Benchmarking Model Checkers with Distributed Algorithms

Étienne Couloumа-Dupont

IPB ENSEIRB MATMECA BORDEAUX

Laboratoire d'Informatique Fondamentale

November 24, 2011

Introduction

LastVoting

Simulator

Benchmarks

Panda Project

Industrial Benchmark Airbus Code

Academic Benchmark Paxos Algorithm

The Consensus Problem

- Context: Distributed Systems
- *Consensus* is the problem of making processes agree on a common value in spite of faults

The Consensus Problem

An Algorithms solves Consensus if and only if it satifies the following conditions:

- *Integrity*: Any decision value is the proposed value of some process.
- *Agreement*: No two different values are decided.
- *Termination*: All process eventually decide.

Consensus Algorithm with industrial applications

Chubby (Google, 2006) [Bur06][CGR07] implements a fault tolerant data-base:

- Fault-tolerance is achieved through replication
- Consistency is achieved using Paxos Algorithm

Paxos

- Family of algorithms built to solve Consensus
- First published by Leslie Lamport in 1998 [Lam98]
- Termination is not guaranteed without additional assumptions on liveness
- Safety is guaranteed

Several implementations in different models:

ChandraToueg	Attach to each process in the system a *failure detector*
LastVoting	Synchronous communication (*rounds*)

Related Works

This algorithm has been intensively studied

- Fuzzati [Fuz08]: proof of Paxos and *ChandraToueg* with rewriting rules
- Küfner et al.: assisted proofs in `Isabelle`
- Tsuchiya & Schiper: model-checking of *LastVoting*
  - SAT solver: up to 10 processes
  - Traditional tools: up to 4 processes

LastVoting : Hypothesis

Assumptions for the environment:

- with transmission faults : in each round, the adversary chooses a set of edges in which all messages will be correctly transmitted
- pseudo-synchronous: any message which is not received in the same round as the one during which it was send is thrown out by the process which receives it
- complete network: simplifies the way algorithms are expressed
- each process has a unique identity

LastVoting : Hypothesis

We assume that each process *p* has at any time access to the following pieces of information:

- The round *r* in which it is
- The coordinator at round *r*

The Algorithm's Procedure

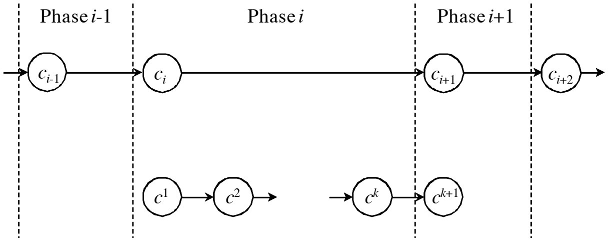


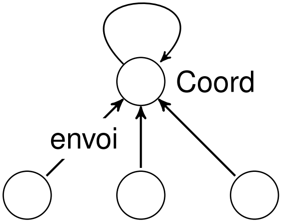
Figure: Transitions of configurations at the phase level (top) and at the round level (bottom) [TS11].

The Algorithm's Procedure

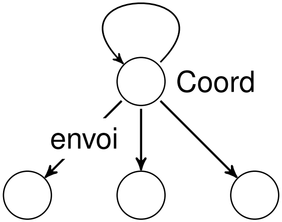
In each round r, each process p:

- can send a messages according to a sending function S_p^r, depending on the process' state
- then can compute a new state according to a state transition function T_p^r, depending on the messages received

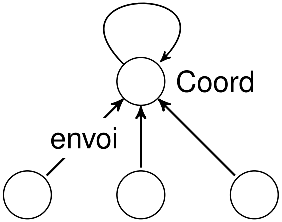
Selection ($r = 4\phi - 3$)



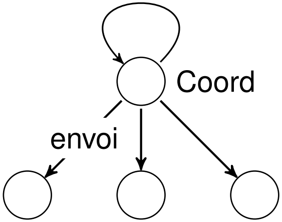
Commit ($r = 4\phi - 2$)



Ack ($r = 4\phi - 1$)



Decision ($r = 4\phi$)



Notations

- $co = Coord(\phi)$
- $\mathfrak{D}(\phi) = \{p \in \Pi \mid d_p^{4\phi-3} \neq ?\}$
- for all $p \in \Pi$ and $r \in [4\phi - 3, 4\phi - 1]$, $RCV(p, r)$ is the set of processes from which p receives a *non empty* message in round r

```
Initialization :
    $x_p \in Val$, initially $v_p$
    // $v_p$ is the proposed value of $p$.
    $vote_p \in Val \cup \{?\}$, initially ?
    // $Val$ is the set of values that may be proposed.
    $commit_p$ a Boolean, initially false
    $ready_p$ a Boolean, initially false
    $ts_p \in \mathbb{N}$, initially 0
```

