Towards synthesis of distributed algorithms with SMT solvers

C. Delporte-Gallet, H. Fauconnier, Y. Jurski, F. Laroussinie and
Arnaud Sangnier

IRIF - Univ Paris Diderot

ANR DESCARTES 9th October 2019

Conception of distributed algorithms

Why is it difficult to conceive a distributed algorithm ?

1. For some distributed problems, there is no algorithm
   - There does not exist a wait free algorithm for consensus
2. To solve this issue, one might consider new execution contexts
   - For instance, wait free vs obstruction free
   - Intuitions behind such contexts can be hard to get
3. The proofs of correctness can be tedious to achieve
   - Due to the interleaving, many behavior to consider
   - Invariants are not easy to express
   - Automatic verification methods are hard to find

Challenges

- Provide tools and techniques to ease the conception of distributed algorithms

In this work

- Focus on algorithms working on shared memory (single writer; multiple reader)
- No other mechanism than write and read to the memory
- Simple distributed problems: consensus and ϵ-agreement
- Propose a model for distributed algorithms and a specification language for ;
  - Classical correctness properties
  - Selecting some specific executions

Investigate whether in some cases, distributed algorithm can be built automatically

Outline

1. Modeling distributed algorithms
2. Using LTL to reason on distributed algorithms
3. Synthesis
4. Experiments

Outline

Process algorithms

- Each process has a local copy of the registers (its current view)
- Each process has a local memory state
  - To behave differently with the same view according to its past behavior
- A process can perform three actions:
  1. Write a data value to its own register **wr**(d)
  2. Read a shared register **re**(k)
  3. Decide a value **dec**(o)
- The action is determined by the local state and the view

A process algorithm

$P = (M, \delta)$ over a set $\mathcal{D}$ of data and in an environment of n processes

- M set of local memory states
- $\delta : M \times \mathcal{D}^n \mapsto M \times Act(\mathcal{D}, n)$
- $\delta_{in} : \mathcal{D} \mapsto M \times Act(\mathcal{D}, n)$ (determine the first action)

Distributed algorithms

A distributed algorithm

$A = P_1 \otimes P_2 \otimes \ldots \otimes P_n$ where each P_i is a process algorithm over a set $\mathcal{D}$ of data and in an environment of n processes.

About the semantics

- A configuration is the local state and the local view of each process + the value of the shared registers
- A write changes the shared register and the local view of the writer
- A read changes only the local view of the reader
- At the moment all interleavings are allowed
- When a process has decided, it cannot change its state nor its view

Example

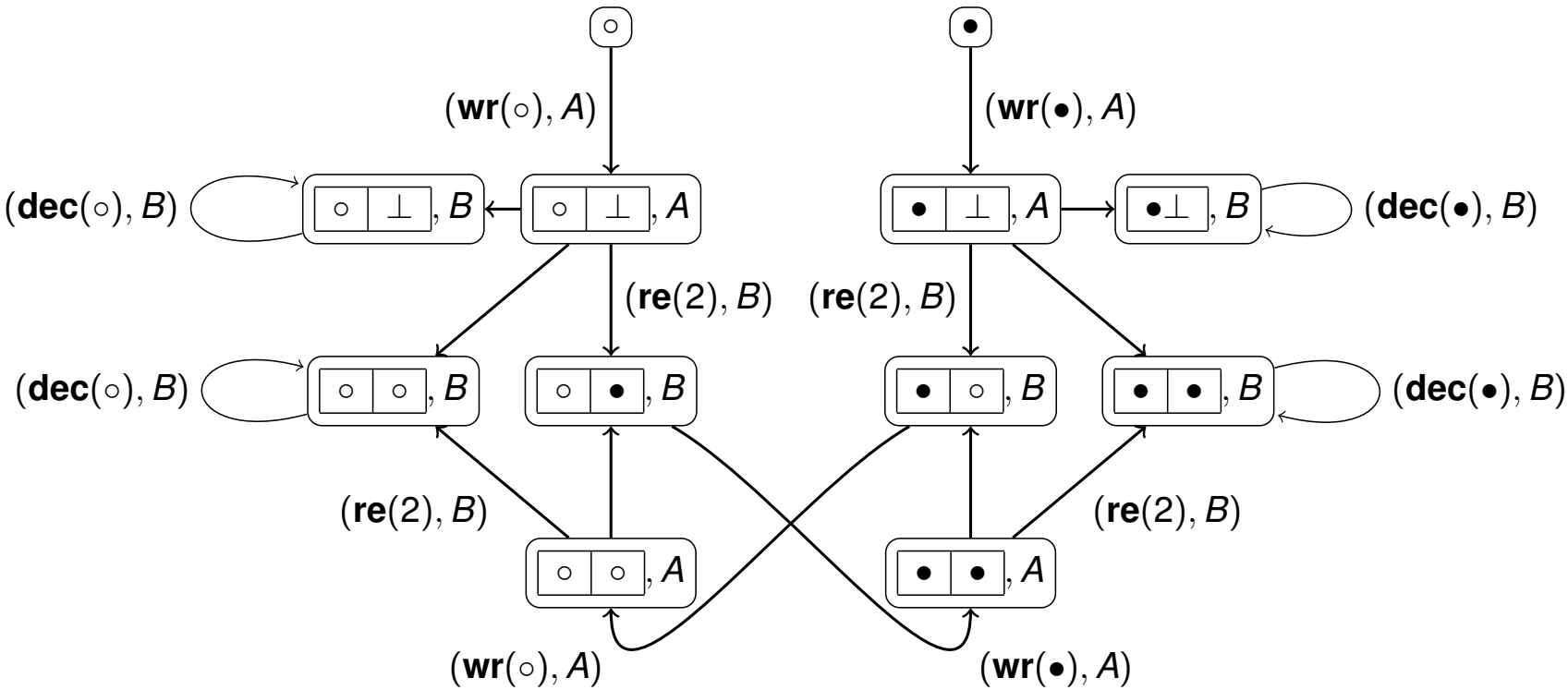
Consensus Algorithm for process *i* in $\{1, 2\}$

