On-the-Fly Model Checking for Extended Action-Based Probabilistic Operators

Radu Mateescu and José Ignacio Requeno

Inria Grenoble and LIG / Convecs

http://convecs.inria.fr

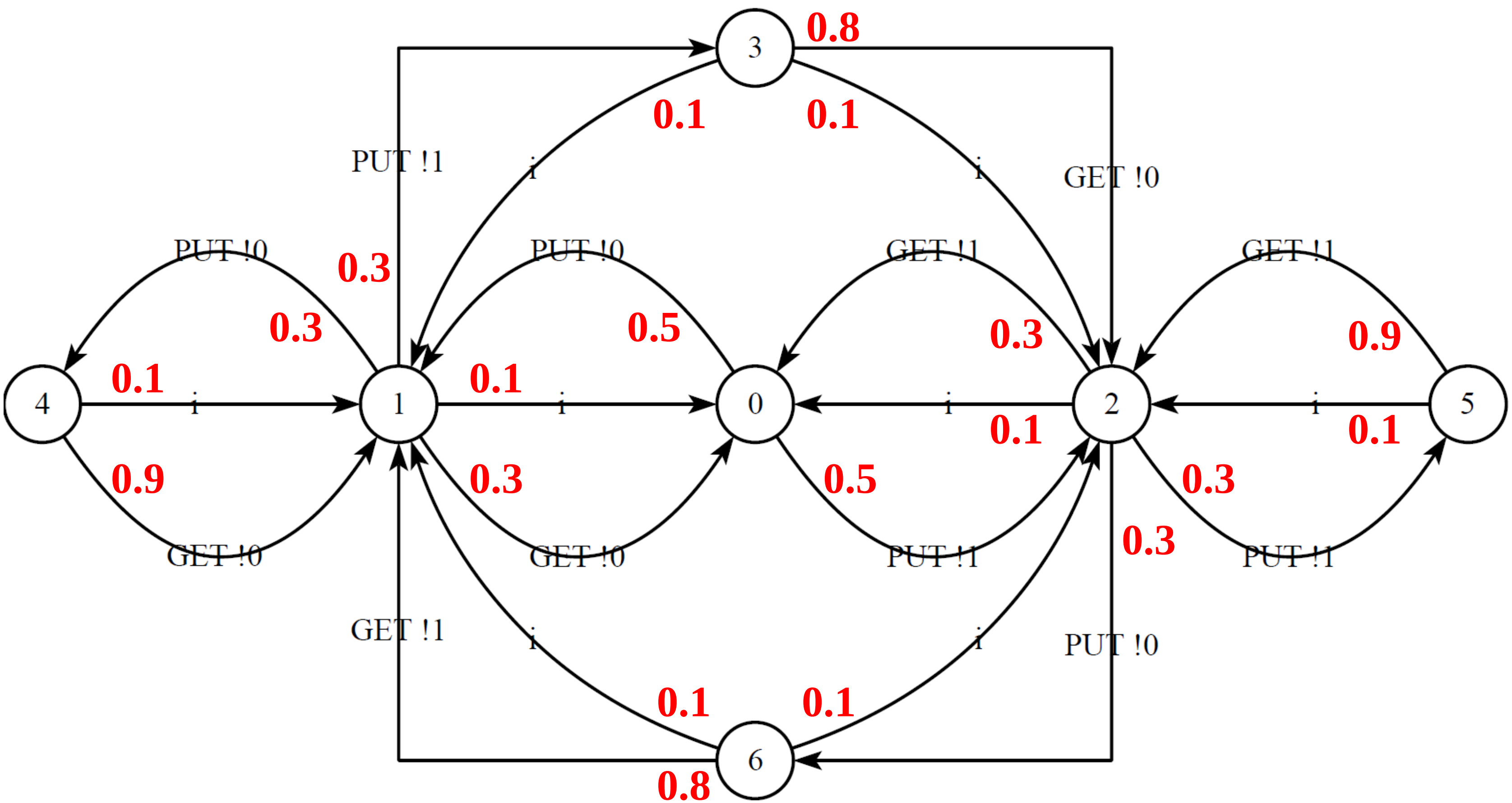
Context

- Concurrent value-passing systems
  - Action-based, branching-time setting
  - Process calculi, bisimulations
- Objectives
  - Handle actions, data, costs, discrete time, probabilities
  - Provide on-the-fly quantitative analysis
  - Integrate into the CADP toolbox (http://cadp.inria.fr)

SENSATION project
FP7-318490
www.sensation-project.eu

Probabilistic Transition Systems

(value-passing)



Related Work

- Action-based probabilistic logics
  - PML (probabilistic HML) on PTSs [Larsen-Skou-91]
  - GPL (probabilistic Lμ_1) [Cleaveland-Iyer-Narasimba-05]
- On-the-fly verification
  - PCTL [Latella-Loreti-Massink-14]
- Our contributions
  - Extension of (action-based) Until PCTL operators
    - Description of complex paths in the PTS
    - Data handling (computation of costs over paths)
  - On-the-fly verification on PTSs