The Algorithm

```
**Round** $r = 4\phi - 3$ :                    // $\#RCV(co, 4\phi - 3) > n/2$
    $S_p^r$ :
        **send** $\langle x_p, ts_p \rangle$ to $Coord(p, \phi)$

    $T_p^r$ :
        **if** $p = Coord(p, \phi)$ **and** *number of* $\langle \nu, \theta \rangle$ *received* $> n/2$
        **then**
            let be the largest $\overline{\theta}$ from $\langle \nu, \theta \rangle$ received;
            $vote_p :=$ one $\nu$ such that $\langle \nu, \overline{\theta} \rangle$ is received;
            $commit_p := \texttt{true}$;
```

```
Round $r = 4\phi - 2$ : // $\#\{p \in \Pi \mid co \in RCV(p, 4\phi - 2)\} > n/2$
    $S_p^r$ :
        if $p = Coord(p, \phi)$ and $commit_p$ then
            send $\langle vote_p \rangle$ to all processes;

    $T_p^r$ :
        if $received\langle v \rangle$ $from$ $Coord(p, \phi)$ then
            $x_p := v$;
            $ts_p := \phi$;
```

```
**Round** $r = 4\phi - 1$ :                    // $\#RCV(co, 4\phi - 1) > n/2$
    $S_p^r$ :
        **if** $ts_p = \phi$ **then**
            **send** $\langle ack \rangle$ to $Coord(p, \phi)$;
    $T_p^r$ :
        **if** $p = Coord(p, \phi)$ **and** *number of* $\langle ack \rangle$ *received* $> n/2$
        **then**
            $ready_p := \texttt{true}$;
```

```
Round $r = 4\phi$ :                    // $\forall p \in \Pi \setminus \mathfrak{D}(\phi) : co \in RCV(p, 4\phi)$
    $S_p^r$ :
        if $Coord(p, \phi)$ and $ready_p$ then
            send $\langle vote_p \rangle$ to all processes;

    $T_p^r$ :
        if $received\langle v \rangle$ from $Coord(p, \phi)$ then
            DECIDE $(v)$
        if $p = Coord(p, \phi)$ then
            $ready_p :=$ false;
            $commit_p :=$ false;
```

Terminology

Given a partial execution, we define

0-Valence	if the value 0 is to be decided
1-Valence	if the value 1 is to be decided
Univalence	if one the above holds
Bivalence	if none of the above holds

Formally, Univalence is defined for *LastVoting* as follows:

$$\begin{aligned}\exists\, v \in Val,\ \exists\, \mathfrak{P} \subseteq \Pi :&\\ &\wedge \#\mathfrak{P} \geq n/2\\ &\wedge\ \mathfrak{P} = \{p \in \Pi \,|\, pr_p = v\} \wedge (\forall q \in \Pi \setminus \mathfrak{P} : ts_q < ts_p)\end{aligned}$$

Termination

Termination holds iff $P^{sync}(\phi)$ eventually holds:

$P^{sync}(\phi) \triangleq \exists \phi > 0, \exists co \in \Pi$:

*ronde*1 : $\wedge \ \#RCV(co, 4\phi - 3) > n/2 \qquad (CPROP(\phi))$

Termination

Termination holds iff $P^{sync}(\phi)$ eventually holds:

$$P^{sync}(\phi) \triangleq \exists \phi > 0, \exists co \in \Pi :$$

$$\begin{array}{lll} ronde1: & \wedge\ \#RCV(co, 4\phi - 3) > n/2 & (CPROP(\phi)) \\ ronde2: & \wedge\ \#\{p \in \Pi \mid co \in RCV(p, 4\phi - 2)\} > n/2 & (BVOTE(\phi)) \end{array}$$

Termination

Termination holds iff $P^{sync}(\phi)$ eventually holds:

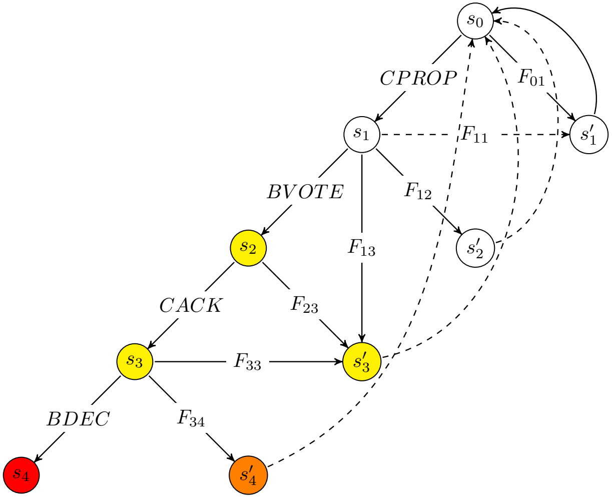
$$
\begin{array}{lll}
\multicolumn{3}{l}{P^{sync}(\phi) \triangleq \exists \phi > 0, \exists co \in \Pi :} \\
ronde1: & \wedge \; \#RCV(co, 4\phi - 3) > n/2 & (CPROP(\phi)) \\
ronde2: & \wedge \; \#\{p \in \Pi \mid co \in RCV(p, 4\phi - 2)\} > n/2 & (BVOTE(\phi)) \\
ronde3: & \wedge \; \#RCV(co, 4\phi - 1) > n/2 & (CACK(\phi))
\end{array}
$$

