Formal Methods for the Verification of Distributed Algorithms

Paul Gastin

Laboratoire Spécification et Vérification
ENS Paris Saclay & CNRS

with C. Aiswarya (CMI) & Benedikt Bollig (LSV)
Motivations

- Distributed algorithms are extremely difficult to get right
- Correctness proofs are often involved
- Formal methods may help verifying the correctness of tricky algorithms
Peterson's algorithm

\[
\text{for } n \text{ from } 0 \text{ to } N-1 \text{ exclusive } \\
\text{  \hspace{1cm} level}[i] \leftarrow n \\
\text{  \hspace{1cm} last_to_enter}[n] \leftarrow i \\
\text{  \hspace{1cm} while } \text{last_to_enter}[n] = i \text{ and there exists } k \neq i, \text{ such that } \text{level}[k] \geq n \\
\text{  \hspace{1cm} wait}
\]

Specification

Mutual exclusion

\[
\bigwedge_{i \neq j} \neg(\text{CS}_i \land \text{CS}_j)
\]
Peterson's algorithm

for n from 0 to N−1 exclusive
    level[i] ← n
    last_to_enter[n] ← i
while last_to_enter[n] = i and there exists k ≠ i, such that level[k] ≥ n
    wait

Specification
Mutual exclusion
\( \bigwedge_{i \neq j} \neg (CS_i \land CS_j) \)
Models for programs/algorithms

- Finite state machine (control points)
- Data structures
  - Boolean variables
  - Integer variables
- Stacks (recursivity)
- Queues (asynchronous communication)
Peterson's algorithm

for n from 0 to N−1 exclusive
level[i] := n
last_to_enter[n] := i

while last_to_enter[n] = i and there exists k ≠ i, such that level[k] ≥ n
wait

level[i] := n
last_to_enter[n] := i

else
n := n+1

n < N
wait

n = N
wait

CS

init

n := 0
trying

last_to_enter[n] = i
max{level[k], k≠i} ≥ n
Franklin’s leader election algorithm
Processes are arranged in an undirected ring.
Each node has a unique identity.
Each node is either active or passive (relay mode) at a given time.

The algorithm executes as follows:
- Each active node sends its identity to its neighbors.
  Let each active node \( p_1 \) receive identities from \( p_0 \) and \( p_2 \). Where \( p_0 \) and \( p_2 \) are its either neighbors in the ring.
- If \( \min(\text{ID}[p_0], \text{ID}[p_2]) > \text{ID}[p_1] \), then \( p_1 \) becomes passive
- If \( \min(\text{ID}[p_0], \text{ID}[p_2]) < \text{ID}[p_1] \), then \( p_1 \) sends its ID to its neighbors again
- If \( \min(\text{ID}[p_0], \text{ID}[p_2]) == \text{ID}[p_1] \), then \( p_1 \) declares itself as leader
- Passive nodes only pass on messages.
- The loop continues until a leader with highest unique ID has been elected.
Languages for the specification

- Modal logics
- Temporal logics
- First-order logic
- Dynamic logics
Model checking: sources of undecidability

- Each infinite/unbounded aspects
  - number of processes/agents
  - Integer variables (pids, timestamps, …)
  - FIFO channels (asynchronous communication)
Model checking (Linear time)

System model

$\mathcal{A}$

$L(\mathcal{A})$

Specification

$\varphi$

$L(\varphi)$

model checking

$L(\mathcal{A}) \subseteq L(\varphi)$?

set of possible traces

set of admissible traces

$\neg F$

$L(\varphi)$

$L(\mathcal{A})$
Model checking: First solution

Reachability

\( L(A) \cap L(A') = \emptyset \)

Behavior

\( L(\varphi) \)

finite automata

model checking

\( L(A) \subseteq L(\varphi) \)?

LTL specification

\( \neg F \)
Model checking: Second solution

Finite automata

LTL specification

Validity

Behavior

$\mathcal{A}$

$\mathcal{L}(\mathcal{A})$

$\mathcal{L}(\varphi)$

$\psi \Rightarrow \varphi$

$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\varphi)$?

