Walking Back and Forth in Labelled Transition Systems

Radu Mateescu

Inria Grenoble – Rhône-Alpes / Convecs

http://convecs.inria.fr

Outline

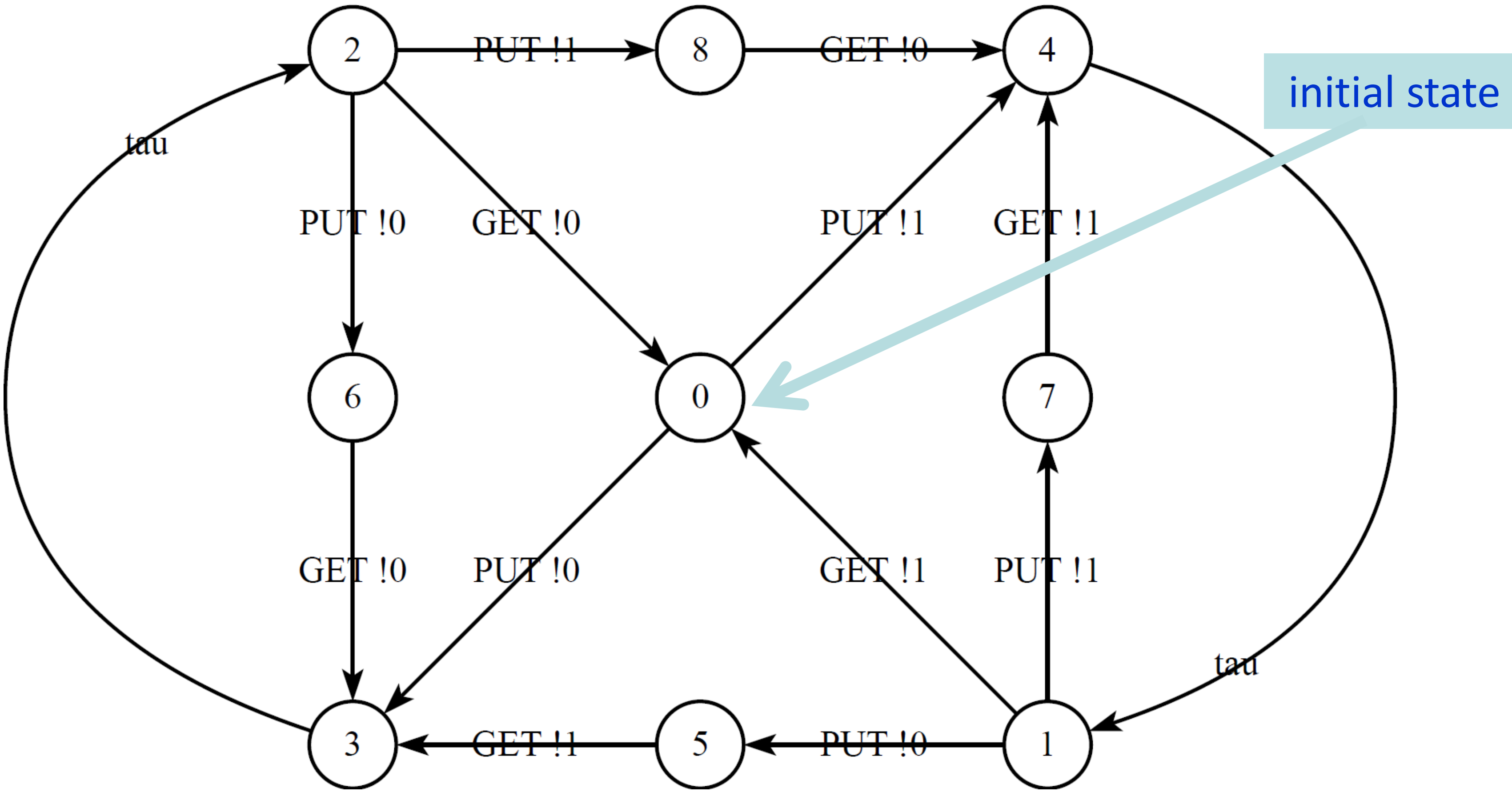
- Concurrent systems, LTSs, and Boolean graphs
- Forward walks
  - MCL – a data-based model checking language
  - On-the-fly model checking
- Backward walks
  - XTL – an executable temporal language
  - Graph querying
- Future walks