Termination

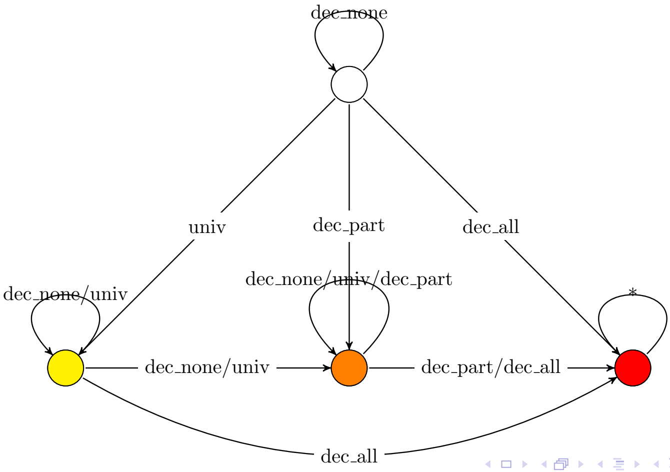
Termination holds iff $P^{sync}(\phi)$ eventually holds:

$$
\begin{array}{lll}
\multicolumn{3}{l}{P^{sync}(\phi) \triangleq \exists \phi > 0, \exists co \in \Pi :} \\
\textit{ronde1} : & \wedge \ \#RCV(co, 4\phi - 3) > n/2 & (CPROP(\phi)) \\
\textit{ronde2} : & \wedge \ \#\{p \in \Pi \mid co \in RCV(p, 4\phi - 2)\} > n/2 & (BVOTE(\phi)) \\
\textit{ronde3} : & \wedge \ \#RCV(co, 4\phi - 1) > n/2 & (CACK(\phi)) \\
\textit{ronde4} : & \wedge \ \forall p \in \Pi \setminus \mathfrak{D}(\phi) : co \in RCV(p, 4\phi) & (BDEC(\phi))
\end{array}
$$

Analysis of the Algorithm



Execution-tree (graph. . .)

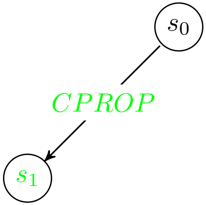


Example of execution

$\phi = 1$		
$\downarrow$		
p_1	p_2	p_3
$pr_1 = 0$ $ts_1 = 0$ $d_1 = ?$	$pr_2 = 0$ $ts_2 = 0$ $d_2 = ?$	$pr_3 = 1$ $ts_3 = 0$ $d_3 = ?$

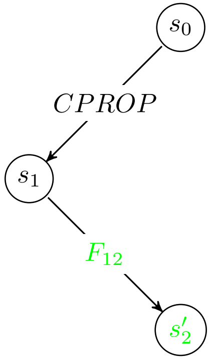
Analysis of the Algorithm

$\phi = 1$		
$\downarrow$		
p_1	p_2	p_3
$ho_1 = 1$	$ho_2 = 0$	$ho_3 = 1$
$pr_1 = 0$	$pr_2 = 0$	$pr_3 = 1$
$ts_1 = 0$	$ts_2 = 0$	$ts_3 = 0$
$d_1 = ?$	$d_2 = ?$	$d_3 = ?$


$$\begin{tikzcd} & s_0 \arrow[dl, "CPROP"] \\ s_1 & \end{tikzcd}$$

Analysis of the Algorithm

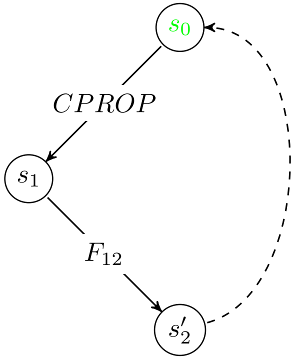
$\phi = 1$		
$\downarrow$		
p_1	p_2	p_3
$ho_1 = 0$	$ho_2 = 0$	$ho_3 = 1$
$pr_1 = 0$	$pr_2 = 0$	$pr_3 = 1$
$ts_1 = 0$	$ts_2 = 0$	$ts_3 = 1$
$d_1 = ?$	$d_2 = ?$	$d_3 = ?$



$$\begin{tikzcd} s_0 \arrow[d, "CPROP"] \\ s_1 \arrow[d, "F_{12}"] \\ s_2' \end{tikzcd}$$

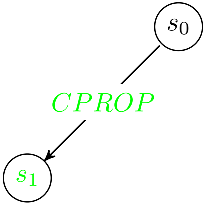
Analysis of the Algorithm

$\phi = 2$		
	$\downarrow$	
p_1	p_2	p_3
$pr_1 = 0$	$pr_2 = 0$	$pr_3 = 1$
$ts_1 = 0$	$ts_2 = 0$	$ts_3 = 1$
$d_1 = ?$	$d_2 = ?$	$d_3 = ?$


$$\begin{tikzcd} & s_0 \arrow[dl, "CPROP"'] \\ s_1 \arrow[dr, "F_{12}"'] & \\ & s'_2 \arrow[uu, dashed, bend right] \end{tikzcd}$$