Effective

Model checking

$\neg F$
Models of Distributed Systems
Distributed algorithms: our hypotheses

- Number of processes: arbitrary, unknown
- Unique process identification
  - Comparisons: <, =
  - No arithmetic
- Topology: fixed degree (ring, …)
- Communication: Synchronous in rounds
  - Round: send messages, receive messages, compute and update local registers
Distributed algorithms

Leader election [Franklin ’82]

Distributed algorithm

Behavior

active

passive

left ! id right ! id
left?r₁ right?r₂
id > r₁ ∧ id > r₂

left ! id right ! id
left?r₁ right?r₂
id < r₁ ∨ id < r₂

left?r₁ right?r₂
id > r₁ ∧ id > r₂

leader election
Distributed algorithms

Leader election [Franklin ’82]

Distributed algorithm

Behavior

- Active
  - id = 47
  - r1 = 23
  - r2 = 19

- Passive
  - id = 23
  - r1 = 19

- Left active
  - id > r
  - id > r

- Right active
  - id < r
  - id < r

- Left passive
  - id > r
  - id > r

- Right passive
  - id < r
  - id < r

Round

Node IDs:
- 5
- 7
- 19
- 42
- 47
- 23
Distributed algorithms

Leader election [Franklin '82]

Distributed algorithm

Behavior

left ! id  right ! id
left ? r₁  right ? r₂
id > r₁ ∧ id > r₂

active

fwd

passive

left ! id  right ! id
left ? r₁  right ? r₂
id < r₁ ∨ id < r₂
Distributed algorithms

Leader election [Franklin ’82]

Distributed algorithm

active

left ! id right ! id
left?r₁ right?r₂
id > r₁ ∧ id > r₂

left ? id right ? id
id = r₁

left ! id right ! id
left?r₁ right?r₂
id < r₁ ∨ id < r₂

leader

fwd

passive

Behavior
Distributed algorithms

Leader election [Franklin ’82]

Distributed algorithm

Behavior

active

left ! id  right ! id
left?r₁  right?r₂
id > r₁ ∧ id > r₂

left ! id  right ! id
left?r₁  right?r₂
id = r₁

left ! id  right ! id
left?r₁  right?r₂
id < r₁ ∨ id < r₂

leader

fwd

passive
Distributed algorithms

- Identical finite-state processes
- Number of processes is unknown and unbounded
- Processes have unique pids (integers — unbounded data)
A formal model for distributed algorithms
An automata-like way of writing DA

Every process can be described by:

- Set of states
- Initial state
- Set of registers
  - stores pid

- Set of transitions
  - send pids to neighbours
  - receive pids from neighbours, and store in registers
  - compare registers
  - update registers
Behaviors

Distributed algorithm

Cylinders
Arbitrary length and width
Labelled with data from an infinite domain

3 sources of infinity
Abstraction of Data Values
Behaviors: Cylinders of arbitrary width and length

Data from an infinite domain

System: Register automata with data comparisons

Specification: Data PDL with data comparisons

UNDECIDABLE
Reduction to Satisfiability of LCPDL: Data abstraction

\[ A \rightarrow \beta \]
valid over cylinders

\[ \iff A \models \varphi \]

Distributed algorithm

PDL with loop (over finite alphabet)

Data PDL
LCPDL: Propositional Dynamic logic with Loop and Converse

\[ \Psi, \Psi' ::= E\psi \mid \neg \Psi \mid \Psi \land \Psi' \]

\[ \psi, \psi' ::= \dagger \mid p \mid \neg \psi \mid \psi \land \psi' \mid \langle \pi \rangle \psi \mid \text{loop}(\pi) \]

\[ \pi, \pi' ::= \{\psi\}? \mid \rightarrow \mid \downarrow \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \mid \pi^{-1} \]
LCPDL: Propositional Dynamic logic with Loop and Converse

\[ \Psi, \Psi' ::= E \psi \mid \neg \Psi \mid \Psi \land \Psi' \]
\[ \psi, \psi' ::= \bot \mid p \mid \neg \psi \mid \psi \land \psi' \mid \langle \pi \rangle \psi \mid \text{loop}(\pi) \]
\[ \pi, \pi' ::= \{\psi\}? \mid \rightarrow \mid \downarrow \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \mid \pi^{-1} \]
LCPDL: Propositional Dynamic logic with Loop and Converse

