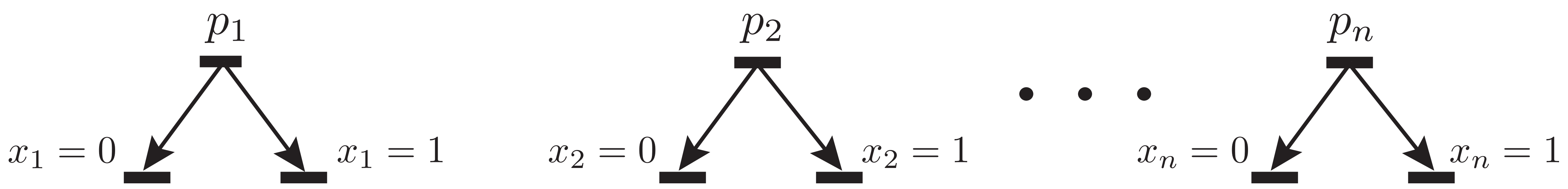
Concurrent systems need not be hard

Igor Walukiewicz

joint work with:
Hugo Gimbert,
Corto Mascle,
Anca Muscholl

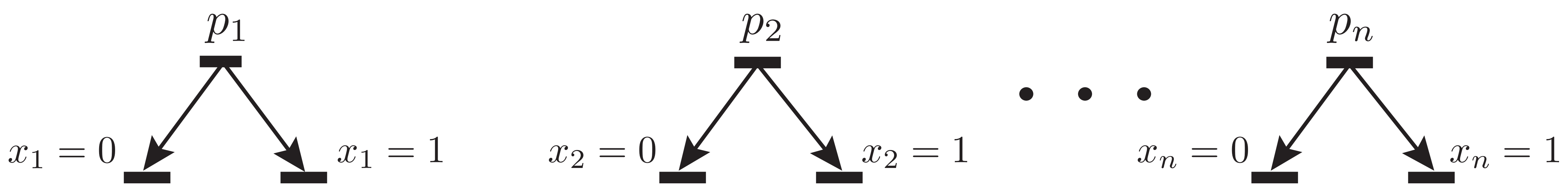
Verification of concurrent systems is hard

Verification of concurrent systems suffers from the state explosion problem.



Verification of concurrent systems is hard

Verification of concurrent systems suffers from the state explosion problem.

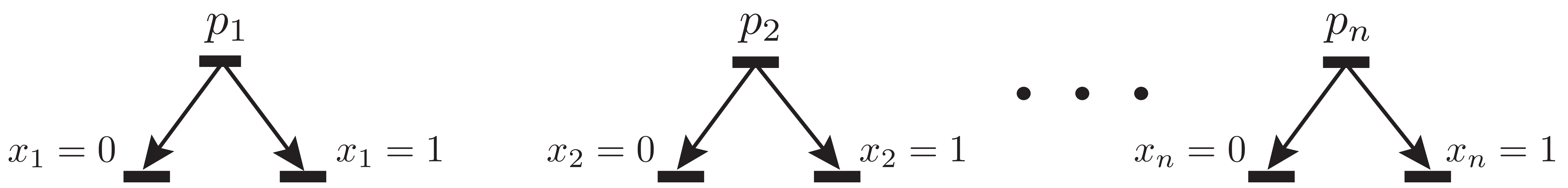


But this is not just state explosion problem that makes it hard.

It is complex interactions between components that make verification hard.

Verification of concurrent systems is hard

Verification of concurrent systems suffers from the state explosion problem.

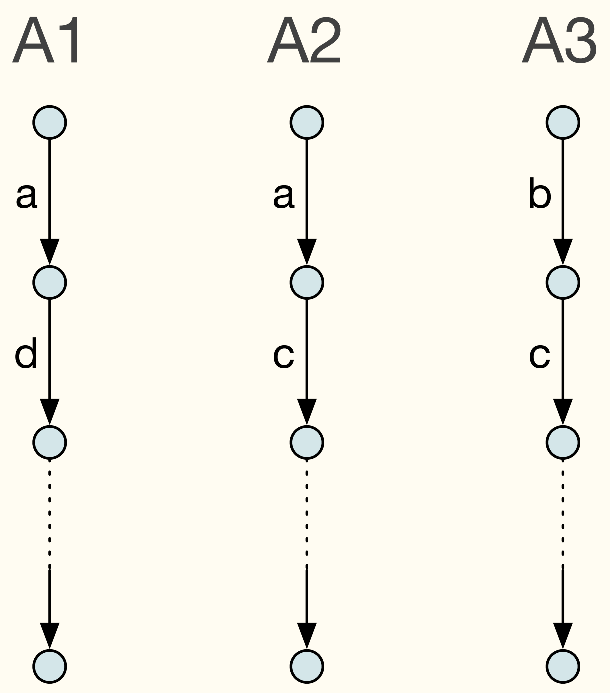


But this is not just state explosion problem that makes it hard.

It is complex interactions between components that make verification hard.

We will see examples of systems where many (but not all) problems become easy.
(NP or PTIME)

Verification of concurrent systems is PSPACE-complete



Reachability for the synchronous product of finite automata is PSPACE-complete.

What can we do about this?

- symbolic methods: SAT, BDD
- partial-order methods, stateless exploration

Maybe there are concurrent systems for which verification is easier?