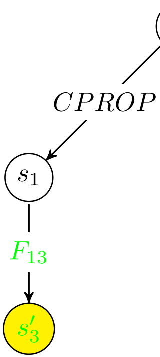
Analysis of the Algorithm

$\phi = 2$		
	$\downarrow$	
p_1	p_2	p_3
$ho_1 = 0$ $pr_1 = 0$ $ts_1 = 0$ $d_1 = ?$	$ho_2 = 1$ $pr_2 = 0$ $ts_2 = 0$ $d_2 = ?$	$ho_3 = 1$ $pr_3 = 1$ $ts_3 = 1$ $d_3 = ?$


$$\begin{tikzcd} & s_0 \arrow[dl, "CPROP"'] \\ s_1 & \end{tikzcd}$$

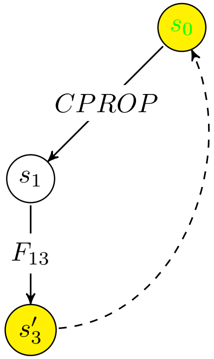
Analysis of the Algorithm

$\phi = 2$		
	$\downarrow$	
p_1	p_2	p_3
$ho_1 = 1$	$ho_2 = 0$	$ho_3 = 0$
$pr_1 = 1$	$pr_2 = 0$	$pr_3 = 1$
$ts_1 = 2$	$ts_2 = 0$	$ts_3 = 1$
$d_1 = ?$	$d_2 = ?$	$d_3 = ?$


$$\begin{tikzcd} s_0 \arrow[d, "CPROP"] \\ s_1 \arrow[d, "F_{13}"] \\ s_3' \end{tikzcd}$$

Analysis of the Algorithm

$\phi = 3$		
		$\downarrow$
p_1	p_2	p_3
$pr_1 = 1$ $ts_1 = 2$	$pr_2 = 0$ $ts_2 = 0$	$pr_3 = 1$ $ts_3 = 1$
$d_1 = ?$	$d_2 = ?$	$d_3 = ?$

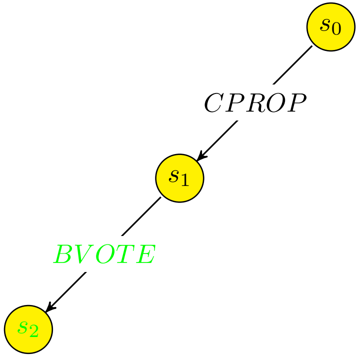

$$\begin{tikzcd} & s_0 \arrow[ld, "CPROP"'] \\ s_1 \arrow[d, "F_{13}"] & \\ s'_3 \arrow[ruu, dashed, bend right] & \end{tikzcd}$$

Analysis of the Algorithm

$\phi = 3$		
		$\downarrow$
p_1	p_2	p_3
$ho_1 = 1$	$ho_2 = 1$	$ho_3 = 0$
$pr_1 = 1$ $ts_1 = 2$	$pr_2 = 0$ $ts_2 = 0$	$pr_3 = 1$ $ts_3 = 1$
$d_1 = ?$	$d_2 = ?$	$d_3 = ?$

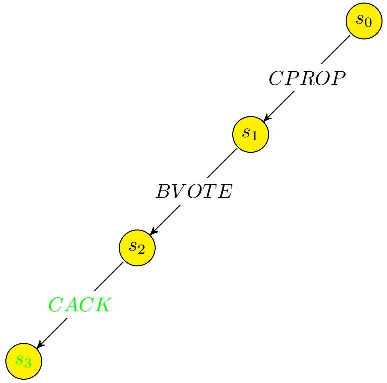
$$\begin{tikzcd} & s_0 \arrow[dl, "CPROP"] \\ s_1 & \end{tikzcd}$$

$\phi = 3$		
		$\downarrow$
p_1	p_2	p_3
$ho_1 = 0$	$ho_2 = 1$	$ho_3 = 1$
$pr_1 = 1$	$pr_2 = 1$	$pr_3 = 1$
$ts_1 = 2$	$ts_2 = 3$	$ts_3 = 3$
$d_1 = ?$	$d_2 = ?$	$d_3 = ?$



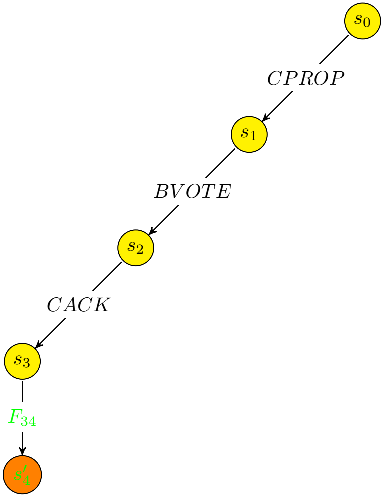
Analysis of the Algorithm

$\phi = 3$		
		$\downarrow$
p_1	p_2	p_3
$ho_1 = 1$	$ho_2 = 0$	$ho_3 = 1$
$pr_1 = 1$ $ts_1 = 2$	$pr_2 = 1$ $ts_2 = 3$	$pr_3 = 1$ $ts_3 = 3$
$d_1 = ?$	$d_2 = ?$	$d_3 = ?$



