Verification of dynamic systems with locks and variables

Corto Mascle
joint work with Anca Muscholl, Igor Walukiewicz

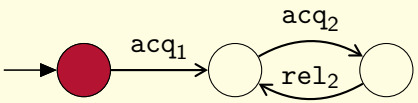
PaVeDys

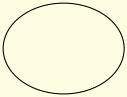
Verification of dynamic systems with locks and variables

Corto Mascle
joint work with Anca Muscholl, Igor Walukiewicz

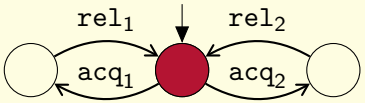
PaVeDys

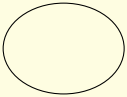
Lock-sharing systems[1]





P



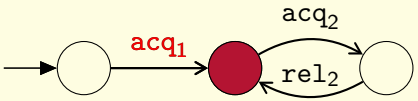


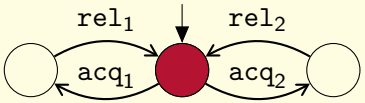
Q

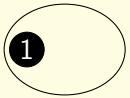
❶ ❷

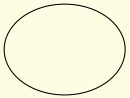
[1] Kahlon, Ivancic, Gupta CAV 2005

Lock-sharing systems[1]



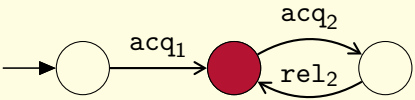


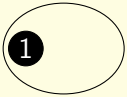




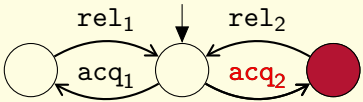
[1] Kahlon, Ivancic, Gupta CAV 2005

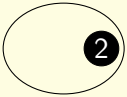
Lock-sharing systems[1]





P

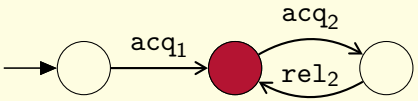


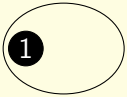


Q

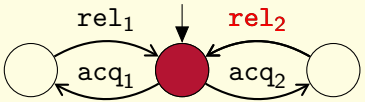
[1] Kahlon, Ivancic, Gupta CAV 2005

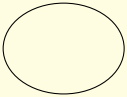
Lock-sharing systems[1]





P



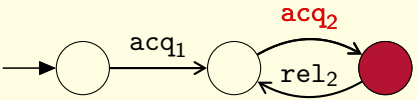


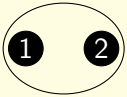
Q

2

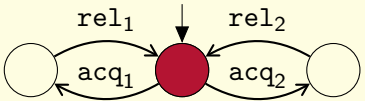
[1] Kahlon, Ivancic, Gupta CAV 2005

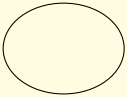
Lock-sharing systems[1]





P



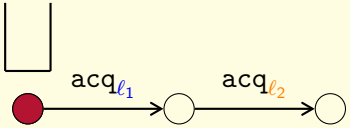


Q

[1] Kahlon, Ivancic, Gupta CAV 2005

Restriction: Nested locking

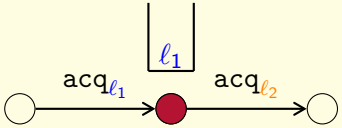
All processes acquire and release locks in a **stack-like order**, i.e., a process can only release the lock it acquired the latest.



This restricts communication between processes.

Restriction: Nested locking

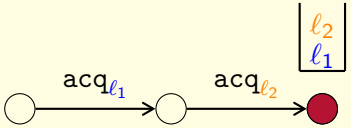
All processes acquire and release locks in a **stack-like order**, i.e., a process can only release the lock it acquired the latest.



This restricts communication between processes.

Restriction: Nested locking

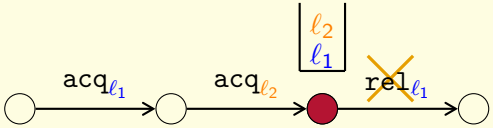
All processes acquire and release locks in a **stack-like order**, i.e., a process can only release the lock it acquired the latest.



This restricts communication between processes.

Restriction: Nested locking

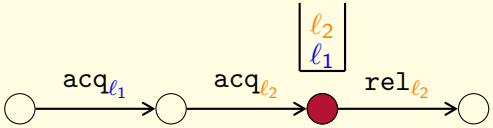
All processes acquire and release locks in a **stack-like order**, i.e., a process can only release the lock it acquired the latest.



