Counting Abstraction for the Verification of Structured Parameterized Networks

Neven Villani

Marius Bozga, Radu Iosif, Arnaud Sangnier

2024-10-11

1. The setting

Goals

Prove correctness of distributed protocols (e.g. mutex, leader election)

- encoded as a network of **communicating** processes
- system of **arbitrary size**
- **fully automated** analysis

Difficulty:

- general problem is undecidable
- techniques to handle finitely many processes with infinite behaviors do not immediately translate to infinitely many processes

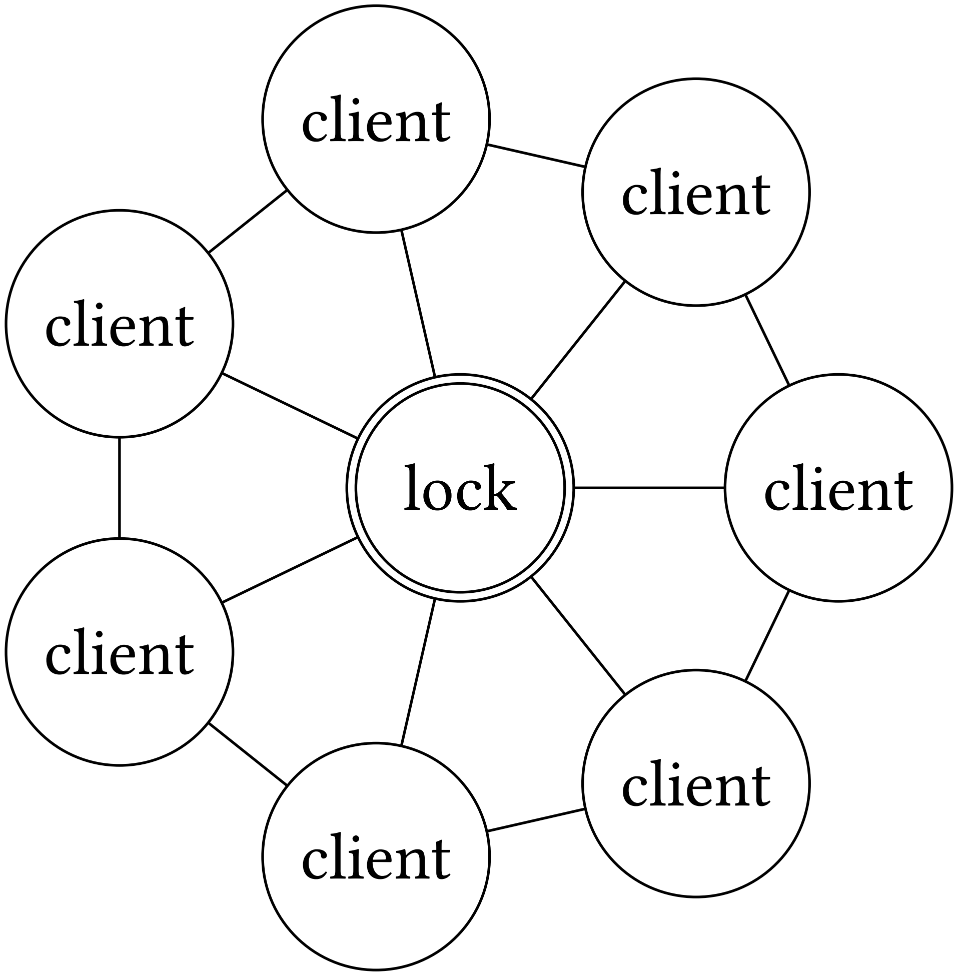
Contribution

Reduction technique

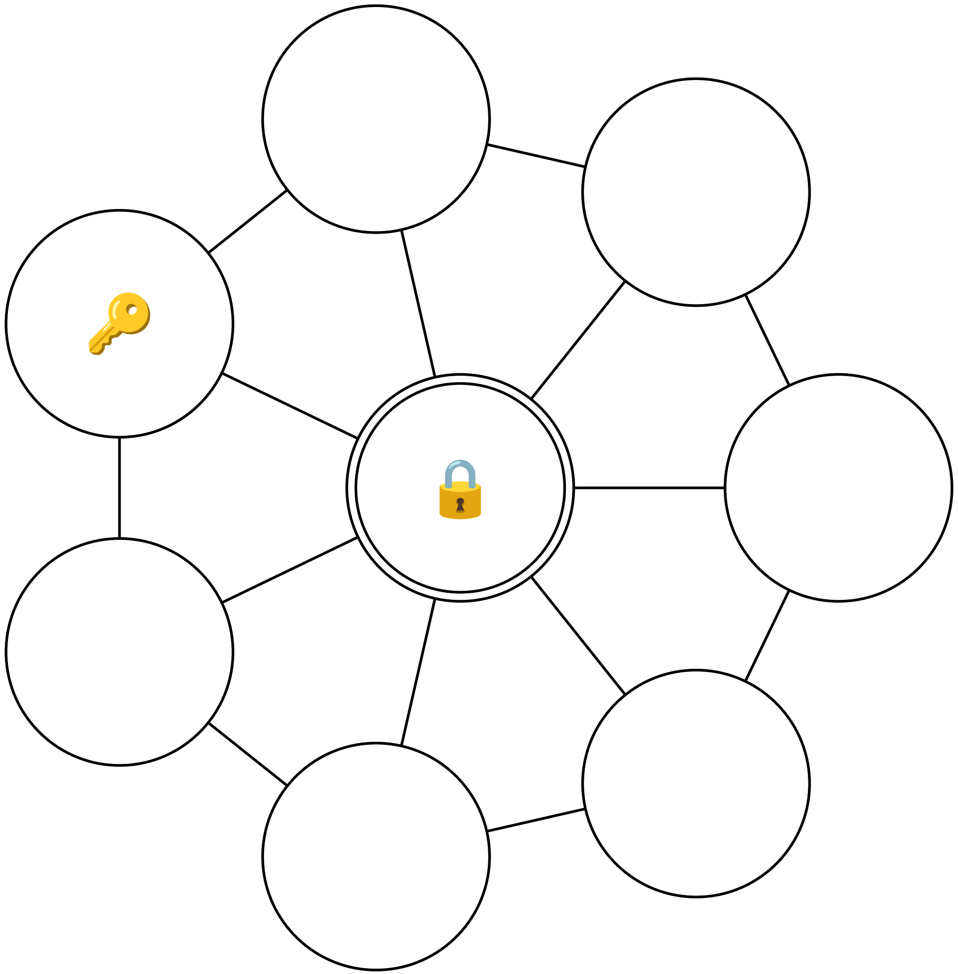
- from
  - infinite system
  - finite behaviors (1-safe Petri nets)
  - reachability problem
- to
  - finite systems
  - infinite behaviors (Petri nets)
  - coverability problem

with support for complex topologies,
fully automated, fully implemented.