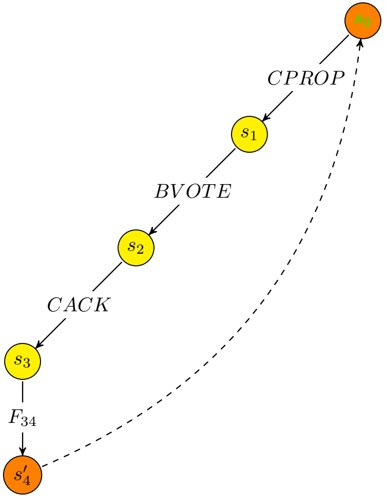
Analysis of the Algorithm

$\phi = 3$		
		$\downarrow$
p_1	p_2	p_3
$ho_1 = 1$	$ho_2 = 0$	$ho_3 = 1$
$pr_1 = 1$ $ts_1 = 2$	$pr_2 = 1$ $ts_2 = 3$	$pr_3 = 1$ $ts_3 = 3$
$d_1 = ?$	$d_2 = ?$	$d_3 = 1$


$$\begin{tikzcd} s_0 \arrow[d, "CPROP"] \\ s_1 \arrow[d, "BVOTE"] \\ s_2 \arrow[d, "CACK"] \\ s_3 \arrow[d, "F_{34}"] \\ s'_4 \end{tikzcd}$$

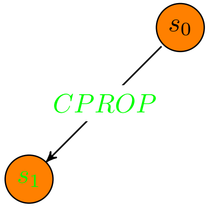
Analysis of the Algorithm

$\phi = 4$		
$\downarrow$		
p_1	p_2	p_3
$pr_1 = 1$ $ts_1 = 2$	$pr_2 = 1$ $ts_2 = 3$	$pr_3 = 1$ $ts_3 = 3$
$d_1 = ?$	$d_2 = ?$	$d_3 = 1$



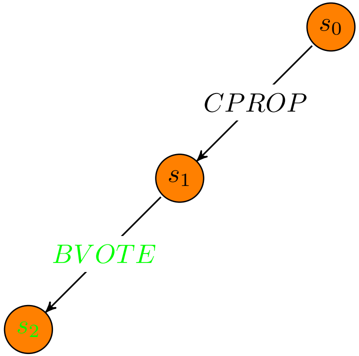
Analysis of the Algorithm

$\phi = 4$		
$\downarrow$		
p_1	p_2	p_3
$ho_1 = 1$	$ho_2 = 1$	$ho_3 = 1$
$pr_1 = 1$ $ts_1 = 2$	$pr_2 = 1$ $ts_2 = 3$	$pr_3 = 1$ $ts_3 = 3$
$d_1 =?$	$d_2 =?$	$d_3 = 1$

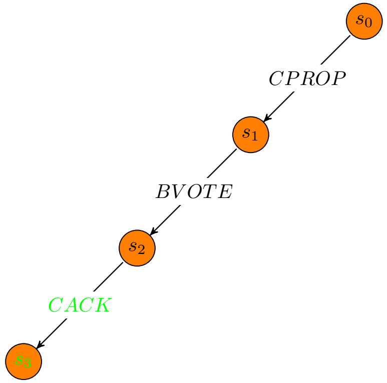


Analysis of the Algorithm

$\phi = 4$		
$\downarrow$		
p_1	p_2	p_3
$ho_1 = 1$	$ho_2 = 0$	$ho_3 = 1$
$pr_1 = 1$	$pr_2 = 1$	$pr_3 = 1$
$ts_1 = 4$	$ts_2 = 3$	$ts_3 = 4$
$d_1 = ?$	$d_2 = ?$	$d_3 = 1$

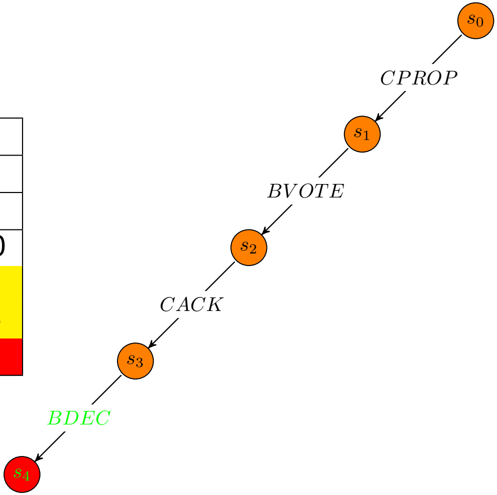

$$\begin{tikzcd} s_0 \arrow[d, "CPROP"] \\ s_1 \arrow[d, "BVOTE"] \\ s_2 \end{tikzcd}$$