Outline

- Glimpse of MCL (Model Checking Language)
- Regular probabilistic operator
- On-the-fly model checking
- Data-handling extensions
- Experiments and peak cost analysis
- Perspectives

MCL: Model Checking Language

(dataless fragment)

Action formulas **(ACTL)**

$\alpha ::= \text{false} \mid \tau \mid a \mid \neg\alpha \mid \alpha_1 \vee \alpha_2$ *boolean op.*

Regular formulas **(PDL)**

$\beta ::= \alpha \mid \beta_1 . \beta_2 \mid \beta_1 | \beta_2 \mid \beta^*$ *regular op.*

$\mid \text{if } \varphi \text{ then } \beta_1 \text{ else } \beta_2 \text{ end if}$ *conditional op.*

State formulas **(PDL + Lμ)**

$\varphi ::= \text{false} \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$ *boolean op.*

$\mid \langle \beta \rangle \varphi \mid [\beta] \varphi \mid$ *modal op.*

$\mid Y \mid \mu Y . \varphi \mid \nu Y . \varphi$ *fixed point op.*

Some Derived Operators

- Derived regular operators:

$\beta^{+} = \beta . \beta^{*}$ *transitive closure*

nil = false* *empty sequence*

if φ then β end if =
  if φ then β else nil end if *"if-then" op.*

φ? = if not φ then false end if *PDL "testing" op.*

- CTL Until operator:

$E [\varphi_1 \; U \; \varphi_2] = < (\varphi_1? \, . \, \text{true})^{*} . \, \varphi_2? > \text{true}$