\[
\begin{align*}
\Psi, \Psi' & ::= \mathsf{E}\, \psi \mid \neg \Psi \mid \Psi \land \Psi' \\
\psi, \psi' & ::= \mathsf{†} \mid p \mid \neg \psi \mid \psi \land \psi' \mid \langle \pi \rangle \psi \mid \mathsf{loop}(\pi) \\
\pi, \pi' & ::= \{ \psi \}\? \mid \rightarrow \mid \downarrow \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \mid \pi^{-1}
\end{align*}
\]
LCPDL: Propositional Dynamic logic with Loop and Converse

\[ \psi, \psi' ::= E \psi \mid \neg \psi \mid \psi \land \psi' \]

\[ \psi, \psi' ::= \uplus \mid p \mid \neg \psi \mid \psi \land \psi' \mid \langle \pi \rangle \psi \mid \text{loop}(\pi) \]

\[ \pi, \pi' ::= \{ \psi \}? \mid \rightarrow \mid \downarrow \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \mid \pi^{-1} \]
LCPDL: Propositional Dynamic logic with Loop and Converse

\[
\begin{align*}
\Psi, \Psi' &::= E \psi \mid \neg \Psi \mid \Psi \land \Psi' \\
\psi, \psi' &::= \frac{\downarrow}{\downarrow} \mid p \mid \neg \psi \mid \psi \land \psi' \mid \langle \pi \rangle \psi \mid \text{loop}(\pi) \\
\pi, \pi' &::= \{ \psi \}? \mid \rightarrow \mid \downarrow \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \mid \pi^{-1}
\end{align*}
\]
Data abstraction: symbolic runs + tracking data

Distributed algorithm

Active
id = 47
r1 = 23
r2 = 19
Data abstraction: symbolic runs + tracking data

Distributed algorithm
A pid distribution realizes a symbolic run if all guards are satisfied.

Pb: Is there a pid distribution realizing a symbolic run?
Data abstraction: symbolic runs + tracking data

- Register updates

Distributed algorithm
Data abstraction: symbolic runs + tracking data

- Register updates

Distributed algorithm
Data abstraction: symbolic runs + tracking data

- Register updates

Distributed algorithm
Data abstraction: symbolic runs + tracking data

- Register updates

Distributed algorithm
Data abstraction: symbolic runs + tracking data

- Register updates
Data abstraction: symbolic runs + tracking data
Data abstraction: symbolic runs + tracking data

- Register updates

Distributed algorithm
Data abstraction: symbolic runs + tracking data

- Register updates
Data abstraction: symbolic runs + tracking data

- Register updates

\[(r_1, id)\text{-path}\]

can be expressed in CPDL

PDL with converse
Data abstraction: symbolic runs + tracking data

- Register updates
- Register equality check
Data abstraction: symbolic runs + tracking data

- Register updates
- Register equality check

\[ \pi_1 := (r_1, id) \text{-path} \]
\[ \pi_2 := (r_2, id) \text{-path} \]
Data abstraction: symbolic runs + tracking data

- Register updates
- Register equality check

\( \pi_1 : (r_1, \text{id})\)-path
\( \pi_2 : (r_2, \text{id})\)-path

\( r_2 = r_1 \) iff loop( \( \pi_1 \); \( \pi_2^{-1} \) )

can be expressed in LCPDL CPDL with loop
Data abstraction: symbolic runs + tracking data

- Register updates
- Register equality check
- Register comparison

Distributed algorithm
Data abstraction: symbolic runs + tracking data

- Register updates
- Register equality check
- Register comparison
Data abstraction: symbolic runs + tracking data

- Register updates
- Register equality check
- Register comparison
Data abstraction: symbolic runs + tracking data

- Register updates
- Register equality check
- Register comparison
Data abstraction: symbolic runs + tracking data

- If there is a $<$-loop, no pid assignments can turn the symbolic cylinder into a valid run.
- If no such loops, then there are pids that allow a valid realization of the symbolic cylinder.
Data abstraction: symbolic runs + tracking data

No loop of the form
$(\Sigma_{i,j} (r_i, id)\text{-path}^{-1}; r_i < r_j; (r_j, id)\text{-path})^+$

- If there is a $<$-loop, no pid assignments can turn the symbolic cylinder into a valid run.
- If no such loops, then there are pids that allow a valid realization of the symbolic cylinder.
Data abstraction: symbolic runs + tracking data

No loop of the form 
\[(\Sigma_{i,j} (r_i, id)\text{-path}^{-1}; r_i < r_j; (r_j, id)\text{-path})^{+}\]