$\phi = 4$		
$\downarrow$		
p_1	p_2	p_3
$ho_1 = 1$	$ho_2 = 0$	$ho_3 = 1$
$pr_1 = 1$ $ts_1 = 4$	$pr_2 = 1$ $ts_2 = 3$	$pr_3 = 1$ $ts_3 = 4$
$d_1 = ?$	$d_2 = ?$	$d_3 = 1$



Analysis of the Algorithm

$\phi = 4$		
$\downarrow$		
p_1	p_2	p_3
$ho_1 = 1$	$ho_2 = 1$	$ho_3 = 0$
$pr_1 = 1$ $ts_1 = 4$	$pr_2 = 1$ $ts_2 = 3$	$pr_3 = 1$ $ts_3 = 4$
$d_1 = 1$	$d_2 = 1$	$d_3 = 1$



Comparison with the asynchronous version

- Each process has a infinite set of integers which is disjoint from the sets of the other processes
- This set is used to uniquely identify the consensus requests
- In the asynchronous version, the coordinator first broadcasts the request number of the new request
- The last broadcast is a flooding broadcast procedure

Computability of Paxos

- Paxos does not terminate [FLP85][Fuz08]
- Optimal condition not known for consensus algorithms

Fuzzati's proof of Paxos

A general pattern of rule:

$$(\textsc{Rule})\ \frac{\text{Condition(s)}}{\text{Action(s) or Event(s)}}$$

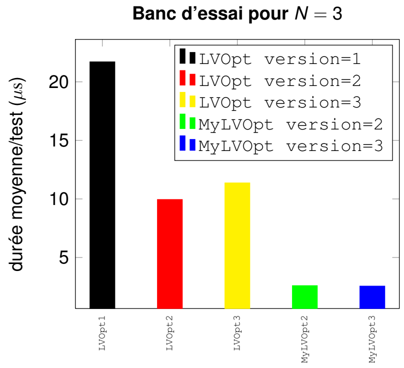
Example of rule from Fuzzati's Paxos' rules:

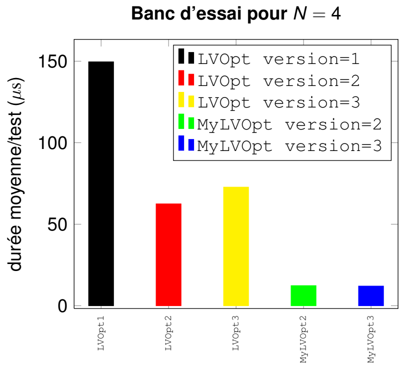
$$(\text{CRASH})\ \frac{\mathrm{S}(i) = (a, r, p, b, (\top, \iota), \bot)}{\langle \mathrm{B}, \mathrm{C}, \mathrm{S} \rangle \rightarrow \langle \mathrm{B}, \mathrm{C}, \mathrm{S}\{i \mapsto (a, r, p, b, (\bot, \iota), \bot)\} \rangle}$$

Motivations

- Correct Algorithms
- Benchmark Algorithms’ performances

Benchmark


Banc d'essai pour $N = 3$
durée moyenne/test (μs)
20
15
10
5
LVOpt version=1
LVOpt version=2
LVOpt version=3
MyLVOpt version=2
MyLVOpt version=3
LVOpt1
LVOpt2
LVOpt3
MyLVOpt2
MyLVOpt3


Banc d'essai pour $N = 4$
durée moyenne/test (μs)
150
100
50
0
LVOpt version=1
LVOpt version=2
LVOpt version=3
MyLVOpt version=2
MyLVOpt version=3
LVOpt1
LVOpt2
LVOpt3
MyLVOpt2
MyLVOpt3

Introduction

LastVoting

Simulator

Benchmarks

Motivations

Benchmarking with Paxos

- Tsuchiya & Schiper:
  - SAT Solvers: up to 10 processes
  - NuSMV: up to 4 processes
  - SPIN: up to 3 processes

Motivations

Benchmarking with Paxos

- Tsuchiya & Schiper:
  - SAT Solvers: up to 10 processes
  - NuSMV: up to 4 processes
  - SPIN: up to 3 processes
- Our work in progress:
  - POEM (Peter Niebert, Marseille):
    frontend for different tools with partial order techniques preprocessing
  - ALCOOL (CEA/List, Panda):
    Topological techniques

Toward Benchmarking of Model-Checkers

- Number n of processes of the system

Toward Benchmarking of Model-Checkers

- Number *n* of processes of the system
- Number of losses for the system:
  - for one round
  - for one phase
  - for *k* phases
  - for one execution
- Number of losses per process:
  - for one phase
  - for *k* phases
  - for one execution