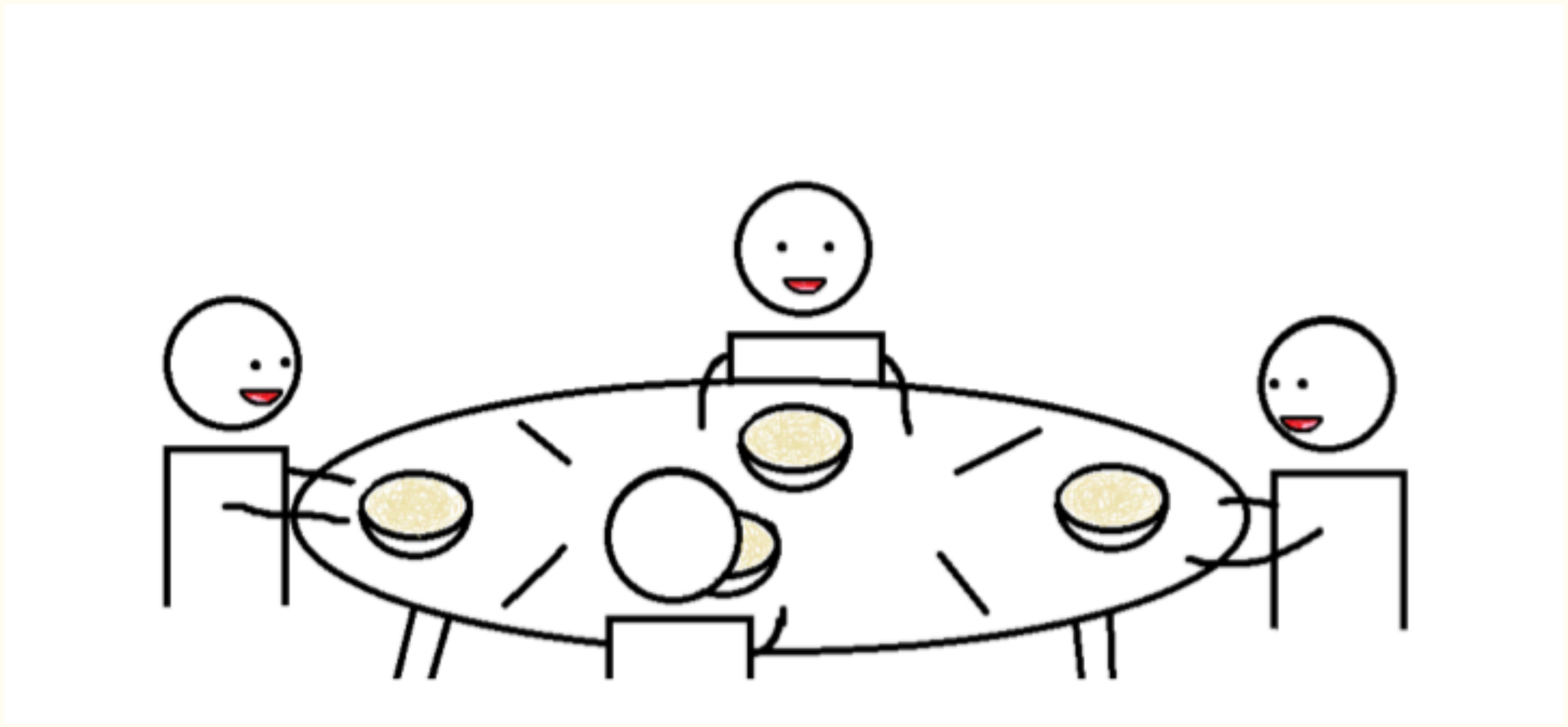
Two examples:

Systems with 2-locks: 2 locks per process, no shared variables.

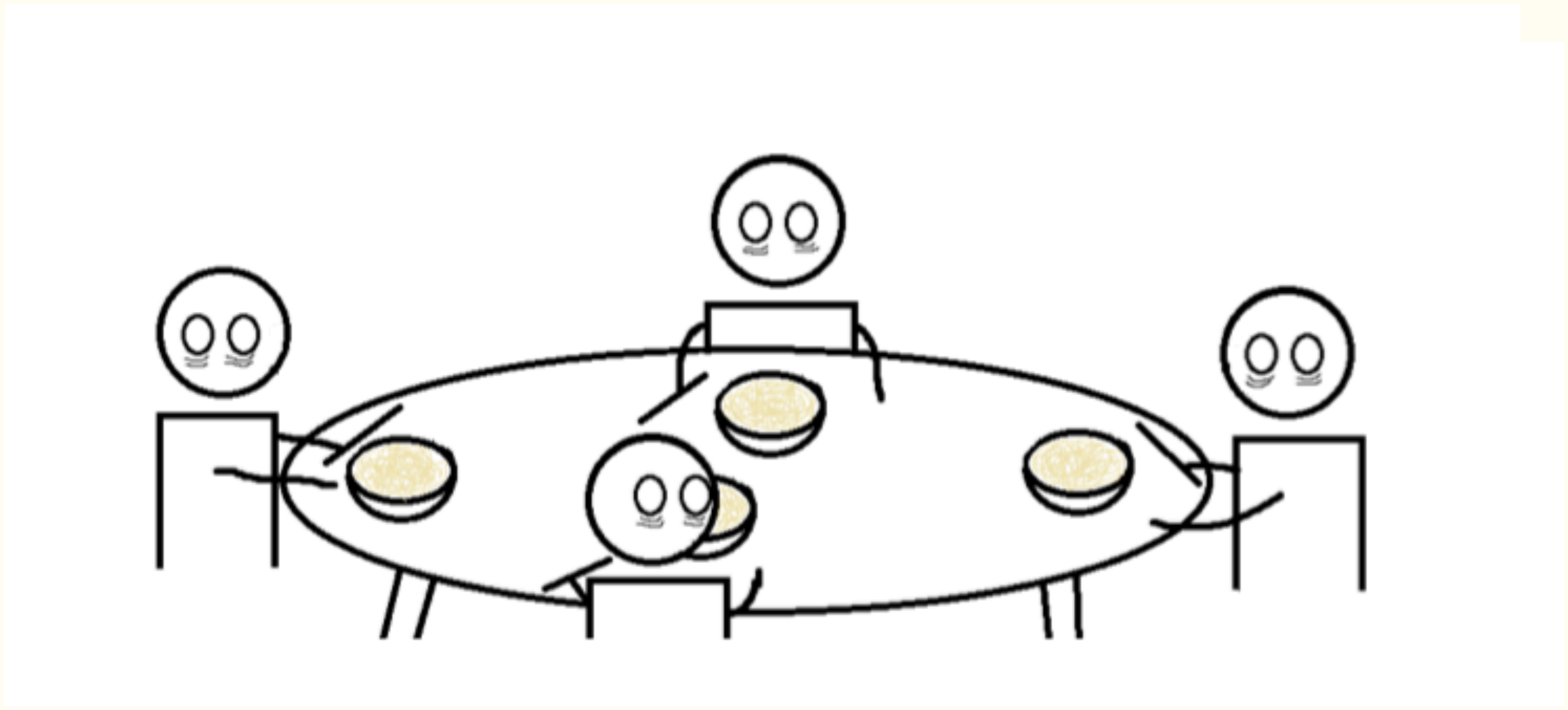
Negotiations: a restricted form of an asynchronous automaton or a Petri net.