Input : V: the input value of process i

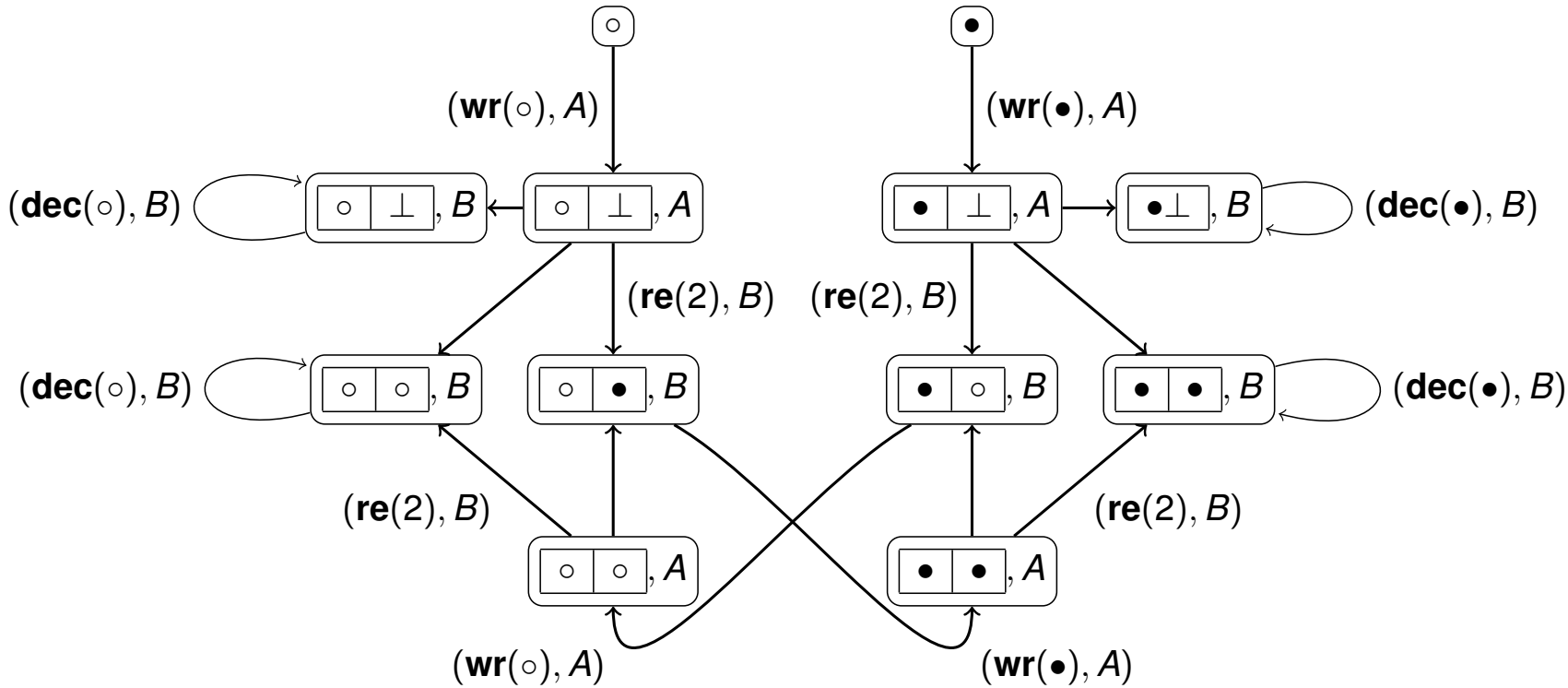
```
 1: while true do
 2:     r[i]:=V
 3:     tmp:=r[3-i]
 4:     if tmp=V or tmp = ⊥ then
 5:         Decide(V)
 6:         Exit()
 7:     else
 8:         V:=tmp
 9:     end if
10: end while
```