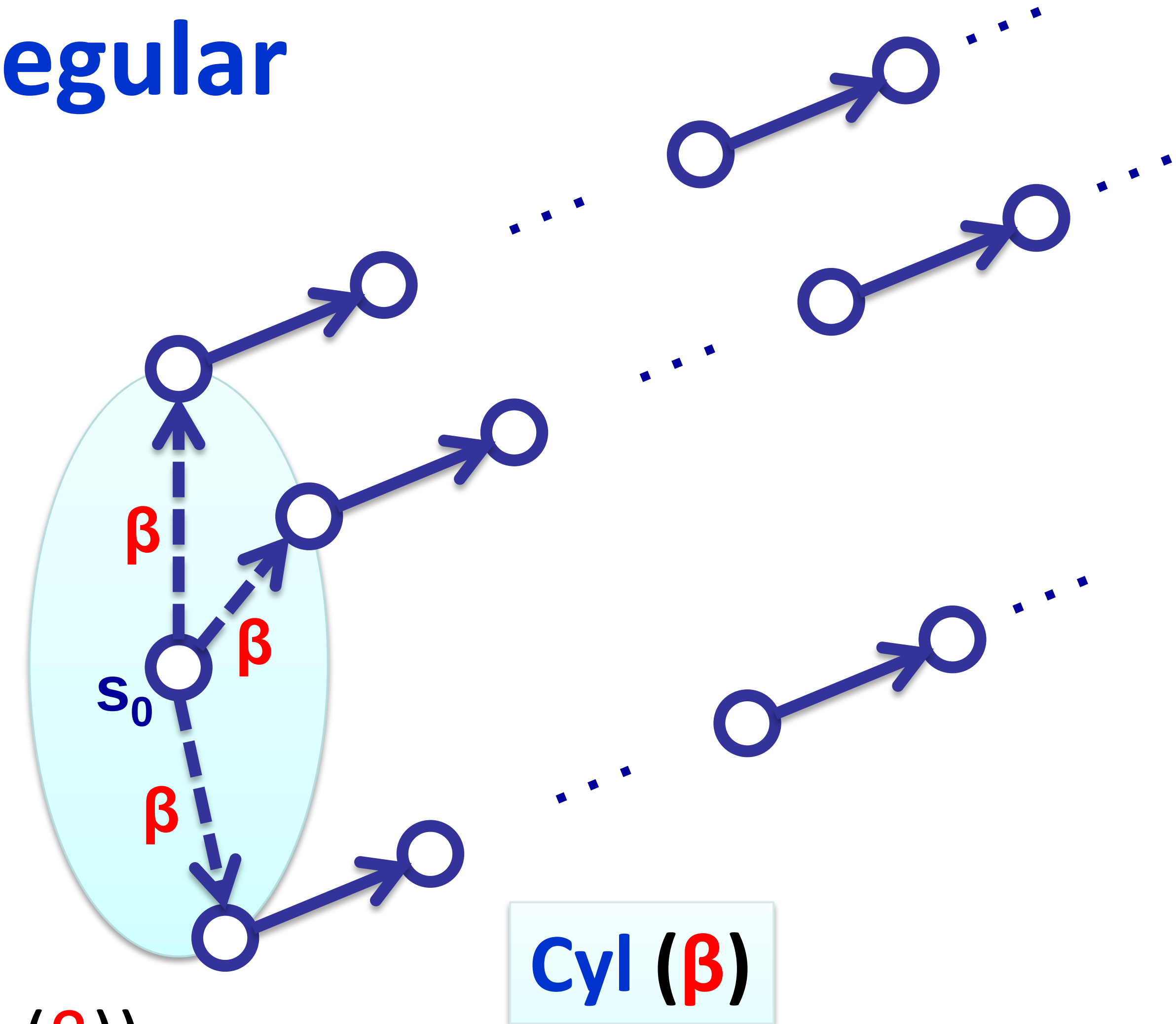
Probabilistic Regular Operator

- Syntax:

  $\{\ \beta\ \}_{op\ \ p}$

  where

  $op \in \{ <, <=, >, >=, = \}$



- Semantics:

  $\mathbf{P}_{op\ \ p}(\beta) = \mu\ (Cyl\ (\beta))\ op\ p =$

  $\sum_{s_0 - a_0\ p_0 \rightarrow s_1 \dots\ s_k - a_k\ p_k \rightarrow s_{k+1}\ \models\ \beta} (p_0 * \dots * p_k)\ op\ p$

Encoding Probabilistic Until Operators

- PCTL "pure" probabilistic Until

  $[\varphi_1 \,U\, \varphi_2]_{>= p} = \{ (\varphi_1? \,.\, \text{true})^* .\, \varphi_2? \}_{>= p}$