This restricts communication between processes.

Restriction: Nested locking

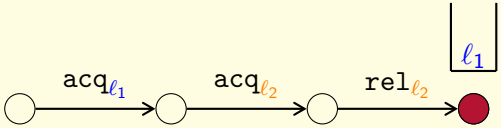
All processes acquire and release locks in a **stack-like order**, i.e., a process can only release the lock it acquired the latest.



This restricts communication between processes.

Restriction: Nested locking

All processes acquire and release locks in a **stack-like order**, i.e., a process can only release the lock it acquired the latest.



This restricts communication between processes.

Dynamic LSS

▷ We want to allow an unbounded number of processes and locks.

Dynamic LSS

- We want to allow an unbounded number of processes and locks.
- A process can spawn other processes

Dynamic LSS

▷ We want to allow an unbounded number of processes and locks.

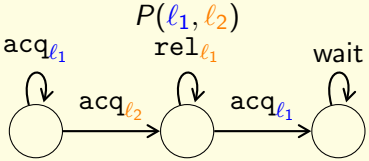
▷ A process can spawn other processes

▷ A process takes parameters, represented by *lock variables*

$Proc = \{P(\ell_1, \ell_2), Q(\ell_1, \ell_2, \ell_3), R(), ...\}$

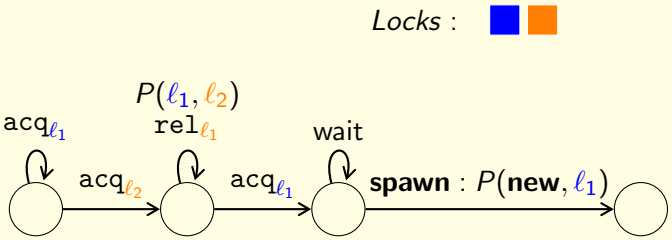
Dynamic LSS[2]

Locks : 



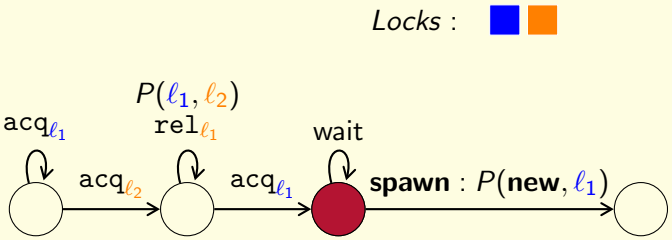
[2] Bouajjani, Müller-Olm, Touili, CONCUR 2005 + Kenter's thesis 2022

Dynamic LSS[2]



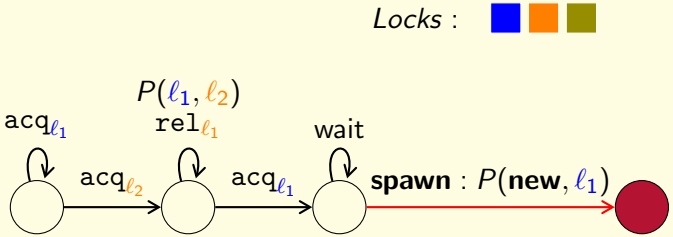
[2] Bouajjani, Müller-Olm, Touili, CONCUR 2005 + Kenter's thesis 2022

Dynamic LSS[2]



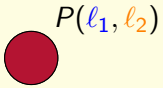
[2]Bouajjani, Müller-Olm, Touili, CONCUR 2005 + Kenter's thesis 2022

Dynamic LSS[2]