It is not the case that the standard algorithms are faster for these systems.

Systems with locks: dinning philosophers



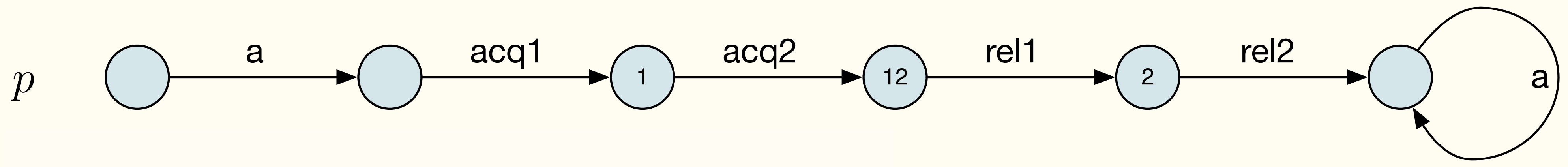
Systems with locks: dinning philosophers



Deadlock avoidance

Verification (no deadlock)

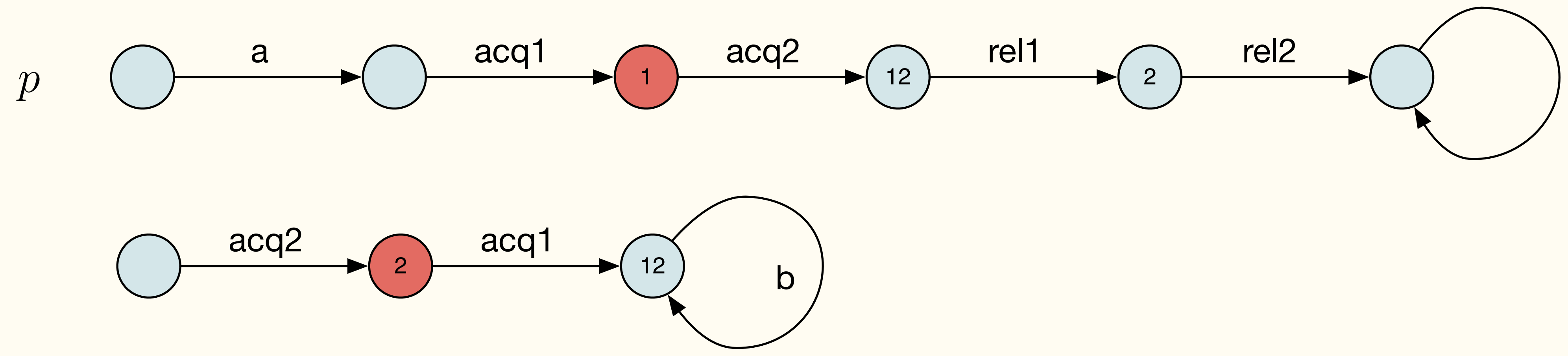
Systems with locks: definition



Systems with locks: definition

$$\begin{tikzcd}
p & {} \arrow[r, "\text{a}"] & {} \arrow[r, "\text{acq1}"] & 1 \arrow[r, "\text{acq2}"] & 12 \arrow[r, "\text{rel1}"] & 2 \arrow[r, "\text{rel2}"] & {} \arrow[loop right, "\text{a}"] \\
q & {} \arrow[r, "\text{acq2}"] & 2 \arrow[r, "\text{acq1}"] & 12 \arrow[loop right, "\text{b}"] & & &
\end{tikzcd}$$