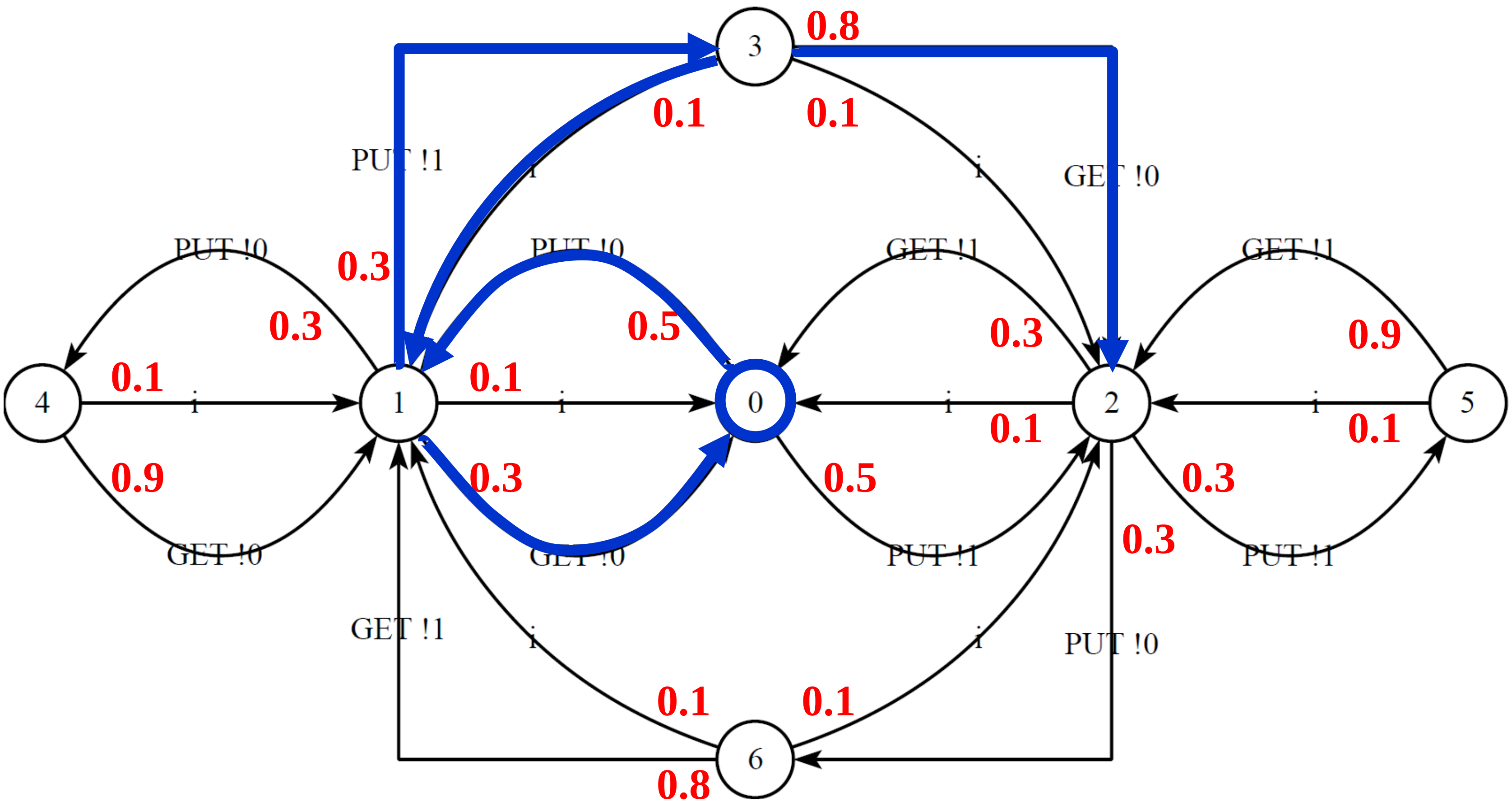
- PACTL "pure" probabilistic Until

  $[\varphi_{1\ \alpha 1} \,U\, \varphi_2]_{>= p} = \{ (\varphi_1?.\, \alpha_1)^* .\, \varphi_2? \}_{>= p}$

  $[\varphi_{1\ \alpha 1} \,U_{\alpha 2}\, \varphi_2]_{>= p} = \{ (\varphi_1?.\, \alpha_1)^* .\, \varphi_1?.\, \alpha_2 \,.\, \varphi_2? \}_{>= p}$

Example

$\{ PUT_0 \, . \, (\text{not } (PUT_0 \text{ or } GET_0))^* . \, GET_0 \} >= 0.25$



On-the-Fly Model Checking

(syntactic steps)

$< PUT_0 \,.\, (\text{not } (PUT_0 \text{ or } GET_0))^* .\, GET_0 > \text{true}$ **MCL**

$X_1 = < PUT_0 \,.\, (\text{not } (PUT_0 \text{ or } GET_0))^* .\, GET_0 > X_2$

$X_2 = \text{true}$

PDLR

- **elimination of regular operators**
- **determinization**

$X_1 = < PUT_0 > X_3$

HMLR

$X_2 = \text{true}$

[Larsen-88]

$X_3 = < GET_0 > X_2 \text{ or } < \text{not } (PUT_0 \text{ or } GET_0) > X_3$

On-the-Fly Model Checking

(semantic steps)

HMLR

$X_1 = < PUT_0 > X_3$

$X_2 = true$

$X_3 = < GET_0 > X_2 \text{ or } < \text{not } (PUT_0 \text{ or } GET_0) > X_3$

- **synchronous products of HMLR and PTS**
- **BES for pruning unreachable suffixes**

LES

$Z_{1,s} = \sum_{s \xrightarrow{PUT_0\, p'} s'} p' * Z_{3,s'}$

$Z_{2,s} = 1.0$

$Z_{3,s} = \sum_{s \xrightarrow{GET_0\, p'} s'} p' * Z_{2,s'} + \sum_{s \xrightarrow{A\, p''} s''} p'' * Z_{3,s''}$