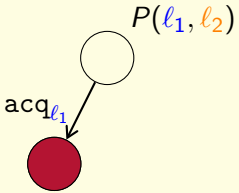


[2] Bouajjani, Müller-Olm, Touili, CONCUR 2005 + Kenter's thesis 2022

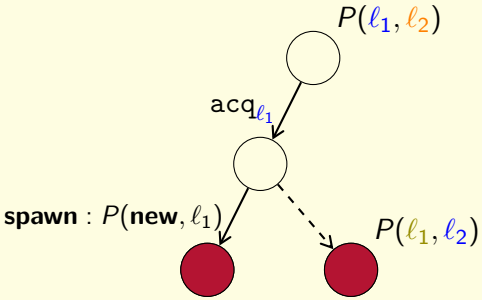
Tree representation



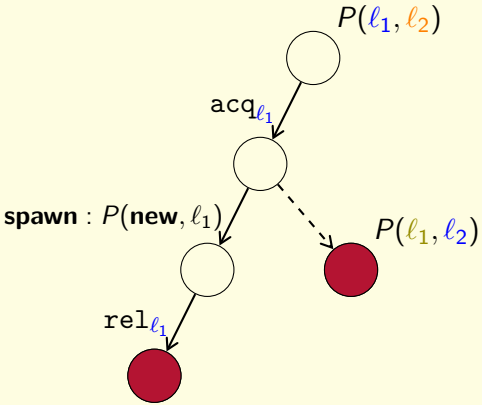
Tree representation



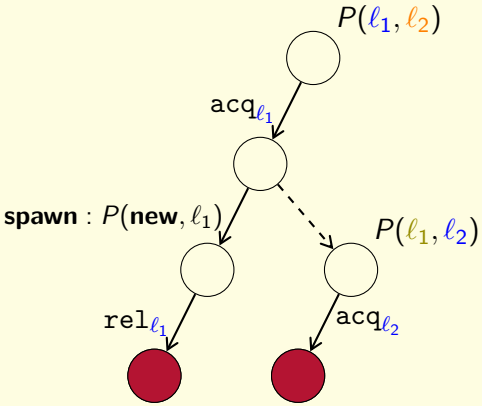
Tree representation



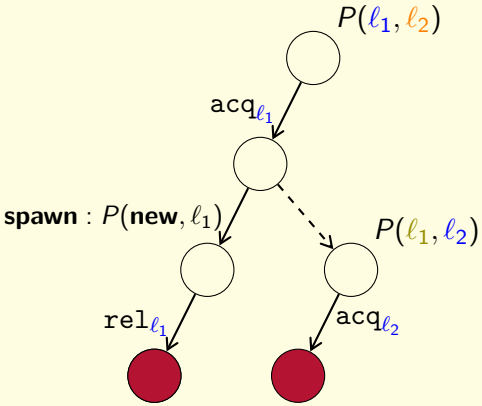
Tree representation



Tree representation

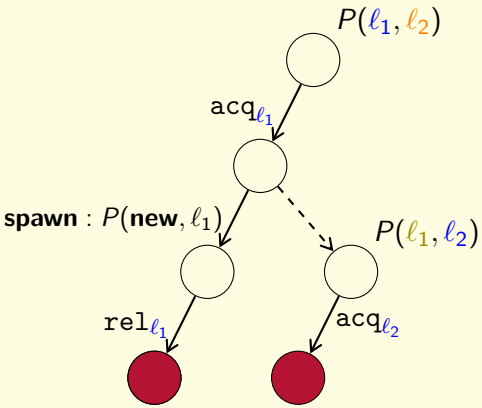


Tree representation



Specifications are ω-regular tree languages.

Tree representation



Specifications are ω-regular tree languages.

"Every process is blocked after some point"

"Finitely many processes are spawned"

"Infinitely many processes reach an error state q_{err}"

Deadlocks

Regular model-checking problem

Input: A DLSS $\mathcal{D}$ and a parity tree automaton $\mathcal{A}$.
Output: Is there a run of $\mathcal{D}$ accepted by $\mathcal{A}$?

Regular model-checking problem

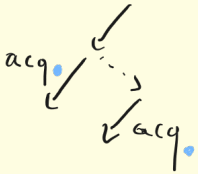
Input: A DLSS $\mathcal{D}$ and a parity tree automaton $\mathcal{A}$.
Output: Is there a run of $\mathcal{D}$ accepted by $\mathcal{A}$?

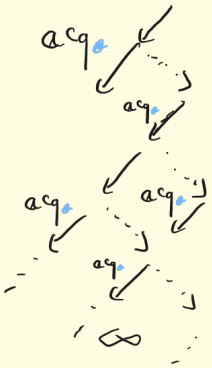
Problem: characterise trees that represent actual executions.

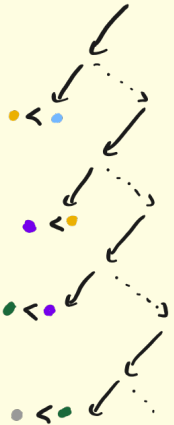
Regular model-checking problem

Input: A DLSS $\mathcal{D}$ and a parity tree automaton $\mathcal{A}$.
Output: Is there a run of $\mathcal{D}$ accepted by $\mathcal{A}$?

Problem: characterise trees that represent actual executions.