- If there is a \(<\)-loop, no pids can turn the symbolic cylinder into a valid run.
- If no such loops, then there are pids that allow a valid realization of the symbolic cylinder.
Data abstraction: symbolic runs + tracking data

Distributed algorithm

PDL with loop (over finite alphabet)

Data PDL
Distributed algorithms: typical properties

- Leader election:
  - At the end there is a unique leader
  - All other processes are passive
  - The leader has the maximal pid
- Distributed sorting algorithm
  - The output values form a permutation of the input values
  - If q is on the right of p, and q ≠ leader then p.v < q.v
Specifications

Data PDL

\[
\langle \pi \rangle r \neq \langle \pi' \rangle r'
\]

\[
\Phi, \Phi' ::= A \phi \mid \Phi \land \Phi'
\]

\[
\phi, \phi' ::= \varphi \mid \phi \land \phi' \mid \varphi \lor \phi \mid [\pi] \phi \mid \langle \eta \rangle r < \langle \eta' \rangle r' \mid \langle \eta \rangle r \leq \langle \eta' \rangle r'
\]

\[
\varphi, \varphi' ::= \dagger \mid p \mid \neg \varphi \mid \varphi \land \varphi' \mid \langle \pi \rangle \varphi \mid \langle \pi \rangle r = \langle \pi' \rangle r' \mid \langle \pi \rangle r \neq \langle \pi' \rangle r'
\]

\[
\pi, \pi' ::= \{ \varphi \}? \mid \rightarrow \mid \downarrow \mid \pi^{-1} \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^*\]

\[
\eta, \eta' ::= \{ \varphi \}? \mid \leftarrow \mid \rightarrow \mid \downarrow \mid \uparrow \mid \eta \cdot \eta' \mid F^\eta
\]

Inspired by [Bojanczyk et al. ’09; Figueira-Segoufin ‘11]
Specifications
Data PDL

\[ \langle \pi \rangle r \neq \langle \pi' \rangle r' \]

Moves inside the behavior

compare values at different nodes

\[ \Phi, \Phi' ::= A\phi \mid \Phi \land \Phi' \]
\[ \phi, \phi' ::= \varphi \mid \phi \land \phi' \mid \varphi \lor \varphi \mid [\pi]\phi \mid \langle \eta \rangle r < \langle \eta' \rangle r' \mid \langle \eta \rangle r \leq \langle \eta' \rangle r' \]
\[ \varphi, \varphi' ::= \top \mid p \mid \neg \varphi \mid \varphi \land \varphi' \mid \langle \pi \rangle \varphi \mid \langle \pi \rangle r = \langle \pi' \rangle r' \mid \langle \pi \rangle r \neq \langle \pi' \rangle r' \]
\[ \pi, \pi' ::= \{\varphi\}? \mid \rightarrow \mid \downarrow \mid \pi^{-1} \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \]
\[ \eta, \eta' ::= \{\varphi\}? \mid \leftarrow \mid \rightarrow \mid \downarrow \mid \uparrow \mid \eta \cdot \eta' \mid F^\eta \]

Inspired by [Bojanczyk et al. ’09; Figueira-Segoufin ‘11]
Leader election [Franklin ’82]

At the end, there is a leader, and the leader is the process with the maximum id.

Distributed algorithm

Behavior

move in the cylinder
Distributed algorithms

At the end, there is a leader, and the leader is the process with the maximum id.

Leader election [Franklin ’82]

move in the cylinder

Distributed algorithm

active

left ! id  right ! id
left?r₁  right?r₂
id > r₁ ∧ id > r₂

left ! id  right ! id
left?r₁  right?r₂
id = r₁

left ! id  right ! id
left?r₁  right?r₂
id < r₁ ∨ id < r₂

leader

fwd

passive

Behavior

Specification
Distributed algorithms

Leader election [Franklin ’82]

«At the end, there is a leader, and the leader is the process with the maximum id.»

move in the cylinder

compare values at different nodes

Distributed algorithm

Behavior

Specification
Distributed algorithms

Behavior

At the end, there is a leader, and the leader is the process with the maximum id.