$A \neq PUT_0, GET_0$

→

BES

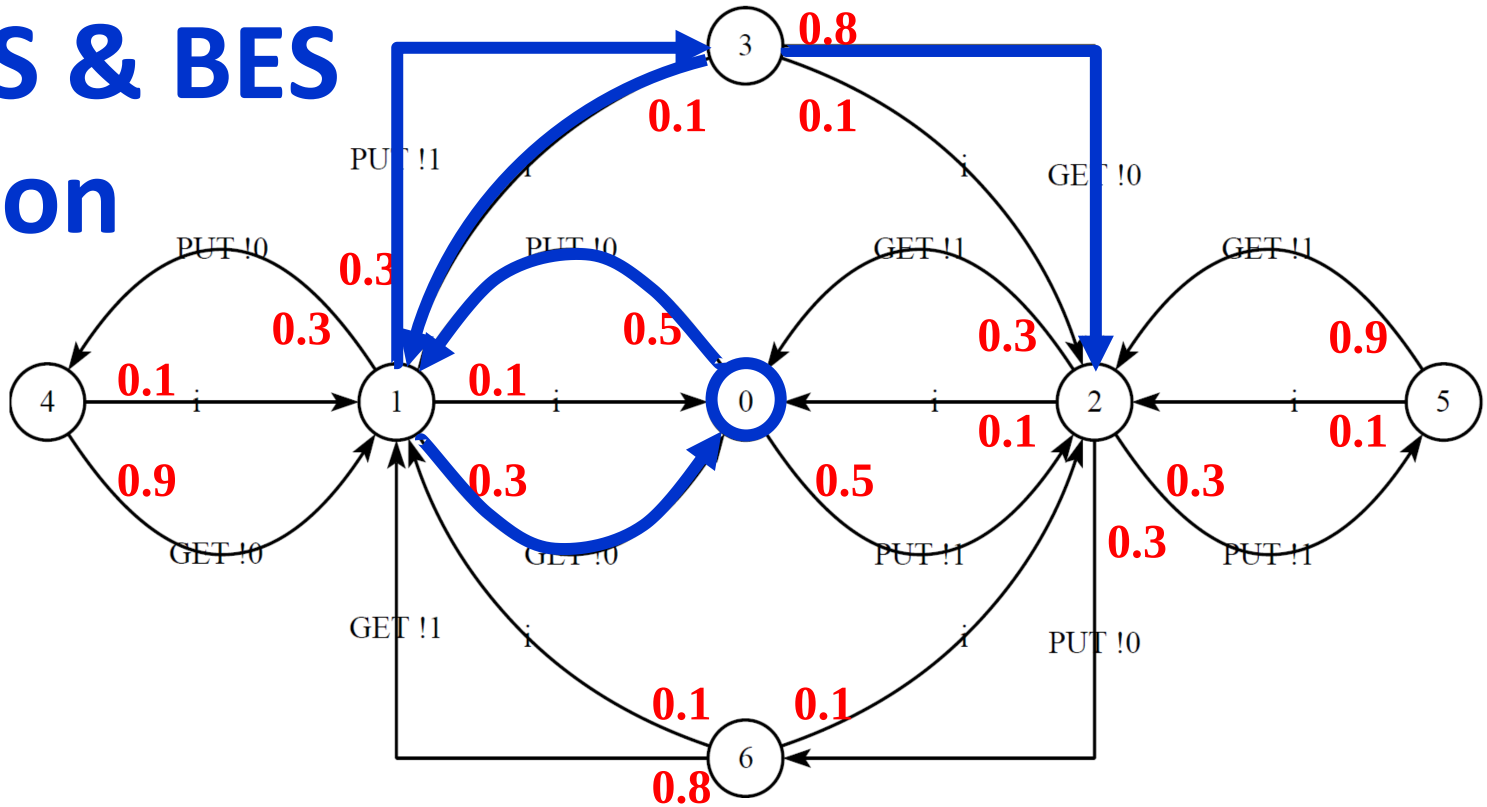
$Y_{1,s} = \bigvee_{s \xrightarrow{PUT_0} s'} Y_{3,s'}$

$Y_{2,s} = true$

$Y_{3,s} = \bigvee_{s \xrightarrow{GET_0} s'} Y_{2,s'} \text{ or } \bigvee_{s \xrightarrow{A} s'} Y_{3,s'}$

$A \neq PUT_0, GET_0$

Local LES & BES Resolution



$$Z_{1,0} = 0.5 * Z_{3,1} = 0.2783$$
$$Z_{2,s} = 1.0$$
$$Z_{3,1} = 0.3 * Z_{2,0} + 0.3 * Z_{3,3}$$
$$Z_{3,3} = 0.8 * Z_{2,2} + 0.1 * Z_{3,1}$$

- **LES represented as Signal Flow Graph**
- **BES represented as Boolean Graph**
- **forward exploration of dependencies**
- **backward propagation / substitution**

Data-Handling Extensions

- PTS for value-passing systems

  data-carrying actions $c\ v_1 \ldots v_n\ p$

- Action predicates of MCL

  { *c* !*e* ?*x*:*T* ... where *b* }

- Data-handling probabilistic regular operator

  $\{\ \beta\ \}_{op\ p}$

  where β

  - Is built over action predicates
  - Enables data capture and propagation

Counter-Based Regular Operators

β ::=	β { *e* }	*iteration e times*
|	β { … *e* }	*iteration ≤ e times*
|	β { e_1 … e_2 }	*iteration between e_1 and e_2 times*
|	for *n*:nat from e_1 to e_2 step e_3 do β end for	*stepwise iteration*

Data-Handling Examples

- Probability of P_i's first access to its critical section

```
{       { NCS ?i:Nat } .
        (not { CS !ENTER ... })* .
        { CS !ENTER !i }
} >= 0.5
```

data capture and propagation

- PCTL full probabilistic Until

$$[\varphi_1 \; U^{<= t} \; \varphi_2]_{>= p} \;=\; \{\, (\varphi_1?.\ \mathrm{true})\ \{\ \ldots\ t\ \}\ .\ \varphi_2?\ \}_{>= p}$$

- *But: reasoning about weights (costs, energy, ...) requires more expressive regular operators*