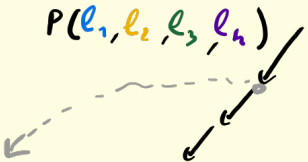


Lemma

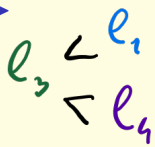
The set of execution trees of a DLSS is recognised by a Büchi tree automaton of exponential size.

Lemma

The set of execution trees of a DLSS is recognised by a Büchi tree automaton of exponential size.

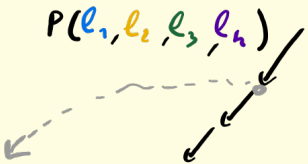


For each node we guess a label of the form

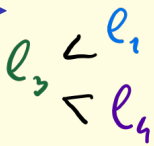
- "ℓ_1 is taken and will never be released", "ℓ_2 will be acquired infinitely many times", ...
- 

Lemma

The set of execution trees of a DLSS is recognised by a Büchi tree automaton of exponential size.



For each node we guess a label of the form

- "ℓ_1 is taken and will never be released", "ℓ_2 will be acquired infinitely many times", ...
- 

The automaton checks that:

- the labels are consistent
- There exists a well-founded linear ordering on locks in which all local orders embed. (Technical part, also see related work [Demri Quaas, Concur '23])

Theorem [M., Muscholl, Walukiewicz Concur 2023]

Regular model-checking of DLSS is EXPTIME-complete, and
PTIME for fixed number of locks per process and parity index.

Theorem [M., Muscholl, Walukiewicz Concur 2023]

Regular model-checking of DLSS is EXPTIME-complete, and
PTIME for fixed number of locks per process and parity index.

What about pushdown processes?

Right-resetting pushdown tree automata

Right-resetting = the stack is emptied every time we go to a right child.

Lemma

Emptiness is decidable in PTIME for right-resetting parity pushdown tree automata when the parity index is fixed.

Right-resetting pushdown tree automata

Right-resetting = the stack is emptied every time we go to a right child.

Lemma

Emptiness is decidable in PTIME for right-resetting parity pushdown tree automata when the parity index is fixed.

Theorem

Regular model-checking of nested **pushdown** DLSS is EXPTIME-complete,
and PTIME when the parity index and the number of locks per process are fixed.

Right-resetting pushdown tree automata

Right-resetting = the stack is emptied every time we go to a right child.

Lemma

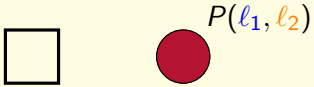
Emptiness is decidable in PTIME for right-resetting parity pushdown tree automata when the parity index is fixed.

Theorem

Regular model-checking of nested **pushdown** DLSS is EXPTIME-complete,
and PTIME when the parity index and the number of locks per process are fixed.

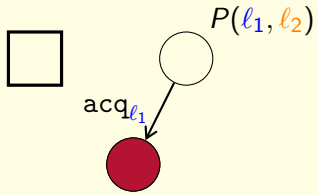
What about shared variables?

DLSS with variables



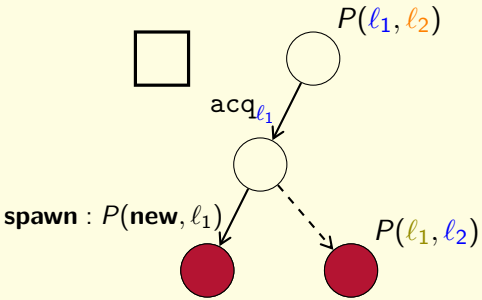
We add a register and operations `wr` and `rd` writing and reading letters from a finite alphabet in the register.

DLSS with variables



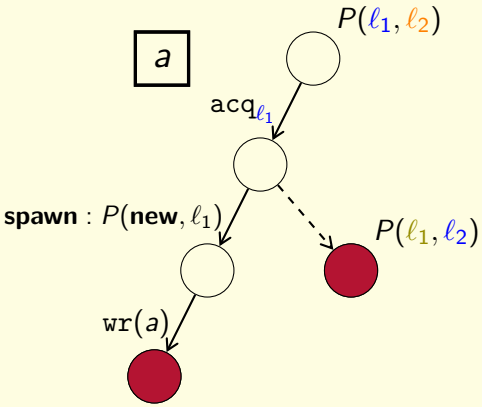
We add a register and operations `wr` and `rd` writing and reading letters from a finite alphabet in the register.