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

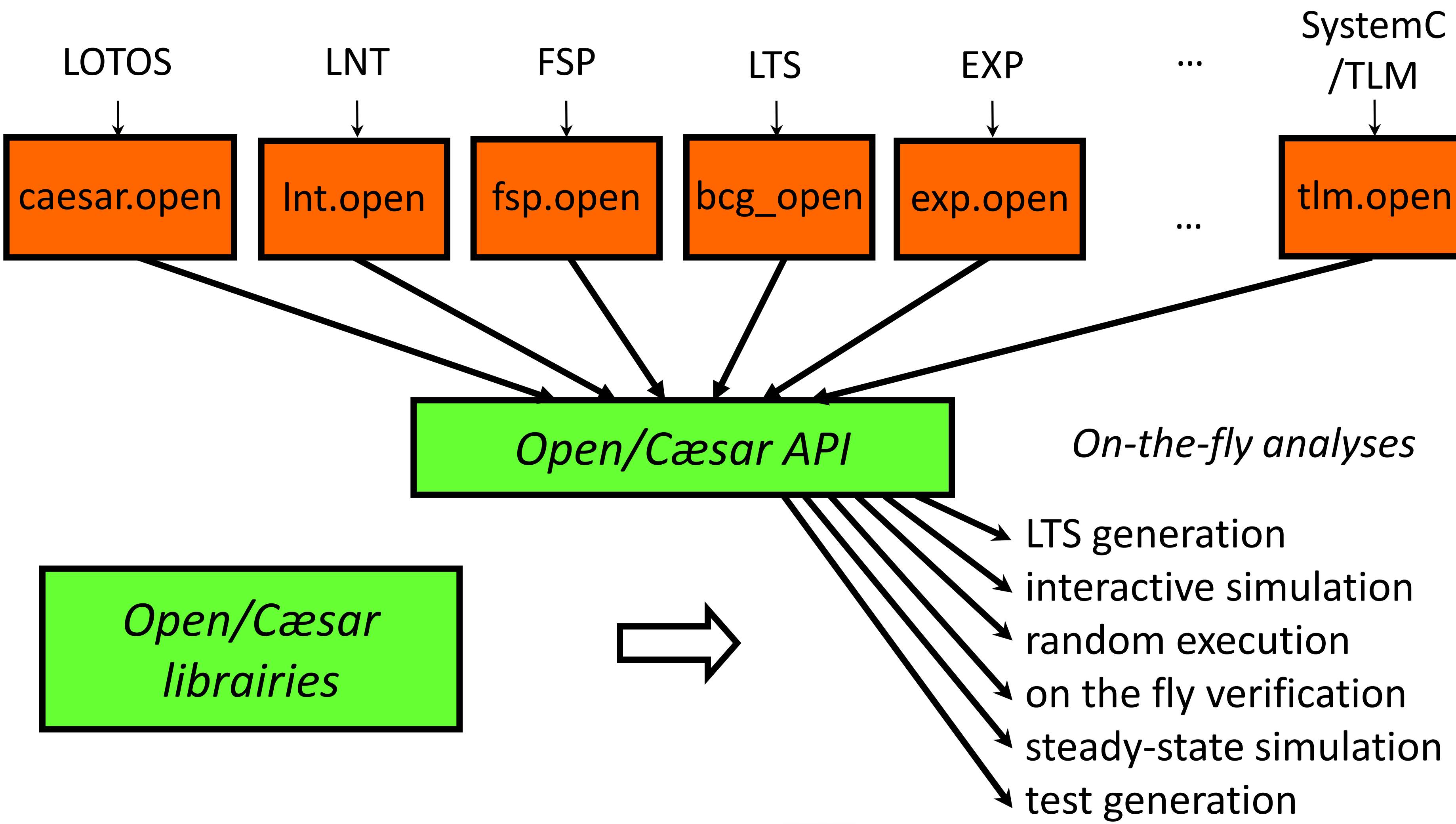
FORWARD WALKS

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/Caesar Architecture



Boolean Equation Systems

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

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

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

no cyclic dependencies between equation blocks (alternation-free)

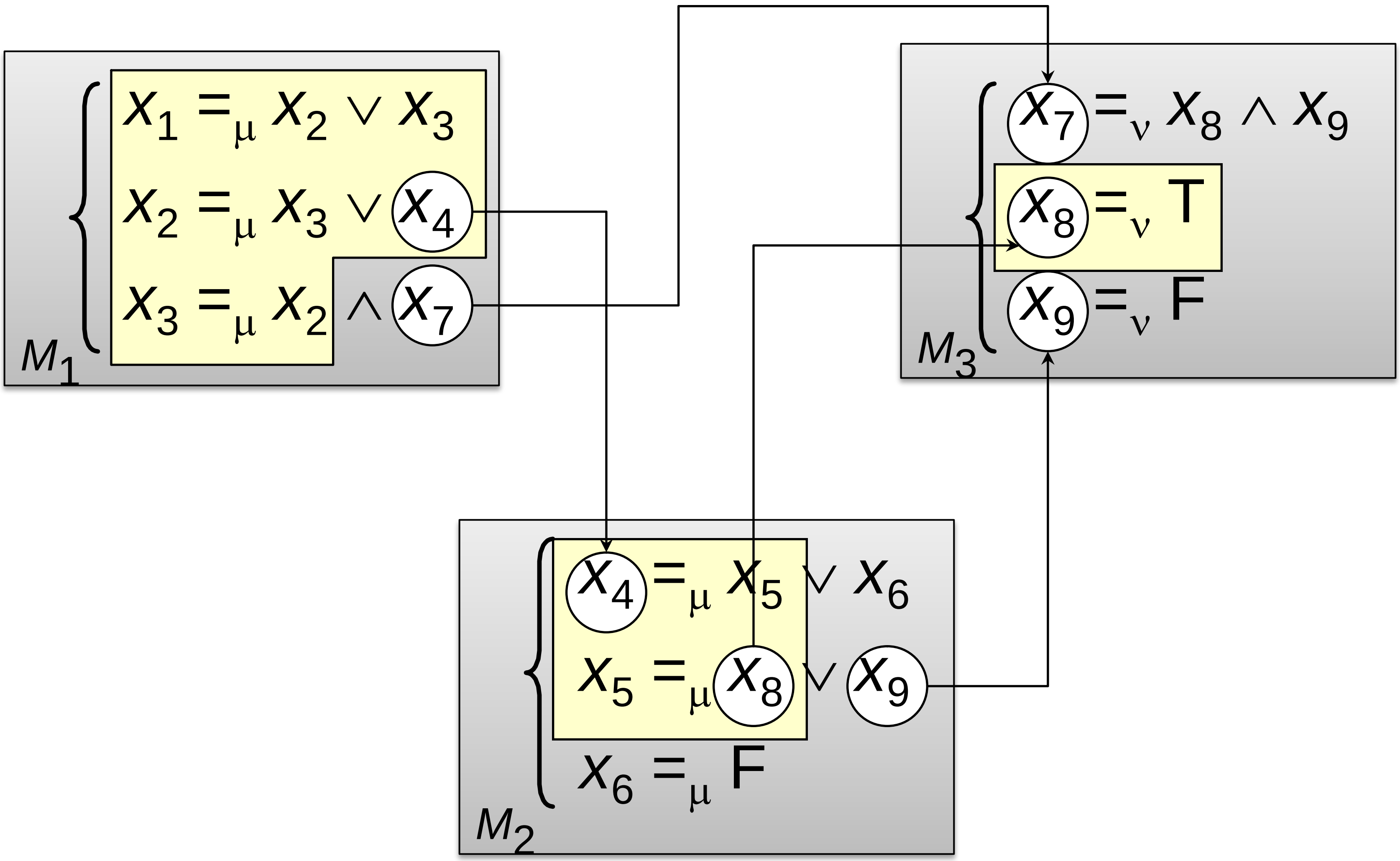
Particular Blocks

- *Acyclic*: no cyclic dependencies between variables
- Var. x_i disjunctive (conjunctive): $op_i = \vee$ ($op_i = \wedge$)
- *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



x1 =μ x2 ∨ x3
x2 =μ x3 ∨ x4
x3 =μ x2 ∧ x7
M1
x7 =ν x8 ∧ x9
x8 =ν T
x9 =ν F
M3
x4 =μ x5 ∨ x6
x5 =μ x8 ∨ x9
x6 =μ F
M2