Data Propositional Dynamic Logic

[Bojanczyk et al. '09; Figueira-Segoufin '11]

move in the cylinder

compare values at different nodes
Distributed algorithms

Behavior

Distributed algorithm

Leader election [Franklin '82]

move in the cylinder

compare values at different nodes

«At the end, there is a leader, and the leader is the process with the maximum id.»

\[
\overrightarrow{*} \ (\ \overleftarrow{} \ \land \ \langle \text{go-to-} \rangle \ \land \ [\downarrow*] (\text{id} \leq \langle \text{go-to-} \rangle \text{id}))
\]

go-to- = (\neg \ \downarrow)* \ \square
Distributed algorithms

At the end, there is a leader, and the leader is the process with the maximum id.

\[ \langle \rightarrow \rangle^* \left( \neg \langle \rightarrow \rangle \land \langle \text{go-to-} \rangle \right) \land [\downarrow^*] \left( \text{id} \leq \langle \text{go-to-} \rangle \text{id} \right) \]

go-to- = (\neg \downarrow)^*
Distributed algorithms

At the end, there is a leader, and the leader is the process with the maximum id.

For all $n$, pid distributions, and accepting runs:

\[
\langle \rightarrow \rangle^* (\neg \langle \rightarrow \rangle \land \langle \text{go-to-} \rangle \land [\downarrow^*] (\text{id} \leq \langle \text{go-to-} \rangle \text{id}))
\]

\[
\text{go-to-} = (\neg \downarrow)^* \text{id}
\]
Specifications: distributed sorting

The output values form a permutation of the input values

- same set of values:
  \[ [\rightarrow^*](\langle\varepsilon\rangle r) = \langle\uparrow^* \{\neg\langle\uparrow\rangle\} ? \rightarrow^* \rangle r \]

- pairwise distinct:
  \[ \neg\langle\rightarrow^*\rangle(\langle\varepsilon\rangle r) = \langle(\rightarrow\{\neg\frac{1}{2}\} ? \rangle^+ \rangle r \]

\[ \Phi, \Phi' ::= A \phi \mid \Phi \land \Phi' \]
\[ \phi, \phi' ::= \varphi \mid \phi \land \phi' \mid \varphi \lor \phi \mid [\pi]\phi \mid \langle\eta\rangle r < \langle\eta'\rangle r' \mid \langle\eta\rangle r \leq \langle\eta'\rangle r' \]
\[ \varphi, \varphi' ::= \frac{1}{2} \mid p \mid \neg\varphi \mid \varphi \land \varphi' \mid \langle\pi\rangle\varphi \mid \langle\pi\rangle r = \langle\pi'\rangle r' \mid \langle\pi\rangle r \neq \langle\pi'\rangle r' \]
\[ \pi, \pi' ::= \{\varphi\}? \mid \rightarrow \mid \downarrow \mid \pi^{-1} \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \]
\[ \eta, \eta' ::= \{\varphi\}? \mid \leftarrow \mid \rightarrow \mid \downarrow \mid \uparrow \mid \eta \cdot \eta' \mid F^\eta \]
Distributed algorithms

**Distributed algorithm**

- **active**
  - left ! id, right ! id
  - left?r_1, right?r_2
    - id > r_1 ∧ id > r_2
  - left?r_1, right?r_2
    - id = r_1
  - left ! id, right ! id
    - left?r_1, right?r_2
      - id < r_1 ∨ id < r_2
  - fwd
    - passive

- **t_1**
  - left ! id, right ! id
  - left?r_1, right?r_2
    - id > r_1 ∧ id > r_2
  - left?r_1, right?r_2
    - id = r_1
  - left ! id, right ! id
    - left?r_1, right?r_2
      - id < r_1 ∨ id < r_2
  - fwd
    - passive

- **t_2**
  - left ! id, right ! id
  - left?r_1, right?r_2
    - id = r_1
  - left ! id, right ! id
    - left?r_1, right?r_2
      - id < r_1 ∨ id < r_2
  - fwd
    - passive

- **t_3**
  - left ! id, right ! id
  - left?r_1, right?r_2
    - id < r_1 ∨ id < r_2
  - fwd
    - passive

- **t_4**
  - left ! id, right ! id
  - left?r_1, right?r_2
    - id > r_1 ∧ id > r_2
  - left?r_1, right?r_2
    - id = r_1
  - left ! id, right ! id
    - left?r_1, right?r_2
      - id < r_1 ∨ id < r_2
  - fwd
    - passive

**Behavior**

- id
- left ! id, right ? id
- left?r_1, right?r_2
  - id > r_1 ∧ id > r_2
  - id = r_1
  - id < r_1 ∨ id < r_2

- id
- t_1
  - left!id
  - t_2
  - left!id
  - t_3
  - left!id
  - t_4
  - left!id

**Data PDL**

«There is a leader, and the leader is the process with the maximum id.»