Generalized Iteration Operators

iteration parameter

return parameter

loop (x:T:=e_0) : (x':T') in

β — body of the loop

end loop

continue (e) — start next iteration and set ***x*** to ***e***

exit (e') — terminate the loop and set ***x'*** to ***e'***

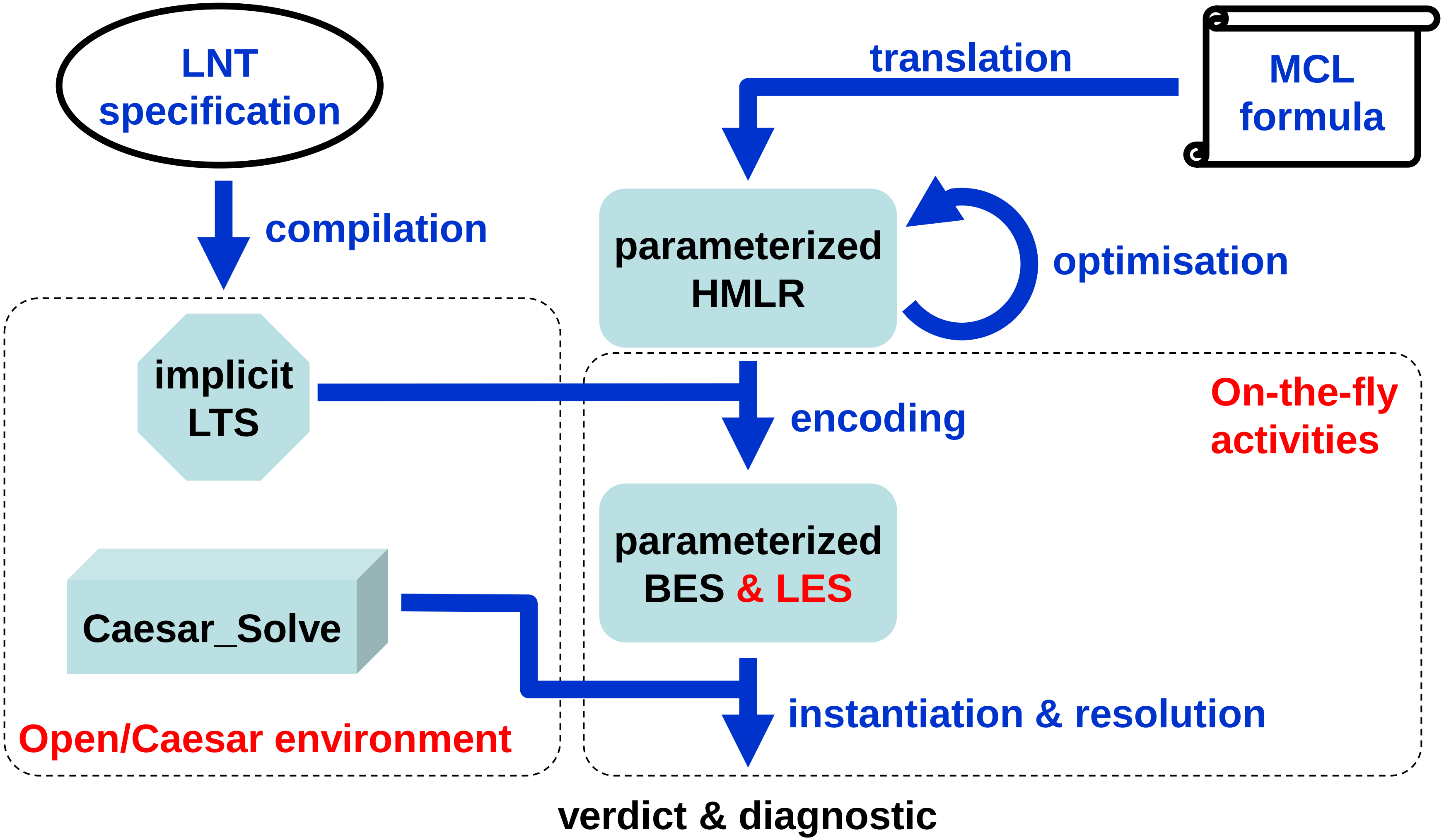
Encoding of Regular Operators

- β* = loop exit | β . continue end loop
- β+ = loop β . (exit | continue) end loop
- β { e_1 … e_2 } = loop (c_1:nat:=e_1, c_2:nat:=$e_2 - e_1$) in

```
    if c1 > 0 then
        β . continue (c1 – 1, c2)
    elsif c2 > 0 then
        exit | β . continue (c1, c2 – 1)
    else
        exit
    end if
end loop
```

On-the-Fly Verification Method

(Evaluator 4.0)



Case Study: Mutual Exclusion Protocols

[Mateescu-Serwe-13]

- N concurrent processes competing for a shared resource
  - Accesses to shared variables (read/write/tas/cas ...)
  - NUMA (caches/local/remote) → various latencies
  - Four sections executed cyclically
    - **Non critical** section // do not use the resource
    - **Entry** section // request access to the resource
    - **Critical** section // access the resource
    - **Exit** section // release the resource
- Five protocols (CLH, MCS, BL, TAS, TTAS) with N ≤ 4
- PTS with equal probabilities for neighbour transitions