Local Resolution Algorithms

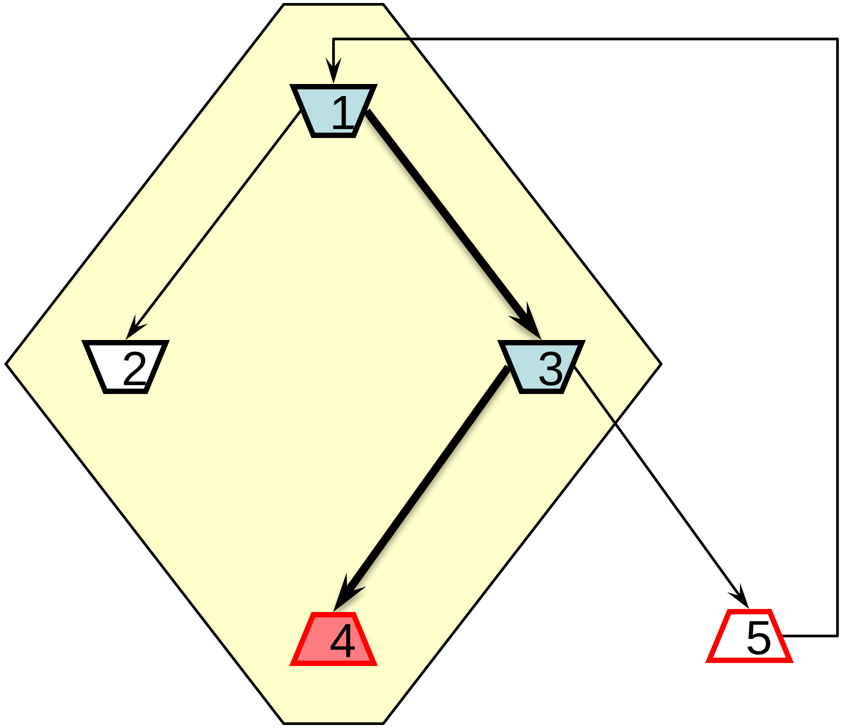
- Represent blocks as *boolean graphs* [Andersen-94]
- To $M = \{ x_j =_\mu op_j \boldsymbol{X}_j \}_{j \text{ in } [1, m]}$ associate $G = (V, E, L, \mu)$
  - $V = \{ x_1, \ldots, x_m \}$: set of vertices (variables)
  - $E = \{ (x_i, x_j) \mid x_j \in \boldsymbol{X}_i \}$: set of edges (dependencies)
  - $L : V \rightarrow \{ \vee, \wedge \}$, $L(x_j) = op_j$: 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-bloc)

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

boolean graph



: ∨-variables

: ∧-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

Alg.	Type of BES	Strategy	Time	Memory
A0	general	DFS	$O(\lvert V\rvert + \lvert E\rvert)$	$O(\lvert V\rvert + \lvert E\rvert)$
A1		BFS		
A2	acyclic	DFS		$O(\lvert V\rvert)$
A3	disjunctive			
A4	conjunctive			
A5	general			$O(\lvert V\rvert + \lvert E\rvert)$
A6	disjunctive, unique resolution	BFS		$O(\lvert V\rvert)$
A7	conjunctive, unique resolution			
A8	general	DFS	$O((\lvert V\rvert + \lvert E\rvert)^2)$	$O(\lvert V\rvert + \lvert E\rvert)$

- Diagnostic generation [Mateescu-00]
- One prototype distributed algorithm

On-the-fly Model Checking in CADP

Evaluator 4.0 [2011]
Model Checking Language (MCL)
+ Data handling
+ Fairness operators

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

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

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

MCL (Model Checking Language)

[Mateescu-Thivolle-08]

- Extension of modal μ-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]

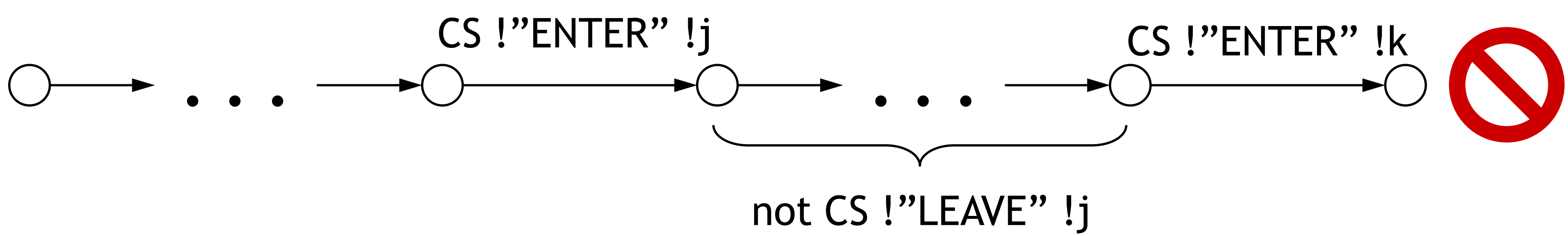
Mutual Exclusion

[Mateescu-Serwe-12]

Two processes can never execute simultaneously their critical sections.

```
[ true* .
  { CS !"ENTER" ?j:Nat } .
  (not { CS !"LEAVE" !j })* .
  { CS !"ENTER" ?k:Nat where k <> j }
] false
```

fully parametric MCL formula (depends only on information present on LTS transitions)



Livelock Freedom

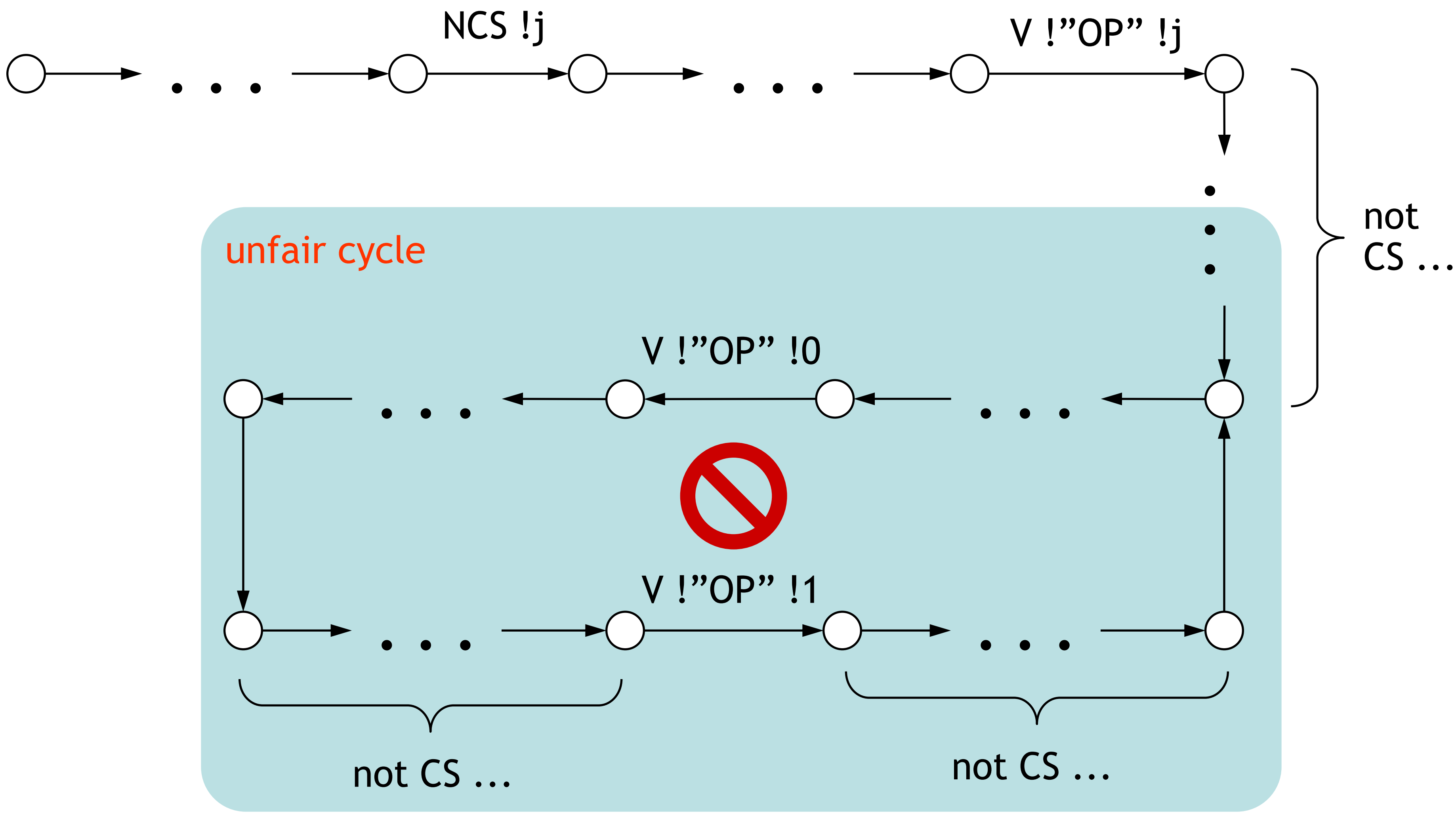
(two processes [Bar-David-Taubenfeld-03])