Systems with locks: definition



Deadlock: no process can do a transition

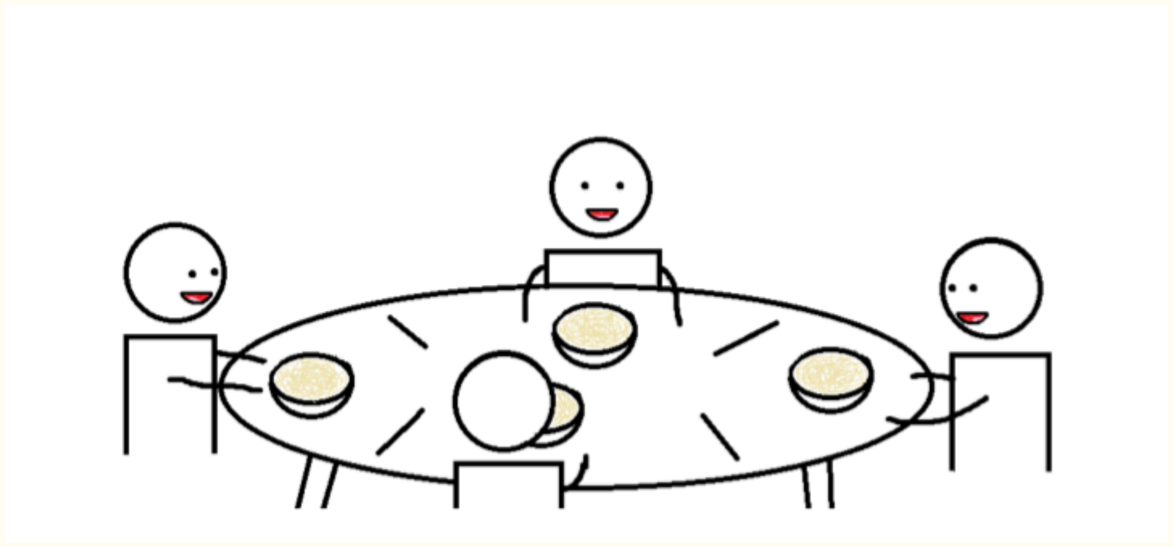
Deadlock avoidance problem:

given a lock system decide if it cannot reach a deadlock

Systems with locks verification

Fact: Deadlock avoidance problem is PSPACE-complete.

2-lock systems: every process has access to at most 2 locks.



Thm: For 2-lock systems the problem is coNP-complete.

Thm: For 2-lock **locally live** systems the problem is in PTIME.

Patterns

A **pattern of a run:** we look at last operations on locks

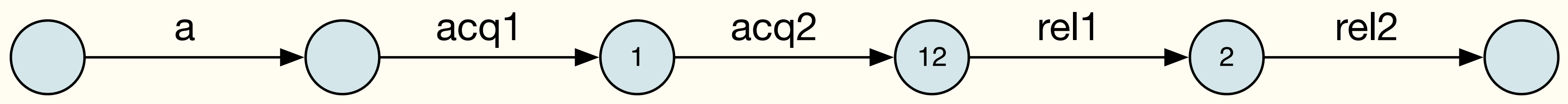
rel1, rel2

acq1, acq2

rel2, acq1

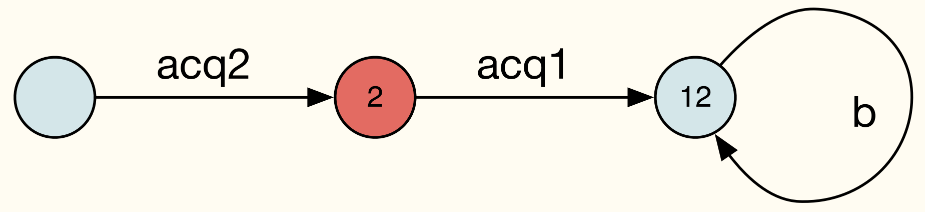
acq1, rel2

Every run is resumed by one pattern



rel1, rel2

q



acq2, acq1

Patterns

A **pattern of a run:** we look at last operations on locks

rel1, rel2

acq1, acq2

rel2, acq1

acq1, rel2

There are 8 possible patterns (for 2 given locks).

Main idea: given a pattern for each local run we can decide if they can form a global run.

(total order on locks compatible with patterns)

Patterns

A **pattern of a run:** we look at last operations on locks

rel1, rel2

acq1, acq2

rel2, acq1

acq1, rel2

Main idea: given a pattern for each local run we can decide if they can form a global run.

(total order on locks compatible with patterns)

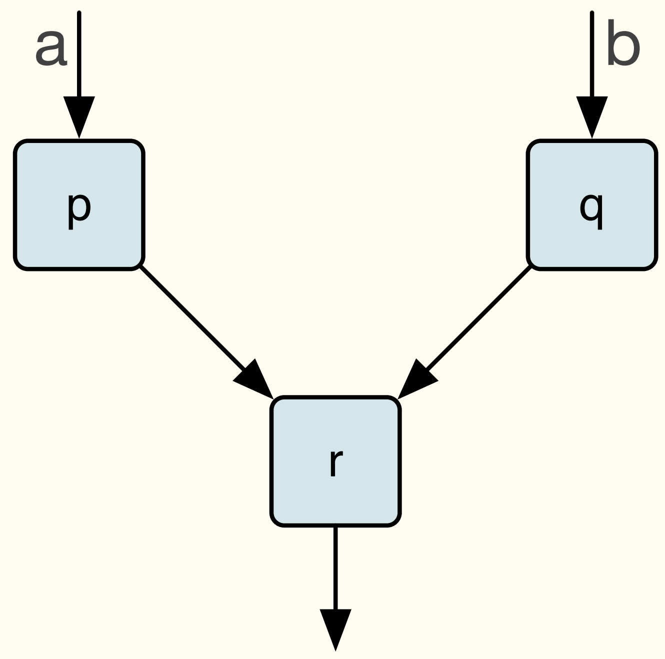
Thm: For 2-lock systems the problem is coNP-complete.

Thm: For 2-lock **locally live** systems the problem is in PTIME.