Overtaking Degree

(for starvation-free protocols)

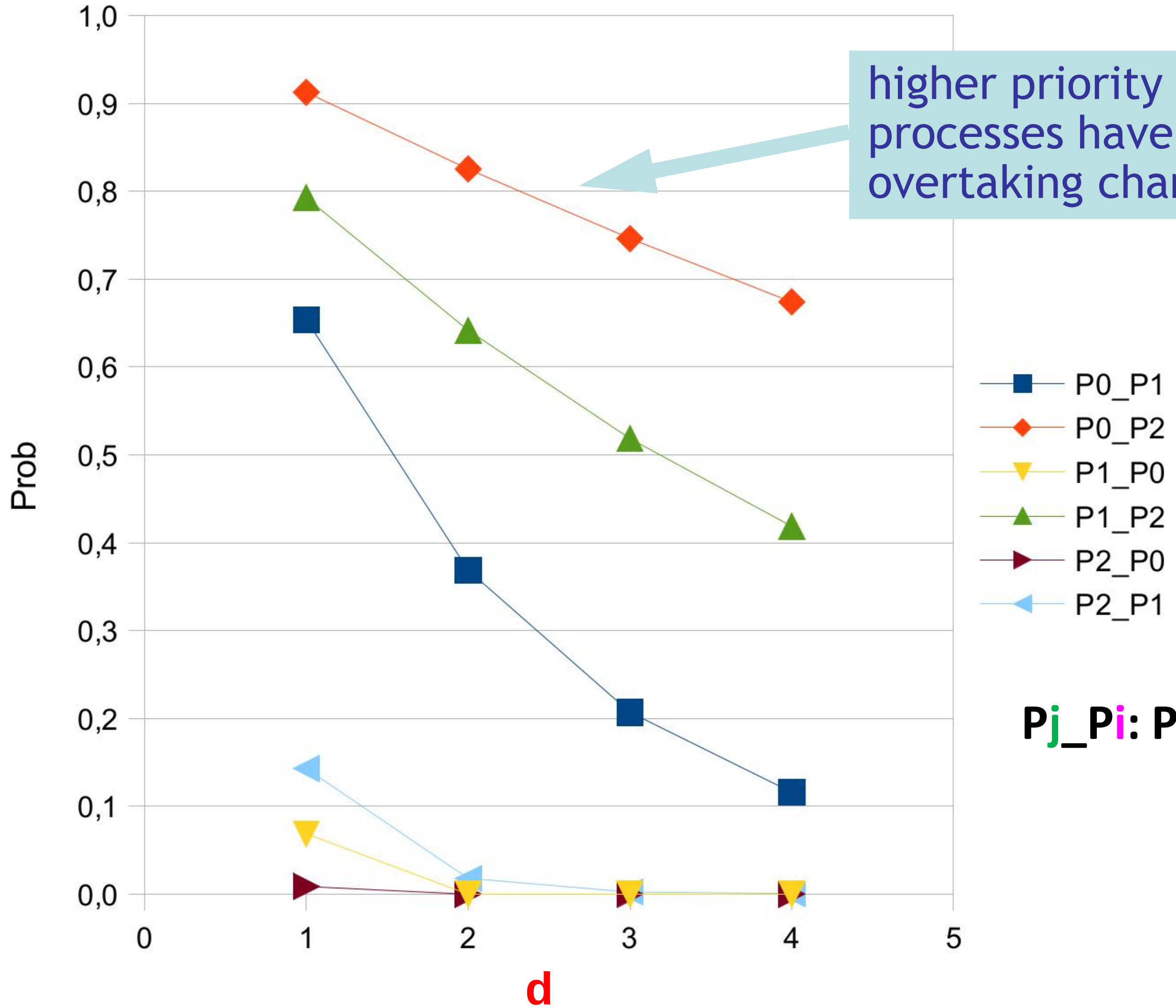
- Sequence with **d** overtakes of P_i by P_j:

```
true*.
NCS (i) . (not MEM (i))*. MEM (i) .
(
   for k:nat from 0 to N-1 do
         (not CS (i))*. MEM (k)
   end for .
   (not CS)* . CS (j)
) { d }
```