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

```
[  true* . { NCS ?j:Nat } .
   (not { ?any ?"READ"|"WRITE" ... !j })* .
   { ?any ?"READ"|"WRITE" ... !j }
] not < (not { CS ... })* .
          { ?G:String ... ?k:Nat where G <> "CS" } .
          (not { CS ... })* .
          { ?G:String ... !1 - k where G <> "CS" }
        > @
```

Livelock Freedom

(LTS view)



Livelock Freedom

(*n* processes)

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

```
[  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
      > @
```

complex cycle containing a set of events (generalized Büchi automaton)

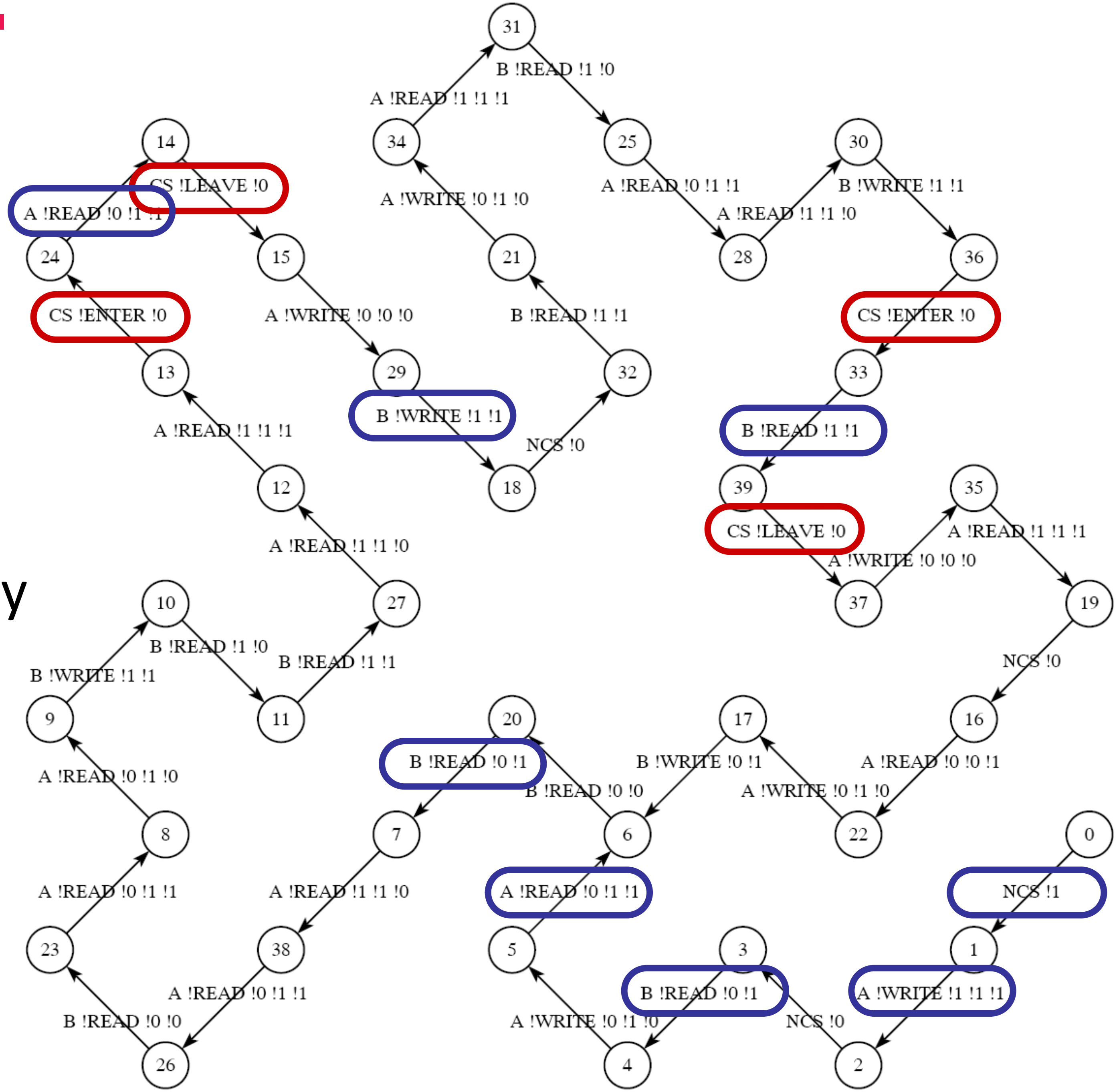
Livelock Freedom

(*n* processes – plain MCL formulation)