Synthesis

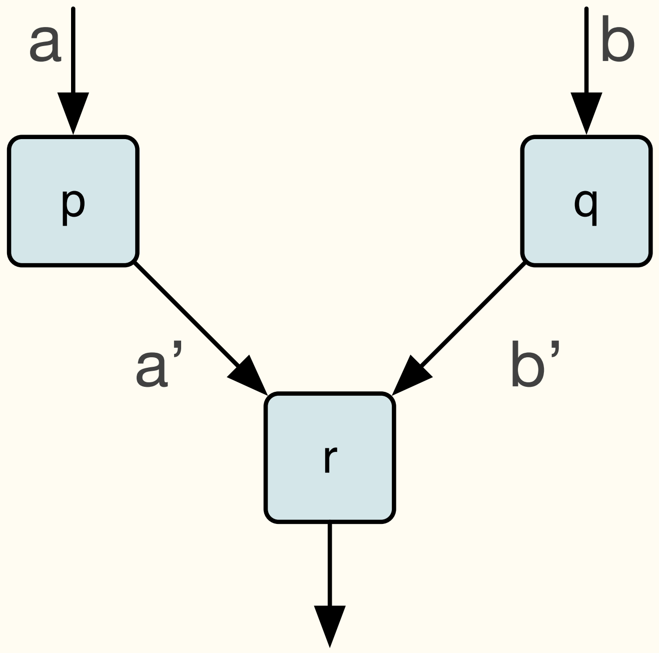
Synthesis for distributed systems is seldom decidable

Pnueli-Rosner setting



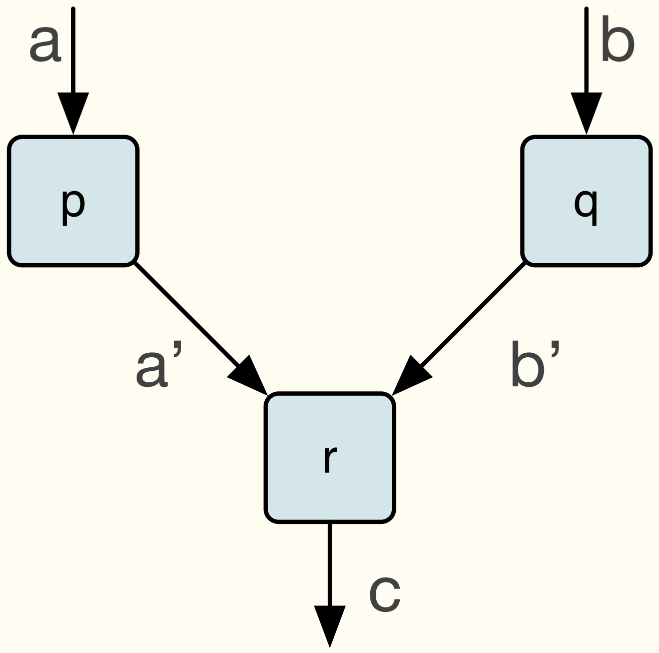
Synthesis for distributed systems is seldom decidable

Pnueli-Rosner setting



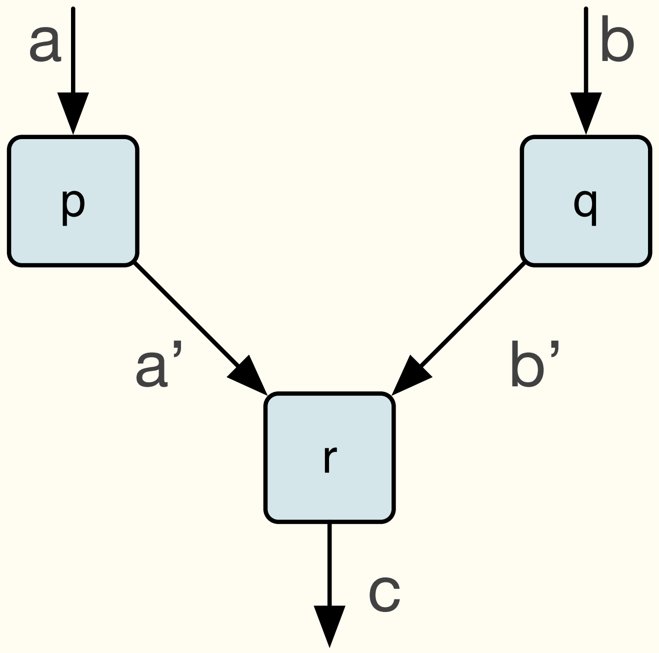
Synthesis for distributed systems is seldom decidable

Pnueli-Rosner setting



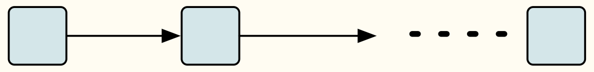
Synthesis for distributed systems is seldom decidable

