
\text{end_def}
\]
**Least fixed point operator of modal μ-calculus:**

\[
\text{macro lfp (X, P) =}
\]
\[
\quad \text{let (R: stateset, any boolean) =}
\]
\[
\quad \quad \text{for}
\]
\[
\quad \quad \quad \text{in} \quad (X: stateset, STABLE: boolean)
\]
\[
\quad \quad \quad \text{while} \quad \text{not (STABLE)}
\]
\[
\quad \quad \quad \text{apply (union, or)}
\]
\[
\quad \quad \quad \text{from} \quad (\{\}, \text{false})
\]
\[
\quad \quad \quad \text{to} \quad \text{let} \quad Y: \text{stateset} = (P)
\]
\[
\quad \quad \quad \quad \quad \text{in}
\]
\[
\quad \quad \quad \quad \quad \quad \quad \quad \quad (Y, X = Y)
\]
\[
\quad \quad \quad \quad \quad \text{end_let}
\]
\[
\quad \quad \quad \text{end_for}
\]
\[
\quad \text{in}
\]
\[
\quad R
\]
\[
\quad \text{end_let}
\]
\[
\text{end_macro}
\]

**E[P_A U_B Q] operator of ACTL (Action-based CTL):**

\[
\text{def EU_A_B (P: stateset, A, B: labelset, Q: stateset) : stateset =}
\]
\[
\quad \text{lfp (X, P and}
\]
\[
\quad \quad \text{(Dia (B, Q) or Dia (A or TAU, X)))}
\]
\[
\quad \text{end_def}
\]
Walking Forward

Computing the radius of an LTS:

```
for in (EXPLORED, LEVEL: stateset, RADIUS: number)
    while LEVEL <> {}
        apply (union, replace, +)
        from ({}, { init }, #0)
        to let NEXT: stateset =
            <| union on S:state among LEVEL |> succ (S)
        in
            (NEXT, NEXT diff EXPLORED, 1)
    end_let
end_for
```
Handling Data

Between two consecutive emissions, there is a corresponding reception:

$$AG \ (\forall m: \text{integer} \ \text{among} \ \{0 \ldots N\} \ \text{in}$$

$$\text{Box} (\text{PUT} (m), \text{AG}_A (\text{not} (\text{GET} (m)), \text{Box} (\text{PUT}_\text{any}, \text{false})))$$

$$\text{end}_\forall$$

macro-definitions on state sets and label sets
Applications of XTL
(http://cadp.inria.fr/case-studies)

- Philips' Bounded Retransmission Protocol (BRP)
- Link layer protocol of the IEEE-1394 FireWire serial bus
- Bull's cluster file system
- Invoicing system
- Xpress transfer protocol
- HAVi leader election protocol
- Synchronous hardware
- Hardware/software codesign
- Asynchronous hardware
- Non-refinement transformations of software architectures
- Abstraction and analysis of clinical guidance trees
- Gossiping networks
- Model Checking of Scenario-Aware Dataflow

model checker for the FULL data-based modal logic
Future Walks

Caesar_Solve
- New algorithms (sequential & distributed)
- Further applications (controller synthesis)

MCL
- More powerful operators (no more fixed points)
- Enrich the type system

XTL
- Improve set representation (use *DDs)
- Logics for quantitative analysis (probabilistic/timed)
Thank you

For more information:
http://cadp.inria.fr