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

Starvation Witness

- Protocol 3b_p2 [BDT-03]
- P_0 overtakes P_1 indefinitely



Bounded Overtaking

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

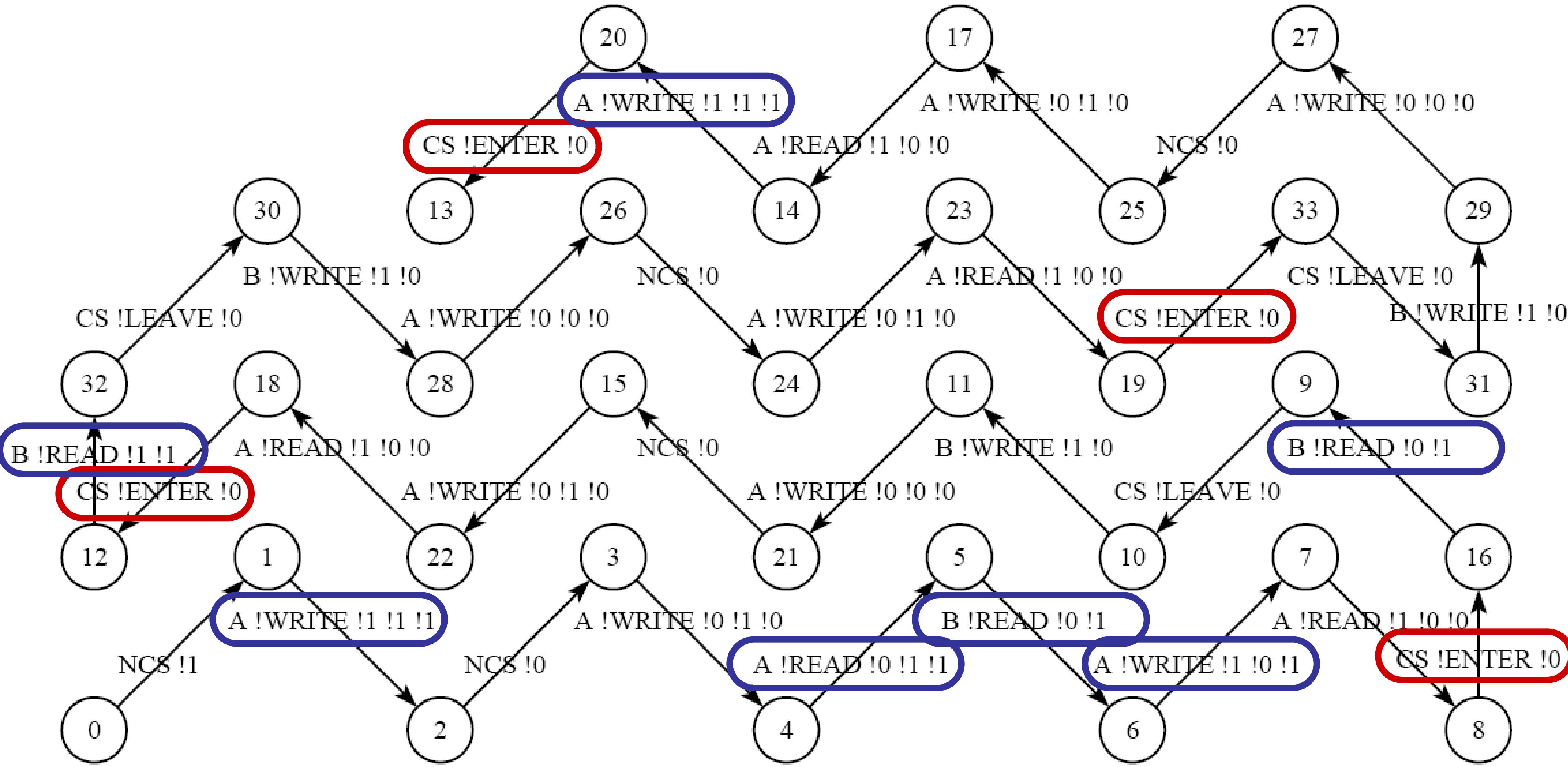
```
< true* . { NCS !i } .
   (not { ?any ?"READ"|"WRITE" ... !i })* .
   { ?any ?"READ"|"WRITE" ... !i } .
   (   (not { CS ?any !i })* .
       { ?G:String ... !i where G <> "CS" } .
       (not { CS ?any !i })* . { CS !"ENTER" !j }
   ) { overtaking_times }
> true
```

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):

  $E\ (P_1\ U\ P_2) = mu\ X\ .\ (P_2\ or\ (P_1\ and < true > X))$

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

- PDL (*Propositional Dynamic Logic*):

  $\left.\begin{array}{l} < R > P \\ [\ R\ ]\ P \\ < R > @ \end{array}\right\}$ included in MCL

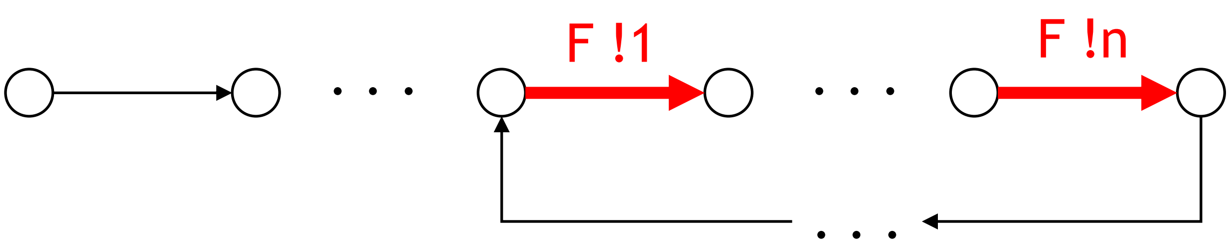
- Temporal patterns [Dwyer et al]

  http://cadp.inria.fr/resources/rafmc.html

Generalized Büchi Automata

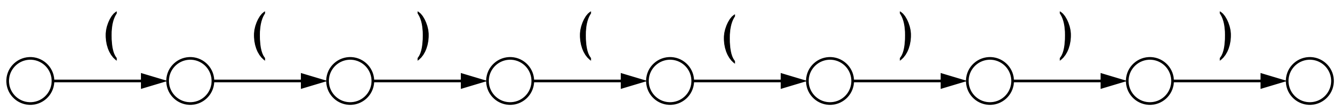
- **TGBAs** (Transition-based GBAs):
  - Accepting cycles must contain *all* kinds of final states
  - Target formalism for LTL model checking [SPOT]
- MCL encoding of accepting cycles in TGBAs:

```
<
   for i:Nat from 1 to n do
      true* . { F !i }
   end for
> @
```



Context-Free Properties

- Syntax analysis on sequences:



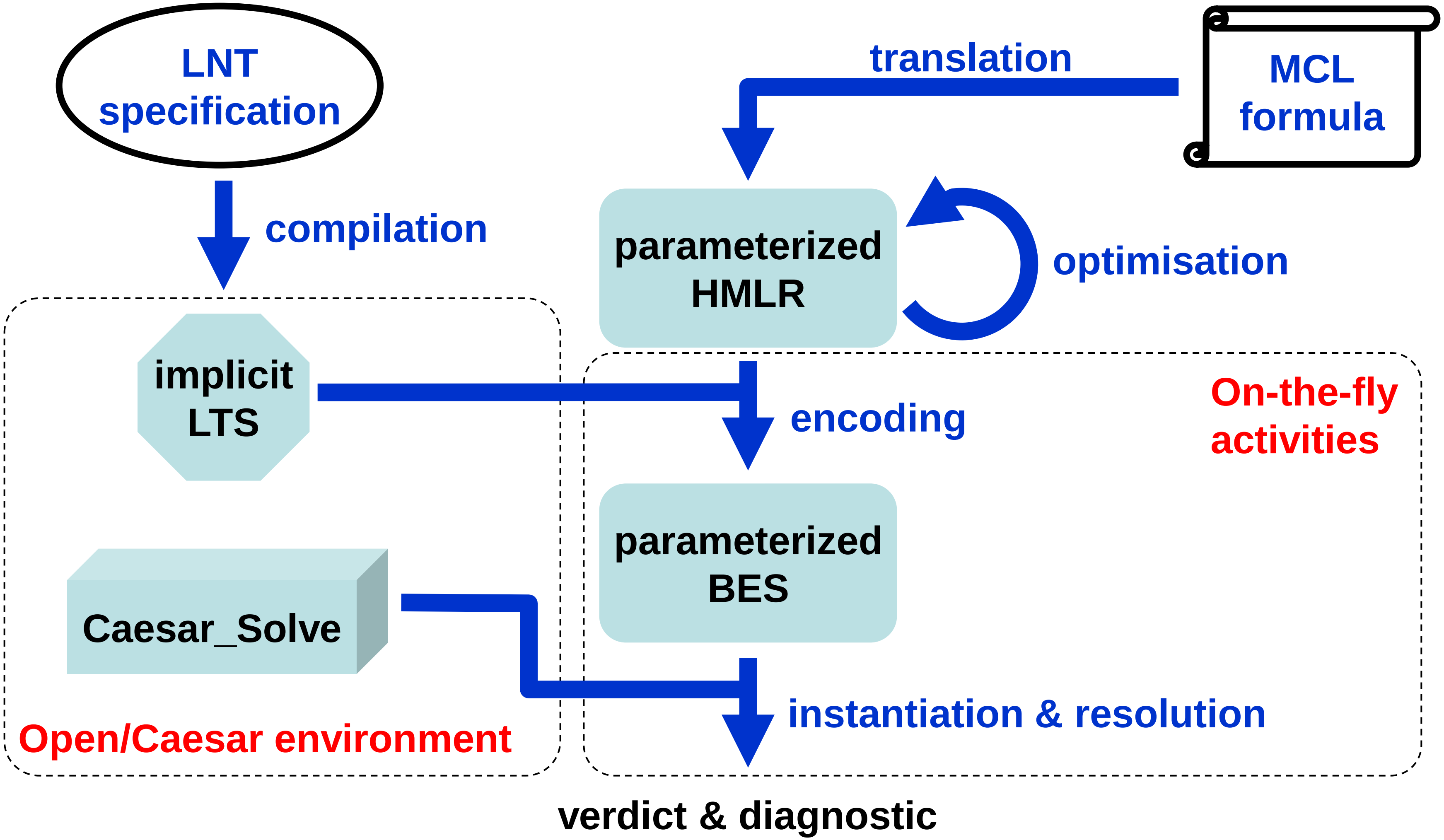

```
mu X (c:Nat := 0) . (
    (([ true ] false) implies (c = 0))    and
    [ "(" ] X (c + 1)                     and
    [ ")" ] ((c > 0) and X (c – 1))
)
```

check for well-formed parantheses

- Allows to simulate pushdown automata (store the stack in a parameter)

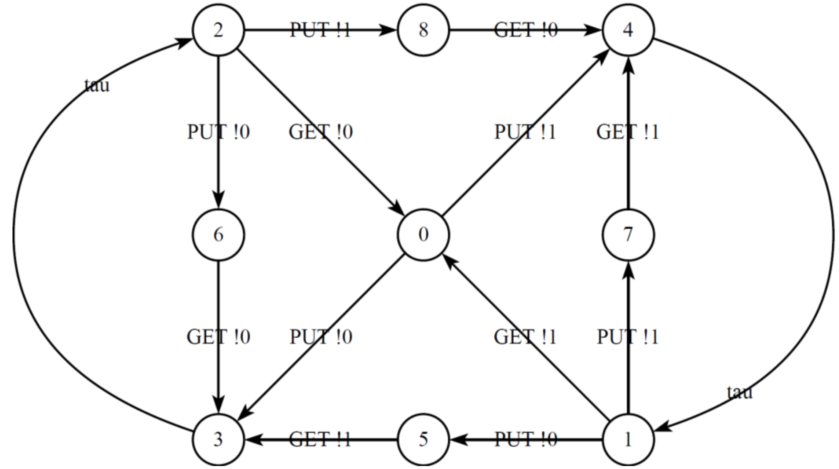
On-the-Fly Verification Method

(Evaluator 4.0)



Example

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



MCL regular modalities

```
[ true* . {PUT ?n:nat} ]
   < true*. {PUT ?m:nat where m <> n} . true*. {GET !m} > @
```

MCL fixed points

```
nu X . ( [ {PUT ?n:nat} ]
            nu Y (n':Nat := n) .
              < true*. {PUT ?m:nat where m <> n} .
                true*. {GET !m} > Y (n')
         and
         [ true ] X )
```

Translation into HMLR

```
nu X . [ {PUT ?n:nat} ]

    and [ true ] X
```

```
nu Y (n':Nat := n) .
    < true* >
    < {PUT ?m:nat where m <> n} >
    < true* > < {GET !m} > Y (n')
```

```
{ X =nu [ {PUT ?n:nat} ] Y (n)

         and

         [ true ] X
}
```

```
{ Y (n':Nat) =mu Z (n')
  Z (n':Nat) =mu
  < {PUT ?m:nat where m <> n'} > W (m,n')
      or < true > Z (n')
  W (m',n':Nat) =mu < {GET !m'} > Y (m)
      or < true > Y (n')

}
```

Translation into BES

$\{ X_s =_{nu} \bigwedge_{s\text{-PUT !n} \rightarrow s'} Y_{s'} (n)$

and

$\bigwedge_{s\text{-a} \rightarrow s'} X_{s'}$

$\}$

$\{ Y_s (n'\text{:Nat}) =_{\mathbf{mu}} Z_s (n')$

$Z_s (n'\text{:Nat}) =_{\mathbf{mu}} \bigvee_{s\text{-PUT !m} \rightarrow s'} W_{s'} (m,n')$

$\quad \text{or } \bigvee_{s\text{-a} \rightarrow s'} Z_{s'} (n')$

$W_s (m',n'\text{:Nat}) =_{\mathbf{mu}} \bigvee_{s\text{-GET !m'} \rightarrow s'} Y_{s'} (m)$