Pnueli-Rosner setting



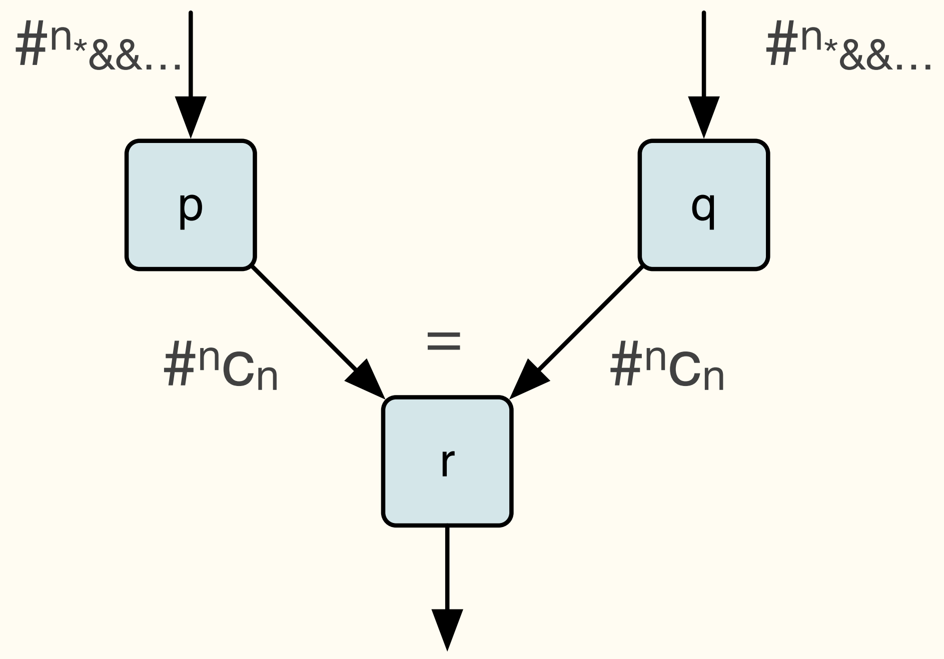
Specification is a relation between inputs and outputs

Thm [Madhusudan, Thiagarajan**]:** Synthesis in this setting is decidable only for pipeline architectures.



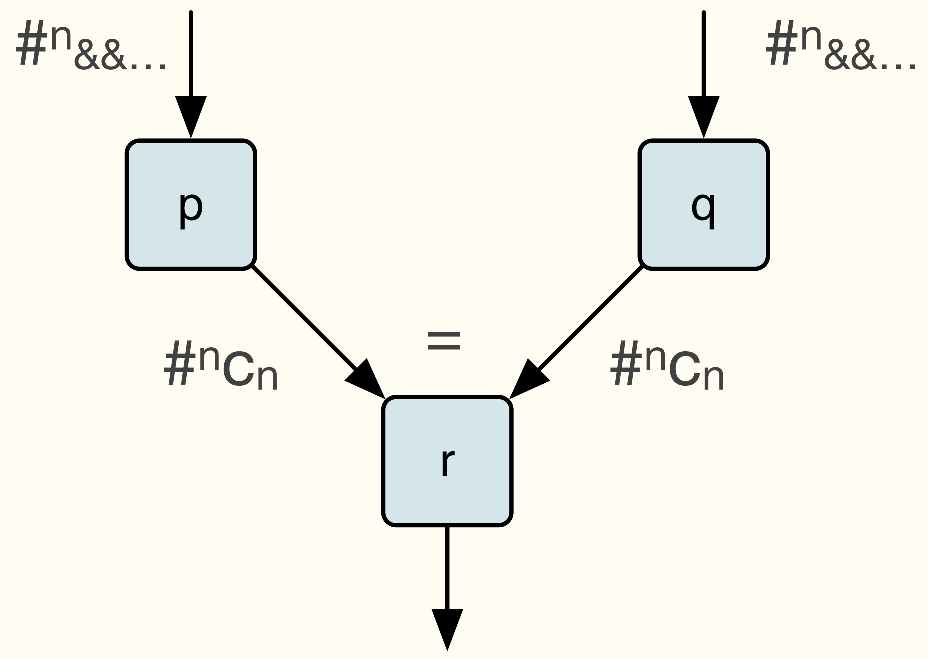
Synthesis for distributed systems is seldom decidable