For all n, pid distributions, accepting runs, and processes:

\[
\langle \leftarrow \rangle^* \left( \neg \langle \rightarrow \rangle \land \langle \text{go-to-} \rangle \right) \land \left[ \downarrow \right)^* (\text{id} \leq \langle \text{go-to-} \rangle \text{id})
\]

goto- = (\neg \downarrow)^*
Distributed algorithms

Distributed algorithm

Behavior

«There is a leader, and the leader is the process with the maximum id.»

For all $n$, pid distributions, accepting runs, and processes:

$$\langle \leftarrow \rangle^* \left( \neg \langle \rightarrow \rangle \land \langle \text{go-to-} \rangle \right) \land [\Downarrow^*] \left( \text{id} \leq \langle \text{go-to-} \rangle \text{id} \right)$$

$$\text{go-to-} = (\neg \Downarrow)^*$$
Distributed algorithms

Behavior

«There is a leader, and the leader is the process with the maximum id.»

For all n, pid distributions, accepting runs, and processes:

\[
\langle \leftarrow \ast \rangle ( \neg \langle \leftarrow \rangle \land \langle \text{go-to-} \rangle \\
\land [\downarrow \ast] (\text{id} \leq \langle \text{go-to-} \rangle \text{id}) )
\]

\[
\text{go-to-} = (\neg \downarrow \ast)
\]
Distributed algorithms

Behavior

Distributed algorithm

Data PDL

«There is a leader, and the leader is the process with the maximum id.»

For all $n$, pid distributions, accepting runs, and processes:

$$\langle \leftarrow \rangle^* \left( \neg \langle \rightarrow \rangle \land \langle \text{go-to} \rangle \right) \land \left[ \downarrow^* \right] \left( \text{id} \leq \langle \text{go-to} \rangle \text{id} \right)$$

$$\text{go-to} = (\neg \downarrow)^*$$
Distributed algorithms

«There is a leader, and the leader is the process with the maximum id.»

For all $n$, pid distributions, accepting runs, and processes:

\[
\langle \leftarrow \star \rangle ( \neg \langle \rightarrow \rangle \land \langle \text{go-to-} \rangle \land [\downarrow \star] (\text{id} \leq \langle \text{go-to-} \rangle \text{id}))
\]

\[
\text{go-to-} = (\neg \downarrow \star)^* \quad \varphi
\]
Distributed algorithms

Behavior

«There is a leader, and the leader is the process with the maximum id.»

For all $n$, pid distributions, accepting runs, and processes:

\[
\langle \rightarrow^* \rangle (\neg \langle \rightarrow \rangle \land \langle \text{go-to-} \rangle \land \downarrow^* (\text{id} \leq \langle \text{go-to-} \rangle \text{id}))
\]

\[
\text{go-to-} = (\neg \downarrow)^* \downarrow
\]
Behavior

There is a leader, and the leader is the process with the maximum id.

For all $n$, pid distributions, accepting runs, and processes:

\[
\langle \pi \rangle r \leq \langle \pi' \rangle r'
\]

Loop $(\pi . (r, r') - \langle \text{path} \rangle . (\pi')^{-1})$

\[
\langle \rightarrow^* \rangle (\neg \langle \rightarrow \rangle \land \langle \text{go-to-} \rangle \land [\downarrow^*] (id \leq \langle \text{go-to-} \rangle \cdot id))
\]

Data PDL

Distributed algorithm

Distributed algorithms
Distributed algorithms

There is a leader, and the leader is the process with the maximum id.