$\quad \text{or } \bigvee_{s\text{-a} \rightarrow s'} Y_{s'} (n')$

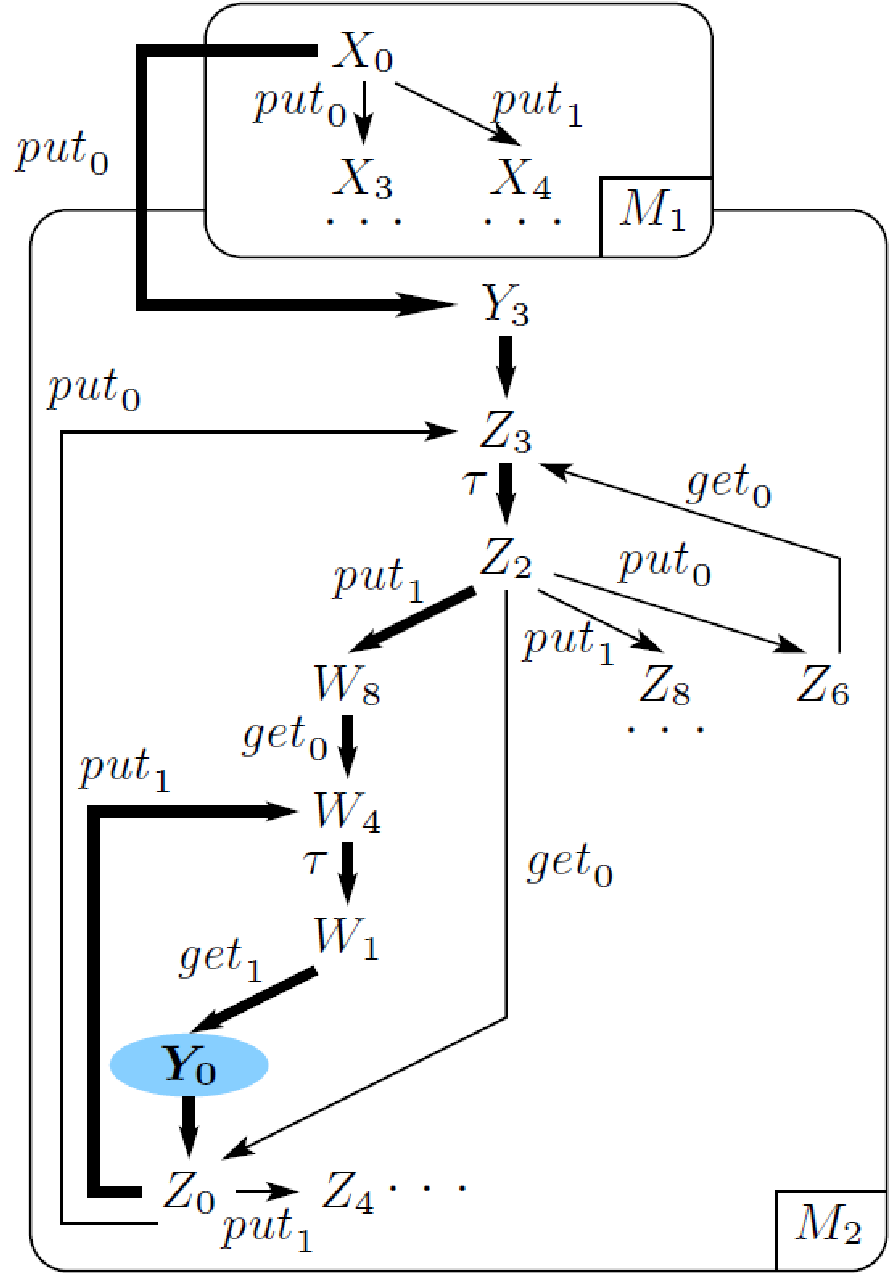
$\}$

Parameterized BES

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

On-the-fly BES Resolution

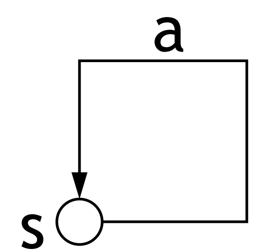
- Block M_1:
  - Conjunctive
  - Algo. A4 (DFS, no SCCs detection)
- Block M_2:
  - Disjunctive, marked
  - Algo. $A3_{cyc}$ (DFS, detection of marked SCCs)



Divergence

- Data parameters of infinite types → termination of model checking not guaranteed anymore
- (Pathological) property:

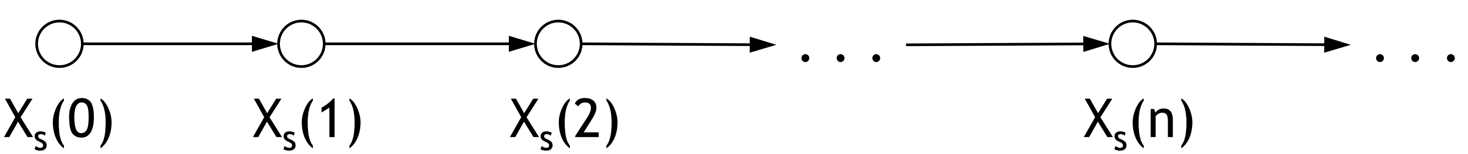
LTS:



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

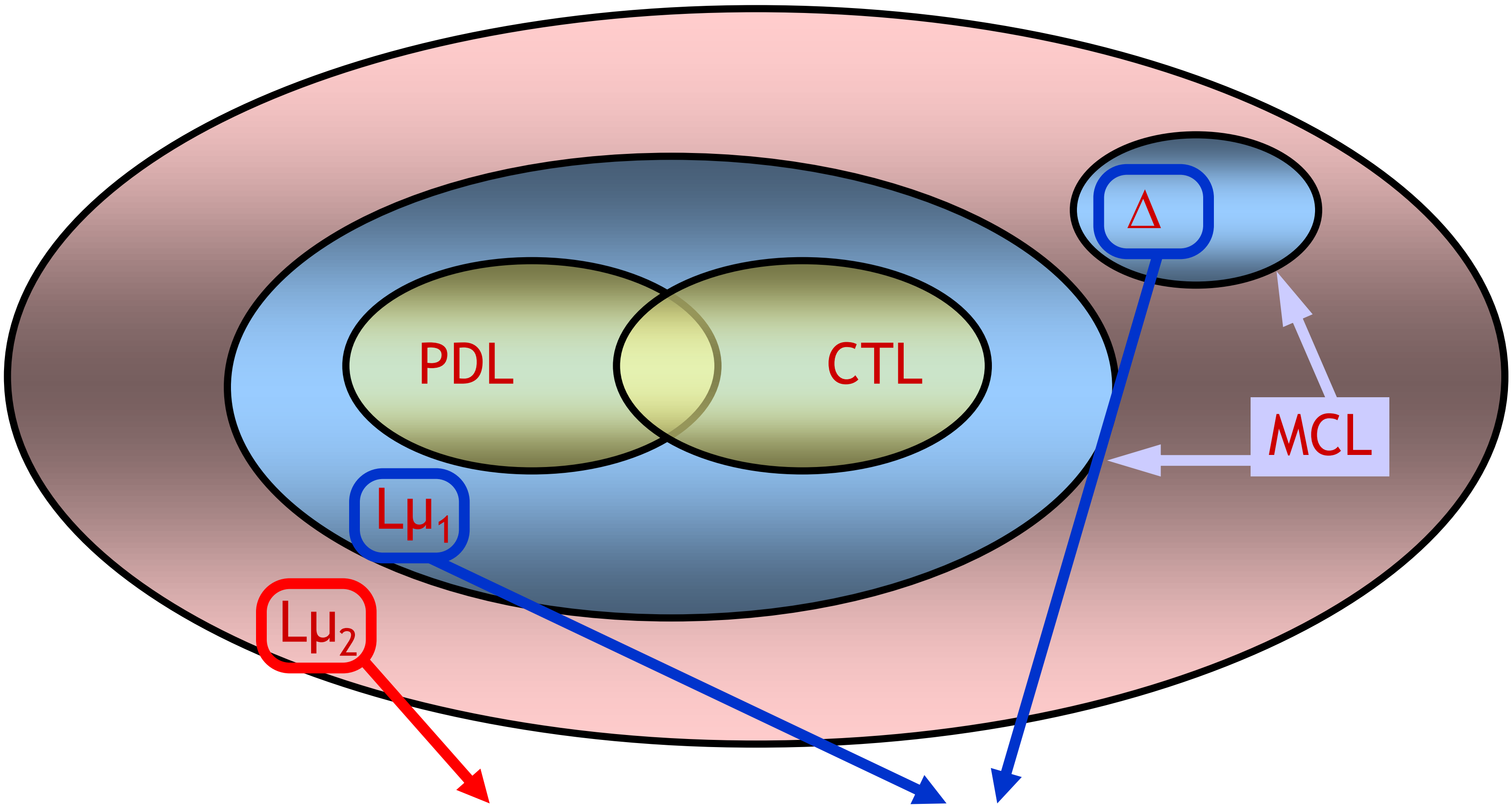
BES: $\{ X_s\ (n{:}Nat) =_{mu} OR_{s \to_a s'}\ X_{s'}\ (n + 1) \} =$