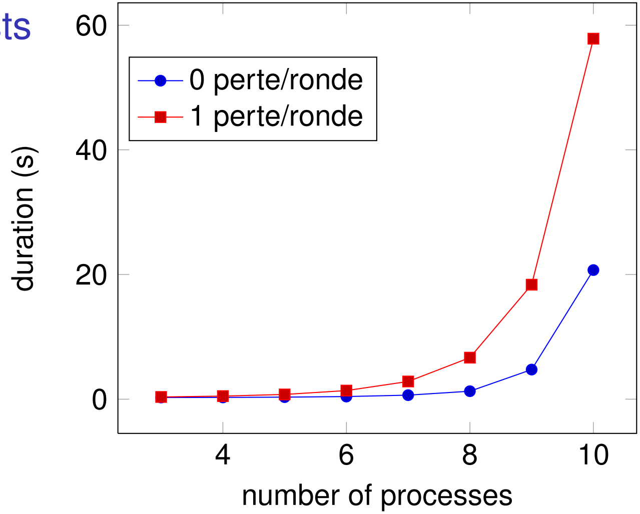
POEM

Checking is done with `IF` requests (first order logic):

`(d[0]<>NOT_DEF) and (d[1]<>NOT_DEF) and (d[0]<>d[1]`

Current problem: false positives (Bug in the frontend?)

First Tests

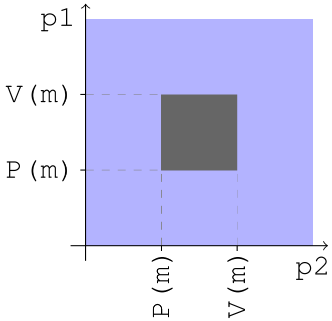

0 perte/ronde
1 perte/ronde
duration (s)
0
20
40
60
4
6
8
10
number of processes

ALCOOL

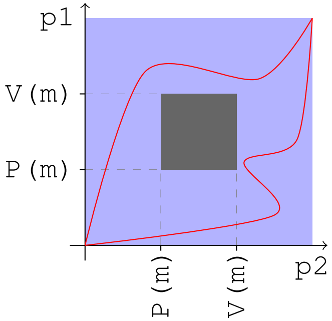
- ALCOOL can detect deadlocks and forbidden zones
- We transform the properties which are to be verified into deadlock properties
- Example:
  B_0 and B_1 two n-ary synchronization barriers
  Agreement $\longrightarrow$ process p waits on B_{d_p}
- Current problem: false positive (feature)

Example 2

```
#mutex m
processes:
p1 = P(m).V(m)
p2 = P(m).V(m)
init:
p1 p2
```

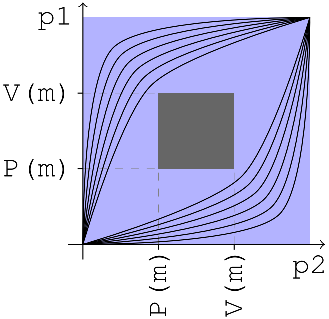


ALCOOL



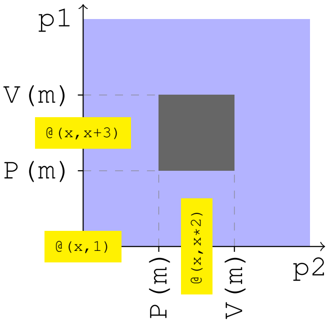
Basically only 2 executions:

- `p1.p2`
- `p2.p1`



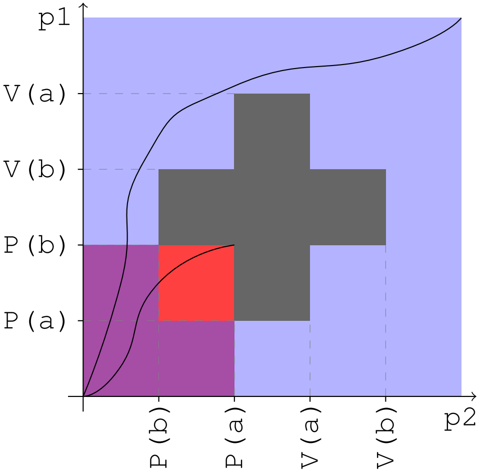
ALCOOL

- `p1.p2 :   x=8`
- `p2.p1 :   x=5`



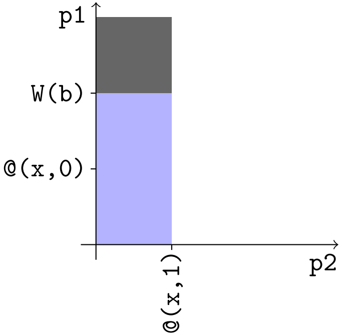
Example 3

```
#mutex a b
processes:
p1 = P(a).P(b).V(b).V(a)
p2 = P(b).P(a).V(a).V(b)
init:
p1 p2
```



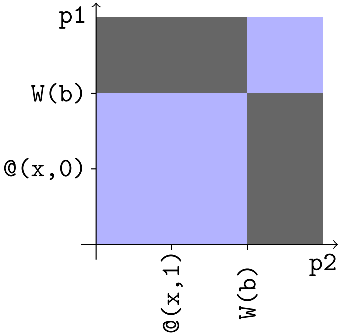
Example 4

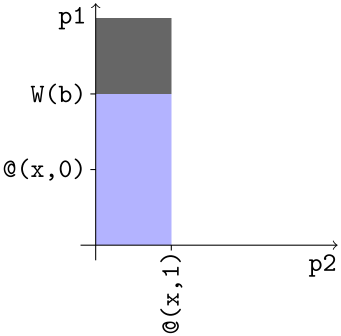
```
#synchronization 2 b
processes:
p1 = @(x,1).
(W(b) + [x==1] + void)
p2 = @(x,0).W(b)
init:
p1 p2
```