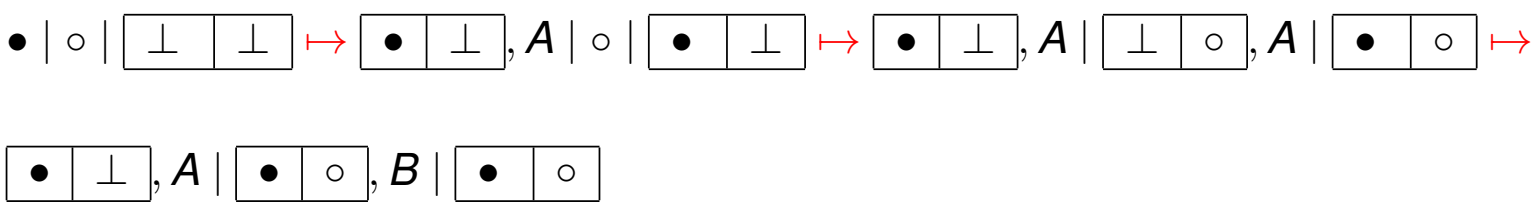
Example in our formalism

Process algorithm for process with id 1



Example of execution





Outline

LTL and Kripke structures

- Linear time Temporal Logic used to specify executions
- It is interpreted over Kripke structures, i.e. graph where each node is labelled with atomic propositions
- If an atomic proposition is present in a node, it is true in this node

Syntax

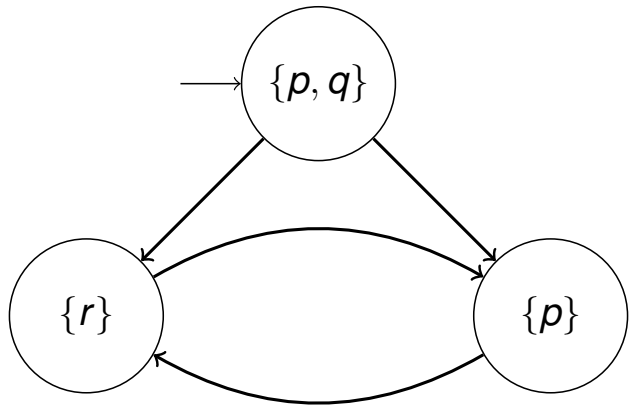
$\phi, \psi ::= p \mid \neg\phi \mid \phi \vee \psi \mid \mathbf{X}\phi \mid \phi \mathbf{U} \psi$

- Interpreted over infinite paths of the structure
- $\mathbf{X}\phi \rightarrow$ the next state verifies ϕ
- $\phi \mathbf{U} \psi \rightarrow$ eventually ψ holds and in the meantime ϕ is true

Classical shortcuts:

- $\mathbf{F}\phi \stackrel{def}{=} \top \mathbf{U} \psi \rightarrow$ eventually ψ holds
- $\mathbf{G}\phi \stackrel{def}{=} \neg\mathbf{F}\neg\phi \rightarrow \phi$ always holds
- $\mathbf{GF}\phi \rightarrow \phi$ holds infinitely often

LTL and Kripke structures



- A structure satisfies a formula ϕ iff all its paths satisfy ϕ
- Formulae satisfied by the system: $\mathbf{GF}p$, $\mathbf{F}r$
- Formulae not satisfied by the system: $\mathbf{X}r$, $\mathbf{G}p$

Specifying distributed algorithms

- The execution graph of the distributed algorithm A is our Kripke structure $\mathcal{K}_A$ with:
  - A initial state which non deterministic goes to a combination of possible initial value
  - All the possible executions are considered
- Add the following atomic propositions:
  - $\text{active}_i \rightarrow$ process i is the last one to perform an action
  - $\text{D}_i \rightarrow$ process i has decided
  - $\text{In}_i^d \rightarrow$ the initial value of process i is d
  - $\text{Out}_i^d \rightarrow$ the output value of process i is d.
- We have the following assumptions:
  - In_i^d is only true in the second states (where the initial values are chosen)
  - $\text{D}_i \Leftrightarrow \bigvee_d \text{Out}_i^d$