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



Expressiveness and Complexity

(dataless part of MCL)



quadratic-time complexity
(LTS size × formula size)^2

linear-time complexity
LTS size × 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

Condition	Algorithm	Comment
$L\mu_1$ input formula -dfs	A0	stores LTS transitions
PDL or (A)CTL input formula -dfs	A3, A4	stores only LTS states
$L\mu_1$ input formula -bfs	A1	small diagnostics stores LTS transitions
PDL or (A)CTL input formula -bfs	A6, A7	small diagnostics stores only LTS states
-acyclic	A2	stores only LTS states

- 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)

Equivalence relation	Condition	Algorithm	Comment
Strong Weak Branching Tau*.a Safety Trace Weak trace	LTSs nondeterministic -dfs	A0	stores LTSs transitions
	1 LTS deterministic and τ-free -dfs	A4	stores only LTSs states
	LTSs nondeterministic -bfs	A1	small diagnostics stores LTSs transitions
	1 LTS deterministic and τ-free -bfs	A7	small diagnostics stores only LTSs states
	-acyclic	A2	stores only LTSs states
Tau-confluence	always	A5, A8	small average complexity

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

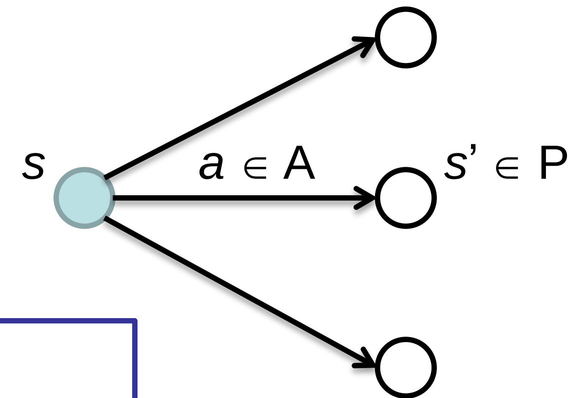
XTL (eXecutable Temporal Language)

[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):

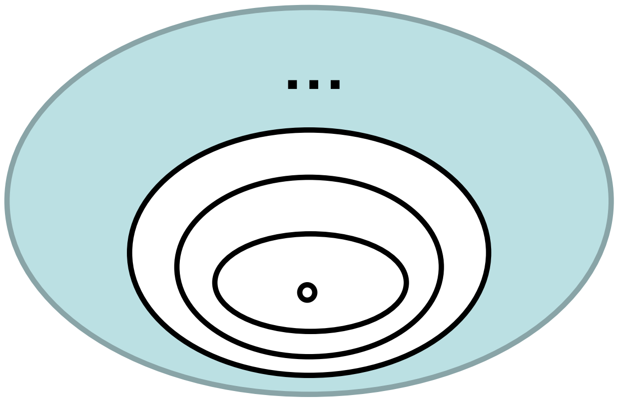



```
def Dia (A: labelset, P: stateset) : stateset =
   { S: state where
      exists T: edge among out (S) in
         (label (T) among A) and (target (T) among P)
      end_exists
   }
end_def
```

Iterating over State Sets

- Least fixed point operator of modal μ-calculus:

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



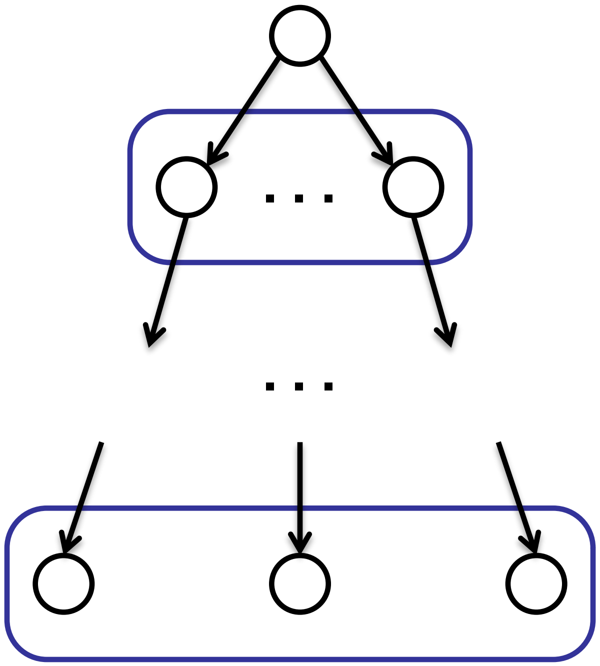
- $E[P_A \ U_B \ Q]$ operator of ACTL (Action-based CTL):

```
def EU_A_B (P: stateset, A, B: labelset,
                    Q: stateset) : stateset =
   lfp (X, P and
              (Dia (B, Q) or Dia (A or TAU, X)))
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:

macro-definitions on state sets and label sets

```
AG (
    forall m: integer among { 0 ... N } in
        Box (
            PUT (m),
            AG_A (not (GET (m)), Box (PUT_any, false))
        )
    end_forall
)
```

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