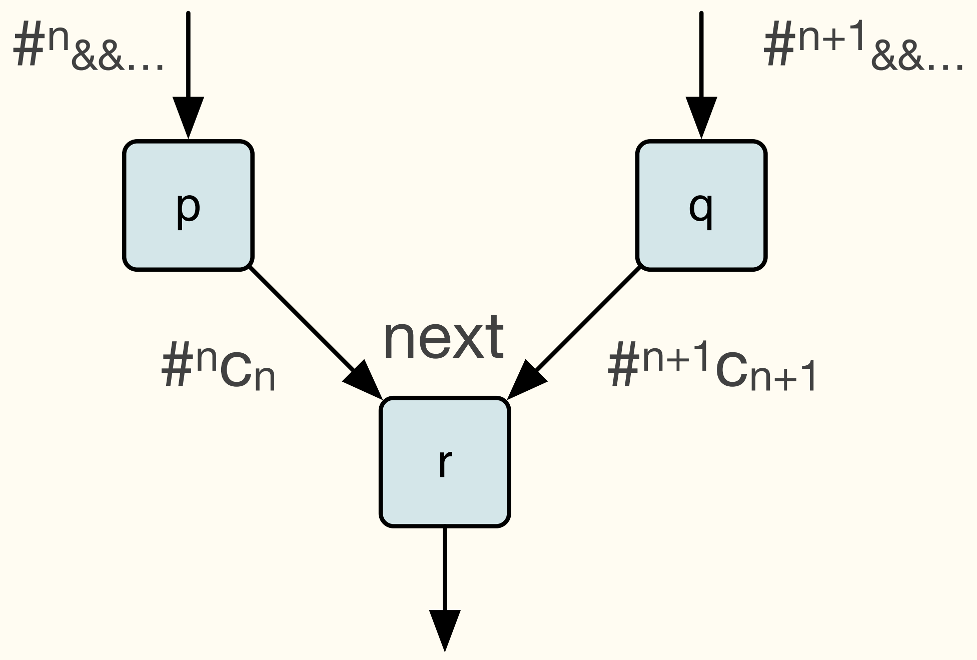
Pnueli-Rosner setting



Synthesis for distributed systems is seldom decidable

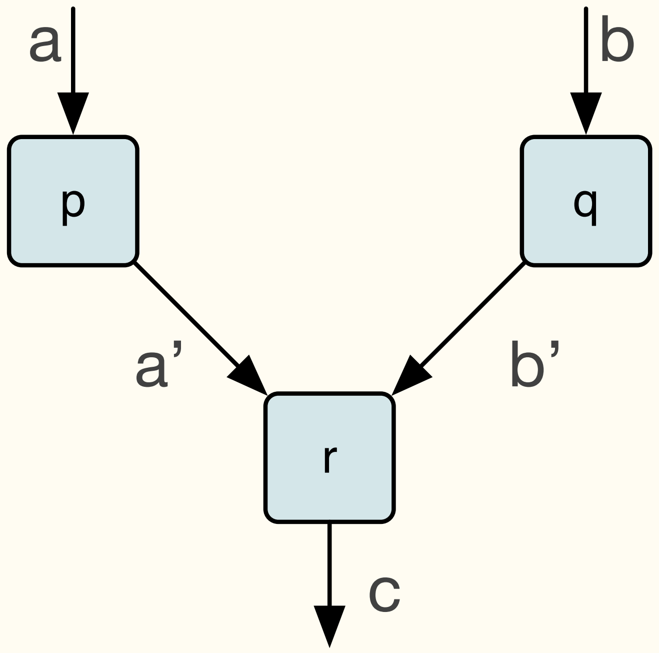
Pnueli-Rosner setting



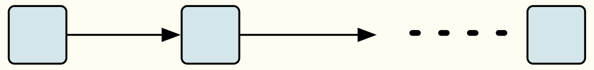


Synthesis for distributed systems is seldom decidable

Pnueli-Rosner setting



Thm [Madhusudan, Thiagarajan**]:** Synthesis in this setting is decidable only for pipeline architectures.



Thm [Gimbert]: In general synthesis in causal memory settings is undecidable.

Systems with locks: synthesis

Actions are split into **system** and **environment** actions.

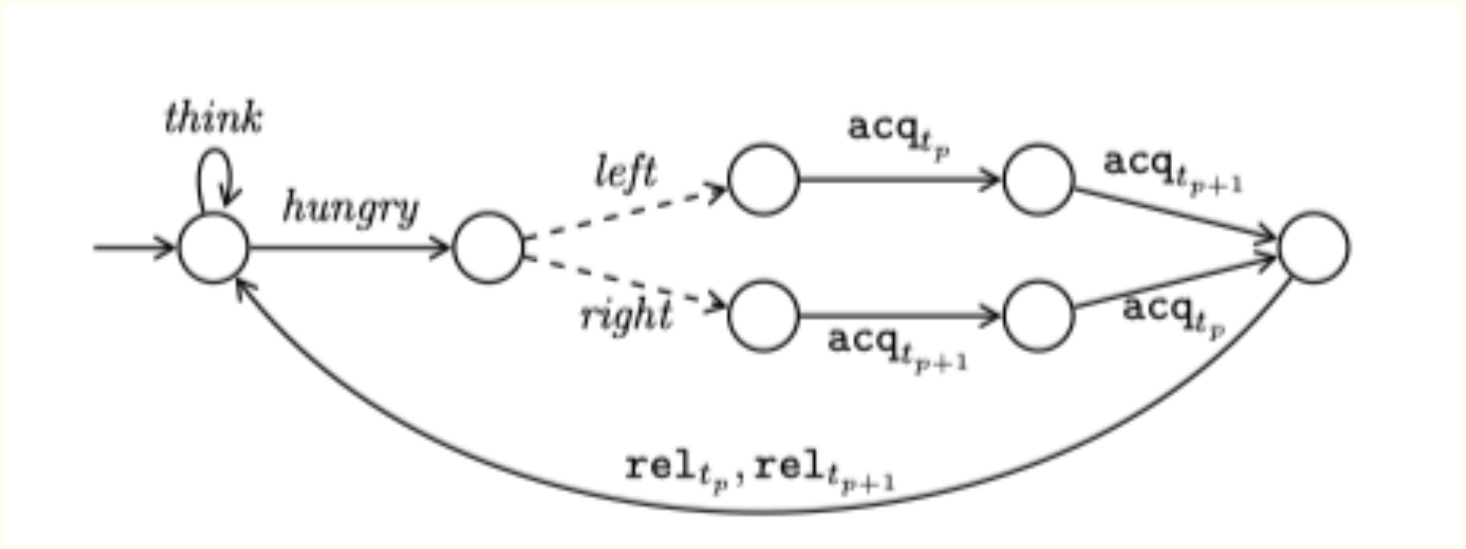
$$\Sigma_p = \Sigma_p^s \cup \Sigma_p^e \qquad \text{for every process } p$$

Local strategy: $\sigma_p : \Sigma_p^* \to \mathcal{P}(\Sigma_p)$ provided $\Sigma_p^e \subseteq \sigma_p(u)$ for all u.