Classical properties

Reminder on consensus

- Each process is equipped with an initial value
- **Agreement:** The processes that decide must decide the same value
- **Validity:** The decided value must be one of the initial ones

In our formalism

- **Agreement:**

$$\Phi^{c}_{\text{agree}} \stackrel{\text{def}}{=} \mathbf{G} \bigwedge_{1 \leq i \neq j \leq n} \Big((\mathsf{D}_i \wedge \mathsf{D}_j) \Rightarrow (\bigwedge_{d} \mathsf{Out}_i^d \Leftrightarrow \mathsf{Out}_j^d) \Big)$$

- **Validity:**

$$\Phi^{c}_{\text{valid}} \stackrel{\text{def}}{=} \mathbf{X} \bigwedge_{1 \leq i \leq n} \bigwedge_{d} \Big(\big(\mathbf{F}\, \mathsf{Out}_i^d \big) \Rightarrow \big(\bigvee_{1 \leq j \leq n} \mathsf{In}_j^d \big) \Big)$$

Specifying execution contexts

What are execution contexts

- They determine the authorized sequences of active processes
- An algorithm might not be correct for all the possible sequences but for a subset of them
- For all the authorized sequences of active processes, any process infinitely often active has to decide

Examples

- **Wait-free:** each process produces an output value after a finite number of its own steps

$$\Phi_{\text{wf}} \stackrel{\text{def}}{=} \bigwedge_{1 \leq i \leq n} \Big((\mathbf{G}\,\mathbf{F}\,\text{active}_i) \Rightarrow (\mathbf{F}\,\text{D}_i) \Big)$$

- **Obstruction-free:** every process that eventually executes in isolation has to produce an output value

$$\Phi_{\text{of}} \stackrel{\text{def}}{=} \bigwedge_{1 \leq i \leq n} ((\mathbf{F}\,\mathbf{G}\,\text{active}_i) \Rightarrow (\mathbf{F}\,\text{D}_i))$$

Outline

Verification vs Synthesis

Verification

- Given an algorithm A and a specification ϕ, check if $\mathcal{K}_A$ satisfies ϕ
- Can be achieved in PSPACE
- For instance, we can verify that our example algorithm satisfies $\Phi^c_{\text{agree}} \wedge \Phi^c_{\text{valid}} \wedge \Phi_{\text{of}}$, it is a correct obstruction free consensus algorithm

Synthesis

- Only the specification is given
- The algorithm is not given but built **automatically**
- The algorithm is correct **by construction**
- Much harder problem

The synthesis problem

The synthesis problem

- **Inputs:** A number n of processes, a data set $\mathcal{D}$, a set of memory values M and a LTL formula Φ
- **Output:** Is there a n processes distributed algorithm over $\mathcal{D}$ which uses memory M and satisfies ϕ ?

Remarks:

- If the answer is **NO**, we cannot conclude that there is no algorithm in general, but that for the given bounds there is no algorithm
- If the answer is **YES**, we would like to have the algorithm, i.e. the method is constructive

Solving the synthesis problem

The synthesis problem is decidable and in the positive cases, our method produces an algorithm

Main ingredients:

1. Build an 'universal' Kripke structure $\mathcal{K}_U$ which from any configuration allows any possible action
2. Add specific atomic propositions to extract an algorithm from $\mathcal{K}_U$
   - Consistent labelling to state the actions to be performed by a process
   - Atomic proposition $P^i_{(a,m)} \rightarrow$ the next action of process i is a and its state changes to m
   - Ensures that with the same local state and the same view, the process performs the same action
3. Extract with an extra LTL formula Φ_{out} the paths corresponding to the algorithm
4. Check whether $\mathcal{K}_U$ satisfies $\Phi_{out} \Rightarrow \Phi$