DLSS with variables



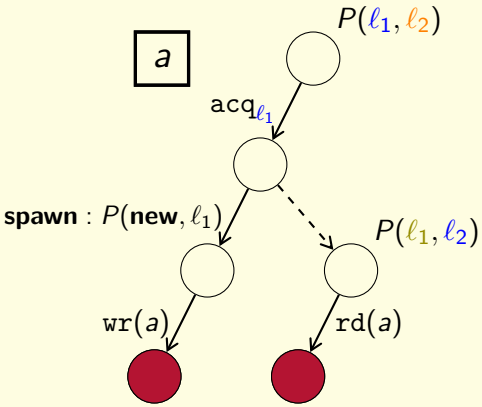
We add a register and operations `wr` and `rd` writing and reading letters from a finite alphabet in the register.

DLSS with variables



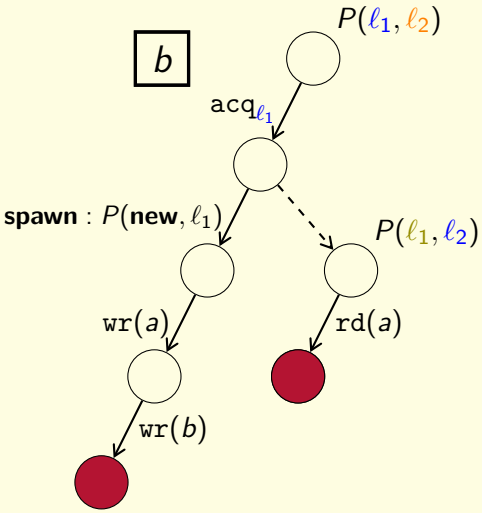
We add a register and operations `wr` and `rd` writing and reading letters from a finite alphabet in the register.

DLSS with variables



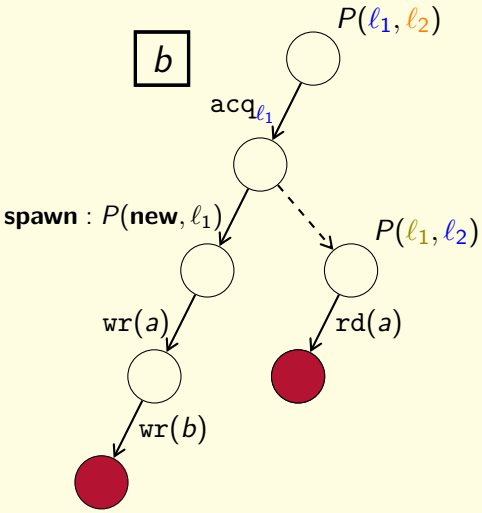
We add a register and operations `wr` and `rd` writing and reading letters from a finite alphabet in the register.

DLSS with variables



We add a register and operations `wr` and `rd` writing and reading letters from a finite alphabet in the register.

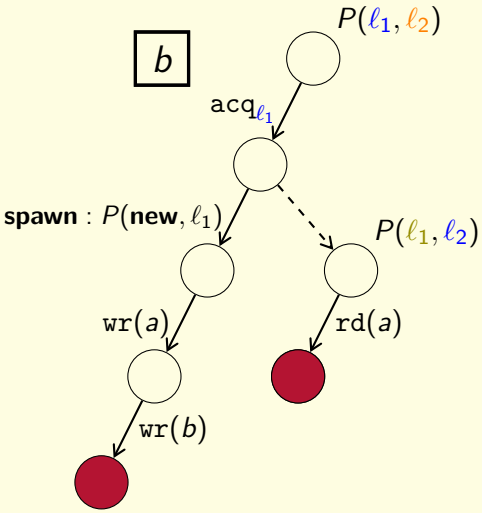
DLSS with variables



We add a register and operations `wr` and `rd` writing and reading letters from a finite alphabet in the register.

Sets of runs are no longer regular.

DLSS with variables



We add a register and operations `wr` and `rd` writing and reading letters from a finite alphabet in the register.

Sets of runs are no longer regular.

Theorem

State reachability is undecidable for DLSS with variables.

Bounded writer reversals

Writer reversal = the process writing in the shared register changes.

[3] Atig, Bouajjani, Kumar, Saivasan FSTTCS 2014

Bounded writer reversals

Writer reversal = the process writing in the shared register changes.

Theorem

State reachability is decidable for DLSSV with bounded writer reversals.

[3] Atig, Bouajjani, Kumar, Saivasan FSTTCS 2014

Bounded writer reversals

Writer reversal = the process writing in the shared register changes.

Theorem

State reachability is decidable for DLSSV with bounded writer reversals.