- Maximum value of **d**: overtaking degree

Overtaking Degree

(BL protocol – probability that Pj overtakes Pi **d** times)



higher priority (lower index) processes have better overtaking chances

Pj_Pi: Pj overtakes Pi

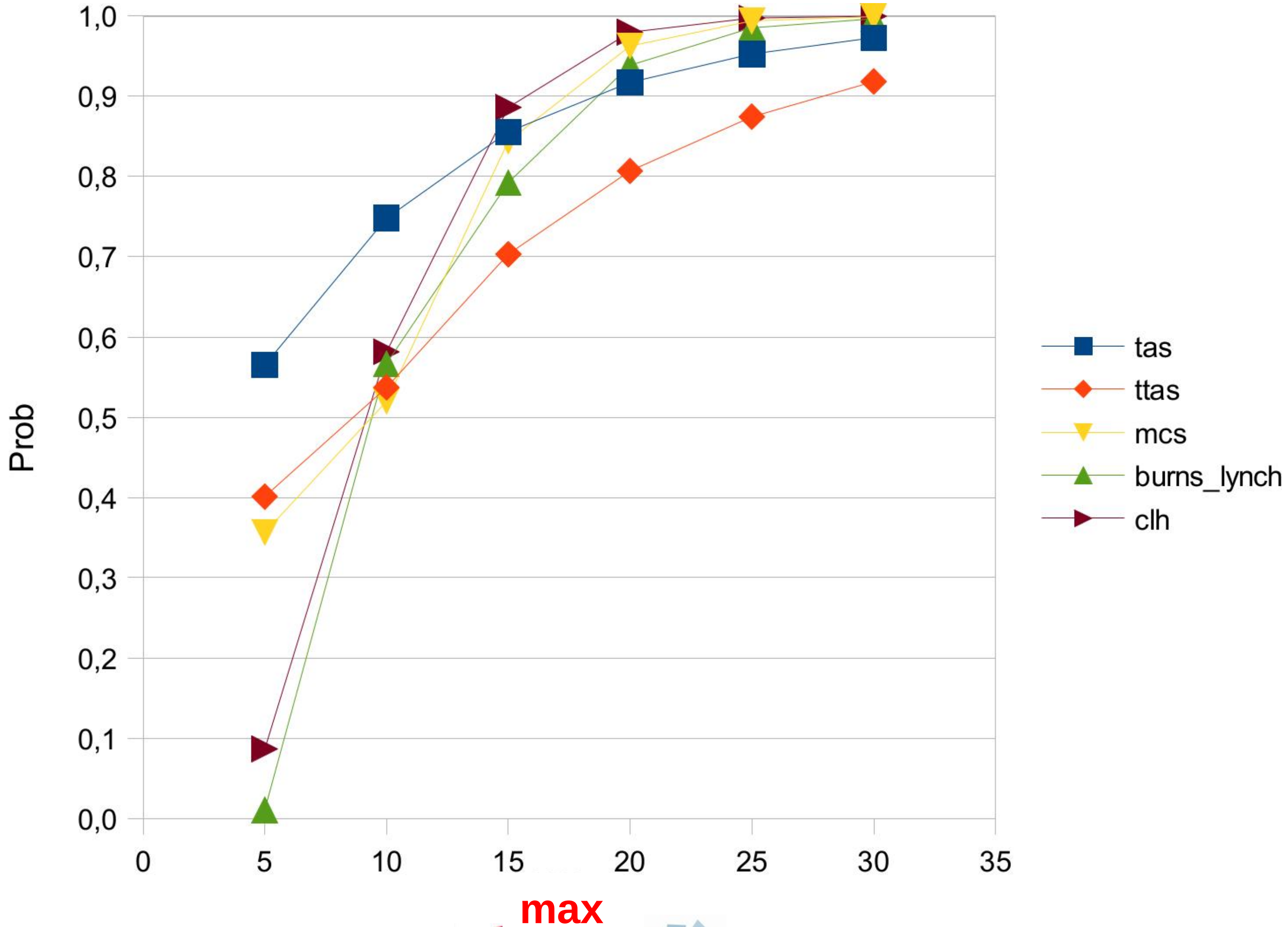
Memory Latency until Critical Section

- Sequence with memory latency **max** for P_i:

```
(not NCS (i))*. NCS (i) .
loop (cost:nat := 0) in
   (not CS (i))*.
   if cost < max then
         { MEM ... ?c:nat !i } . continue (cost + c)
   else
         exit
   end if
end loop .
CS (i)
```

Memory Latency

(probability to access the CS with a memory latency **max**)



Analysis of Peak Cost Sequences

1. specify the desired sequence as β (cost)
2. if sequences with peak cost are finite then
   - Check < β (cost) > true using standard on-the-fly model checking
   - Use dichotomic search to determine peak cost
   - Check { β (peak) } $_{op\ p}$ to get the probability of peak cost sequence
3. else
   - Check { β (cost) } $_{op\ p}$ for variable cost values

Perspectives

- Extend MCL and the model checker
  - User-defined types and functions
  - Connect to IPCs [Coste-Hermanns-et-al-10]
  - Extend the approach to handle infinite sequences using the < β > @ operator of MCL
- Enhance the back-end verification engine
  - Distributed resolution of BESs
  - Connection to the distributed MUMPS LES solver
- Further experiments
  - Compare with (explicit-state) PRISM on PCTL formulas

Thank you

Further information

http://cadp.inria.fr