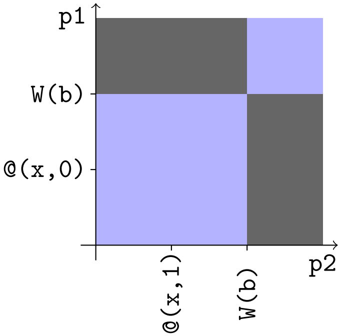


Example 4

```
#synchronization 2 b
processes:
p1 = @(x,1).W(b)
p2 = @(x,0).W(b)
init:
p1 p2
```

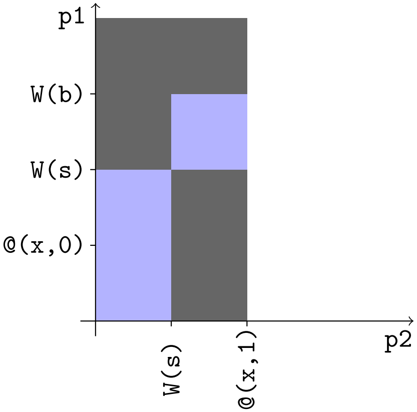



p1
W(b)
@(x,0)
@(x,1)
p2


p1
W(b)
@(x,0)
@(x,1)
W(b)
p2

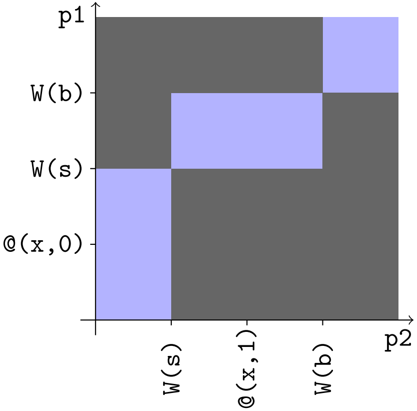
Example 5

```
#synchronization 2 b s
processes:
p1 = void.W(s).@(x,1).
(W(b) + [x==1] + void)
p2 = @(x,0).W(s).W(b)
init:
p1 p2
```

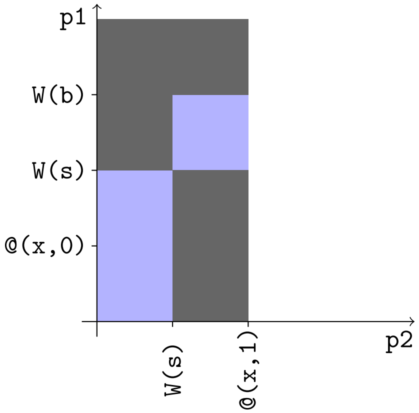


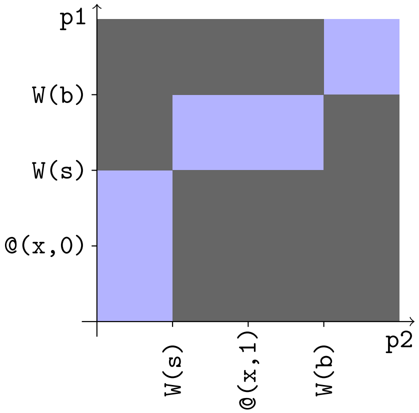
Example 5

```
#synchronization 2 b s
processes:
p1 = W(s).@(x,1).W(b)
p2 = @(x,0).W(s).W(b)
init:
p1 p2
```



ALCOOL





Benchmarks: ALCOOL

- Analysis of numerical variables
- Analysis of potential deadlocks
- ALCOOL +POEM?

BURROWS, MIKE: *The Chubby lock service for loosely-coupled distributed systems*. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 335–350, Berkeley, CA, USA, 2006. USENIX Association.

CHANDRA, TUSHAR D., ROBERT GRIESEMER and JOSHUA REDSTONE: *Paxos made live: an engineering perspective*. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, PODC '07, pages 398–407, New York, NY, USA, 2007. ACM.

FISCHER, MICHAEL J., NANCY A. LYNCH and MICHAEL S. PATERSON: *Impossibility of distributed consensus with one faulty process*. J. ACM, 32:374–382, April 1985.

- Fuzzati, Rachele: *A formal approach to fault tolerant distributed consensus*.
  PhD thesis, École Polytechnique Fédérale de Lausanne, Lausanne, 2008.
- Lamport, Leslie: *The part-time parliament*.
  ACM Transactions on Computer Systems, 16:133–169, 1998.
- Tsuchiya, Tatsuhiro and André Schiper: *Verification of consensus algorithms using satisfiability solving*.
  Distributed Computing, 23:341–358, 2011.