Outline

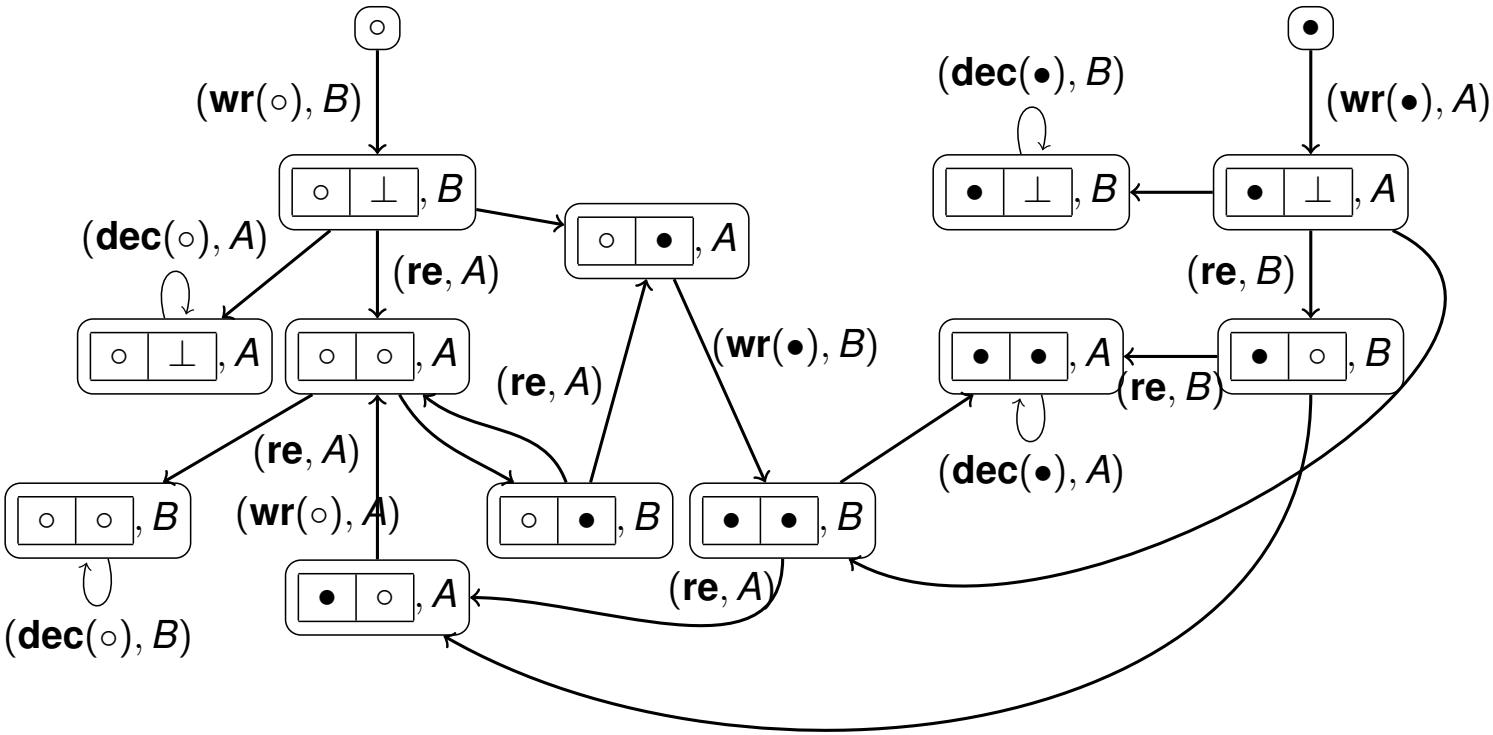
Methodology

- The goal was to check whether some SMT solver could handle our method
- Our prototype does not handle general LTL formula but an ad hoc encoding for some specific executive contexts (wait-free, obstruction free, round-robin like)
- It produces an existentially quantified formula and gives it to the SMT solver Z3
- If the SMT solver returns SAT, we can extract the algorithm
- We furthermore check with a classical model-checker that the produced algorithm is correct
- Inputs of our prototype : number of processes, data range, size of memory, type of execution context, value of ϵ for ϵ agreement

Results

- We study two distributed problems: **consensus** and **ϵ-agreement**
- We effectively face the state explosion problem and restrict ourselves to two processes
- We had to help the solver with some heuristics to make it converge
- Our implementation is really naive (and could be improve a lot)
- Found automatically algorithms in this case:
  - **Consensus**: Obstruction free but as well more unusual contexts like obstruction free+round-robin
  - **ϵ-agreement**: Wait-free and different values of ϵ

Example



Algorithm for consensus supporting obstruction free and round-robin contexts

Conclusion

What have we done ?

- Propose a general framework to model algorithms and to specify executive contexts
- Test how hard is in practice the synthesis problem when bounding everything (size of algorithms and exchanged data)
- An important amount of implementation work is necessary to make such a synthesis work for more than two processes

What's next ?

- Big Challenge : Find decision procedure in the general case
- In general checking whether a distributed problem can be solved by a wait-free algorithm is an undecidable problem
- Find distributed problems and executive contexts for which the synthesis problem can be solved
- Find way to deal with dynamic changes of executive contexts