For all n, pid distributions, accepting runs, and processes:
\[
\langle \pi \rangle r \leq \langle \pi' \rangle r'
\]

Loop (\(\pi \cdot (r, r') \leftarrow \text{path} \cdot (\pi')^{-1}\))

```
\langle \rightarrow^* \rangle (\neg \langle \leftarrow \rangle \land \langle \text{go-to-} \rangle)
\land [\downarrow^*] (id \leq \langle \text{go-to-} \rangle id)
\varphi
```

go-to-

\(= (\neg \text{left} \leftarrow \downarrow)^* \text{right} \text{right} \)
Distributed algorithms

no loop
⇒
no evidence of \( \varphi \)
⇒
there are pids making \( \varphi \) false

\[ \langle \pi \rangle r \leq \langle \pi' \rangle r' \]
Loop ( \( \pi \cdot (r,r') \langle \text{-path} \rangle \cdot (\pi')^{-1} \) )

\[ \langle \text{go-to-} \rangle \]

«There is a leader, and the leader is the process with the maximum id.»

For all \( n \), pid distributions, accepting runs, and processes:
\[
\langle \rightarrow^* \rangle ( \neg \langle \rightarrow \rangle \land \langle \text{go-to-} \rangle \\
\land [\downarrow^*] (id \leq \langle \text{go-to-} \rangle \cdot id) )
\]

\[ \text{go-to-} = (\neg \downarrow \downarrow)^* \]

Behavior

Data PDL
Distributed algorithms

no loop
⇒
no evidence of $\varphi$
⇒
there are pids making $\varphi$ false

$\text{id} \leq \langle \downarrow \rangle \text{id}$
\[ \lor \]
$\text{id} > \langle \downarrow \rangle \text{id}$

 DISTRIBUTED ALGORITHM

there is loop
\[ \iff \]
$\varphi$ holds here

$\langle \pi \rangle \mathcal{r} \leq \langle \pi' \rangle \mathcal{r}'$

Loop ($\pi . (r,r') \leftarrow \text{-path} . (\pi')^{-1}$)

«There is a leader, and the leader is the process with the maximum id.»

For all $n$, pid distributions, accepting runs, and processes:

$\langle \rightarrow^* \rangle$ ( $\neg \langle \rightarrow \rangle$ \land \langle \text{go-to-} \rangle$

$[\downarrow^*]$ ( $\text{id} \leq \langle \text{go-to-} \rangle \text{id}$ )

$\varphi$

go-to- = (\neg \text{id} \downarrow)^* \text{id}$

Data PDL

Behavior
Specifications Data PDL

\[ \langle \eta \rangle r < \langle \eta' \rangle r' \]

\[ \eta \quad \eta' \]

\[ r \quad < \quad r' \]

deterministic paths
Data abstraction

Distributed algorithm

PDL with loop (over finite alphabet)

Data PDL

\[ \alpha \Rightarrow \beta \]

valid over cylinders

\[ \mathcal{A} \models \varphi \]
Data abstraction

Distributed algorithm

PDL with loop (over finite alphabet)

Data PDL

UNDECIDABLE

two unbounded dimensions
Under approximate verification

Distributed algorithm

PDL with loop (over finite alphabet)

Data PDL

\[ A = \text{valid over cylinders} \]

undecidable

\[ \Leftrightarrow A \models \varphi \]

restrict to bounded number of rounds
Distributed algorithm

PDL with loop (over finite alphabet)

Data PDL

exponentially smaller than # of processes

\[ A \models \varphi \]

valid over cylinders

\[ \alpha \Rightarrow \beta \]

undecidable

restrict to bounded number of rounds
PDL with loop over bounded cylinders

\[\Rightarrow\]

PDL with loop over words
PDL with loop over bounded cylinders

\[\Rightarrow\]

PDL with loop over words
PDL with loop over bounded cylinders

PDL with loop over words
PDL with loop over bounded cylinders

⇒

PDL with loop over words
PDL with loop over bounded cylinders

↔

PDL with loop over words

↔

Alternating 2-way Automata

↔

PSPACE

[Göller-Lohrey-Lutz '08] [Serre '08]
Summary & Conclusion
Theorem (Aiswarya-Bollig-Gastin; CONCUR ’15).
Round-bounded model checking distributed algorithms* against Data PDL is PSPACE-complete**.

* with registers, register guards, and register updates (no arithmetic)
** unary encoding of # of rounds

Summary

exponentially smaller than # of processes
Conclusion

‣ What is the right temporal logic? Use generic Data PDL.
‣ How to deal with data? Use symbolic technique.
‣ How to deal with undecidability? Under-approximation.

Future work …

• Other operations? (increment only, decrement only, …)
• Other topologies?
• Other restrictions? (bounded tree-width, …)
• Other hypotheses on DA?
Thank you!