It is undecidable when the processes are pushdown systems[3].

[3] Atig, Bouajjani, Kumar, Saivasan FSTTCS 2014

Proof sketch

Consider a run with one process writing and others reading.

Proof sketch

Consider a run with one process writing and others reading.

Phase: run section where

- the writer is in the same state and has the same locks at the start and at the end,
- none of the locks used by the writers in the phase are held by another process at the start or the end

Proof sketch

Consider a run with one process writing and others reading.

Phase: run section where

- the writer is in the same state and has the same locks at the start and at the end,
- none of the locks used by the writers in the phase are held by another process at the start or the end

Lemma

Every finite run with a single writer can be cut into $2^{O(|Q|)}$ phases.

Proof sketch

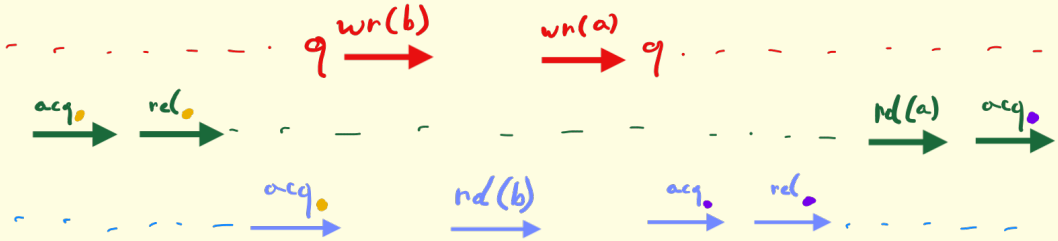
Consider one phase.

Proof sketch

Consider one phase.

Lemma

Every phase can be replaced by a sequence of phases where at most one reader moves.

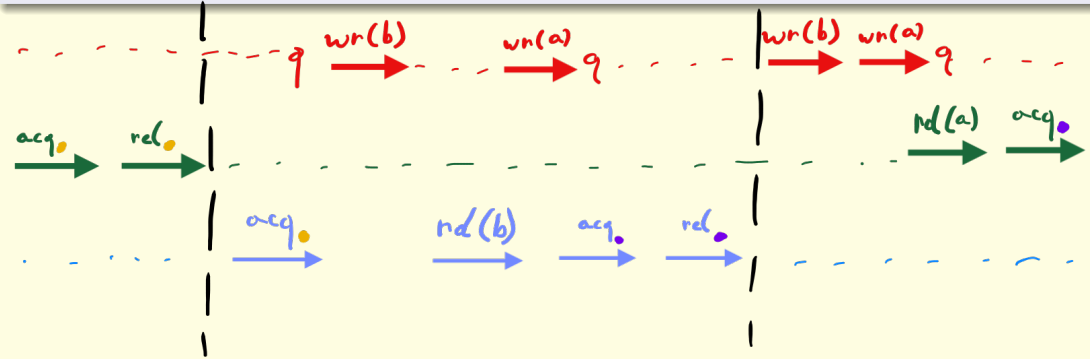


Proof sketch

Consider one phase.

Lemma

Every phase can be replaced by a sequence of phases where at most one reader moves.



Proof sketch

Consider one phase.

Lemma

Every phase can be replaced by a sequence of phases where at most one reader moves.

Proof sketch

Consider one phase.

Lemma

Every phase can be replaced by a sequence of phases where at most one reader moves.

Construct $\mathcal{A}$ that:

- guesses a partition of the tree in $K2^{O(|Q|)}$ phases, each with a single writer.
- checks lock conditions
- checks compatibility of each reader with the writer

Proof sketch

Consider one phase.

Lemma

Every phase can be replaced by a sequence of phases where at most one reader moves.

Construct $\mathcal{A}$ that:

- guesses a partition of the tree in $K2^{O(|Q|)}$ phases, each with a single writer.
- checks lock conditions
- checks compatibility of each reader with the writer

What is left to do

Conjecture

Verification of DLSSV against ω-regular tree specifications is decidable.

What is left to do

Conjecture

Verification of DLSSV against ω-regular tree specifications is decidable.

- Controller synthesis: local strategies enforcing the specification
- Parameterised complexity w.r.t. the number of locks per process: everything is in XP, can we do better?

What is left to do

Conjecture

Verification of DLSSV against ω-regular tree specifications is decidable.

- Controller synthesis: local strategies enforcing the specification
- Parameterised complexity w.r.t. the number of locks per process: everything is in XP, can we do better?

Thanks!