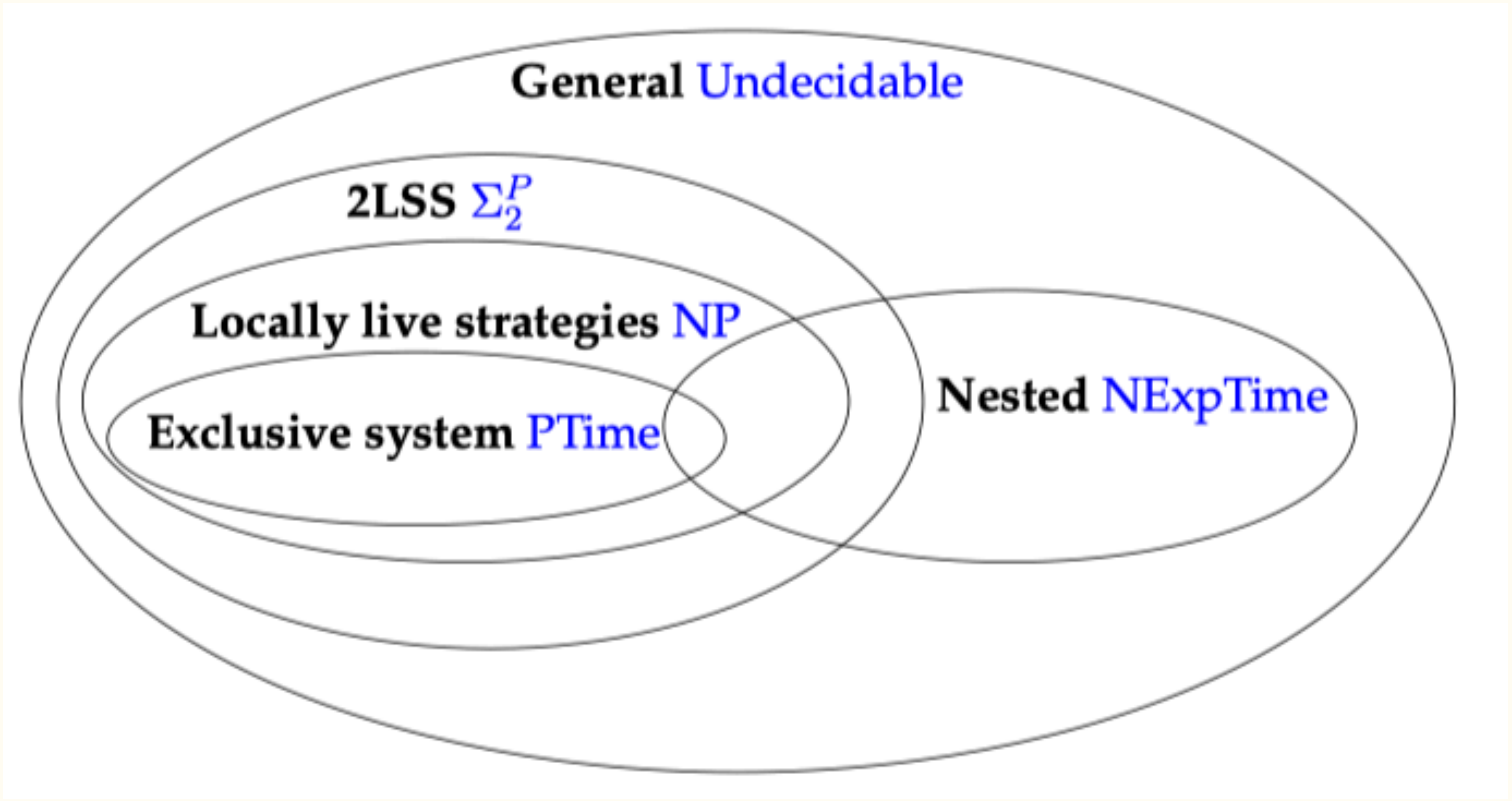
Distributed strategy: $\{\sigma_p\}_{p \in Proc}$

Synthesis problem: given a system decide if there is a distributed strategy s.t.

no run respecting this strategy reaches a deadlock.



Systems with locks: synthesis



Synthesis for 2-lock systems

2-lock systems: every process has access to 2 locks

A **pattern of a run:** we look at last operations on locks

rel1, rel2

acq1, acq2

rel2, acq1

acq1, rel2

Prop: Given a set of patterns B for a process p we can check in PTIME if there is a local strategy σ_p allowing only runs with these patterns.

A Σ_2^P-algorithm:

- Guess a set of patterns B_p for every process p.
- Check in PTIME if there is a local strategy allowing only these patterns.
- Verify in coNP that these patterns do not give a deadlock.

Thm: The synthesis problem for 2-lock systems is Σ_2^p-complete.

Synthesis for 2-lock systems: locally-live strategies

2-lock systems: every process has access to 2 locks

A **pattern of a run:** we look at last operations on locks

rel1, rel2

acq1, acq2

rel2, acq1

acq1, rel2

Prop: Given a set of patterns B for a process p we can check in PTIME if there is a local strategy σ_p allowing only runs with these patterns.

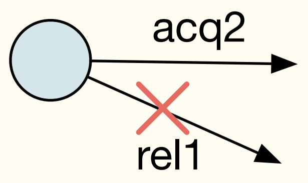
A NP-algorithm:

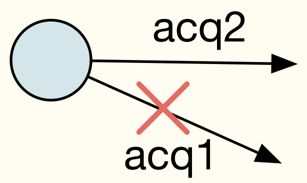
- Guess a set of patterns B_p for every process p.
- Check in PTIME if there is a local strategy allowing only these patterns.
- verify in PTIME that these patterns do not give a deadlock.

Thm: The synthesis problem for 2-lock systems and locally-live strategies is in NP.

Locally live strategies + exclusive systems

Exclusive systems: if there is an acquire action then there are no other actions





Thm: The synthesis problem for 2-lock exclusive systems and locally-live strategies is in PTIME

Conclusions

Classes of concurrent systems for which some problems can be decided efficiently, but by specialized algorithms.

Is it worth it, or a Swiss knife of SAT/BDD is always enough?

Perspectives:

Lock-systems: some other restrictions on lock usage?

Find other contexts where this phenomenon appears.

Applications?

[A Compositional Deadlock Detector for Android Java, Brotherson, Brunet, Gorogiannis, Kanovich, 2021]