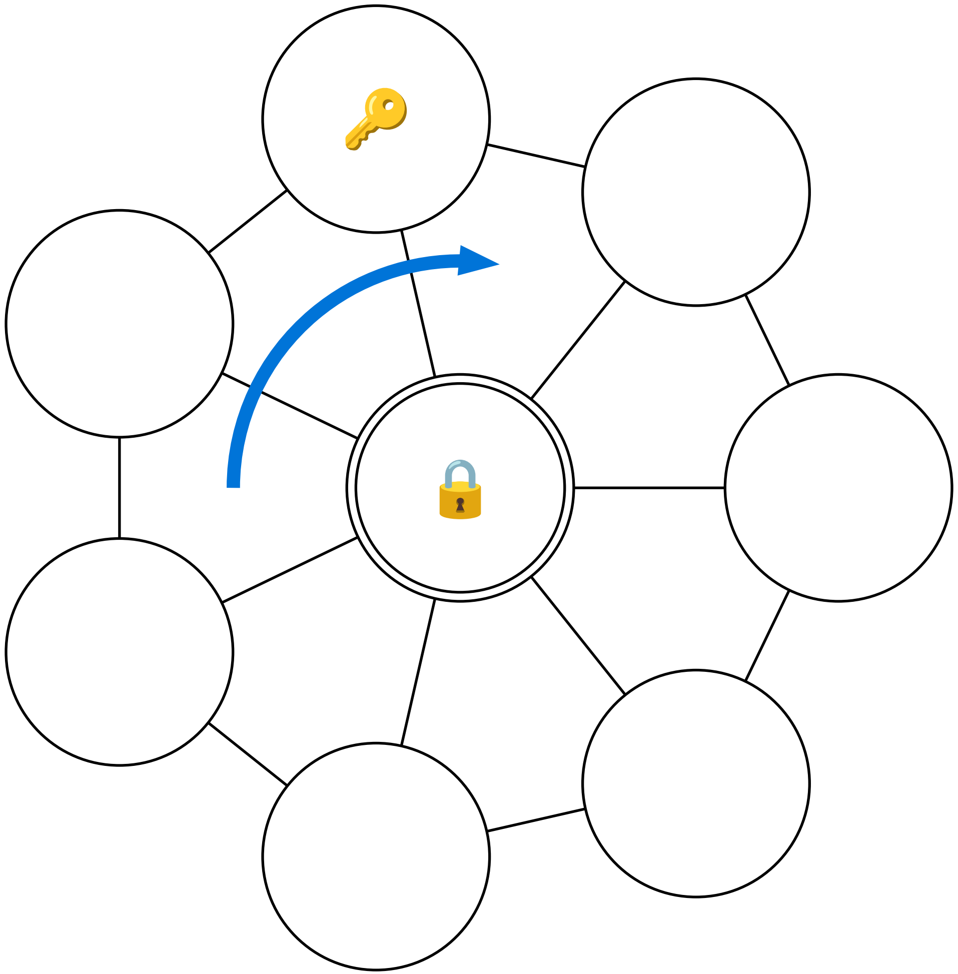
Token ring with resource



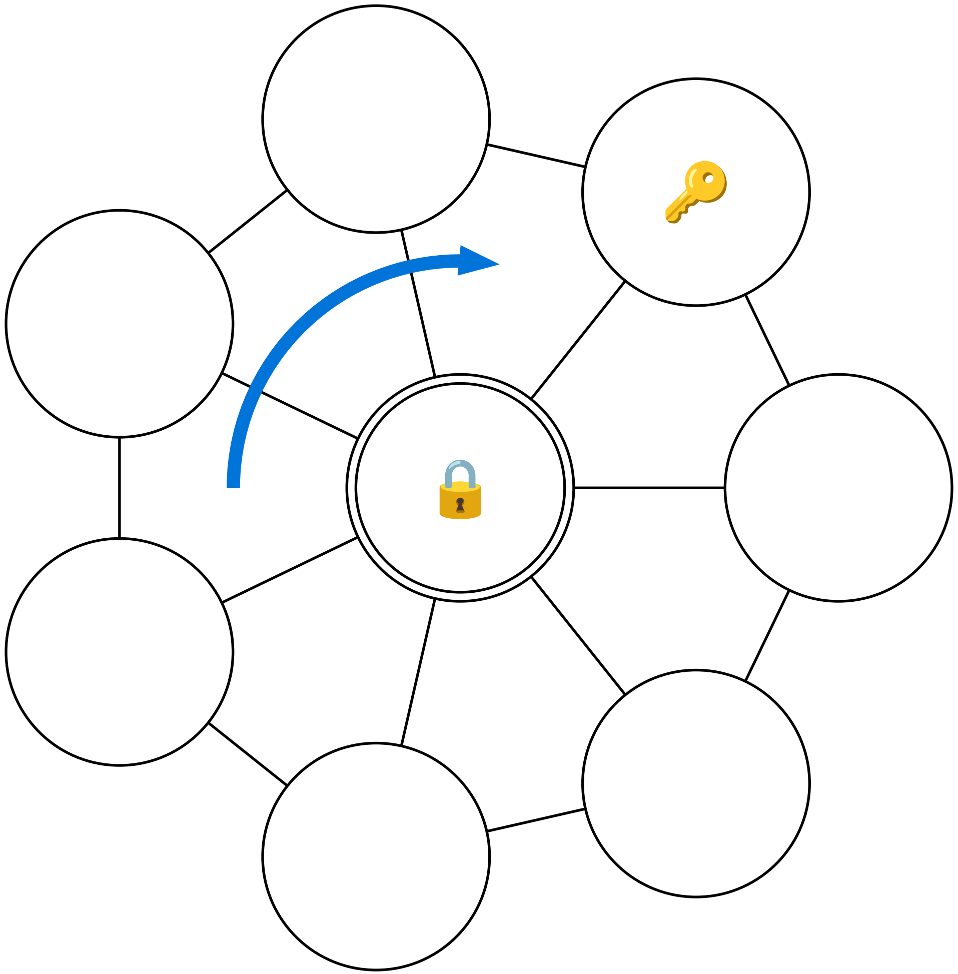
Token ring with resource



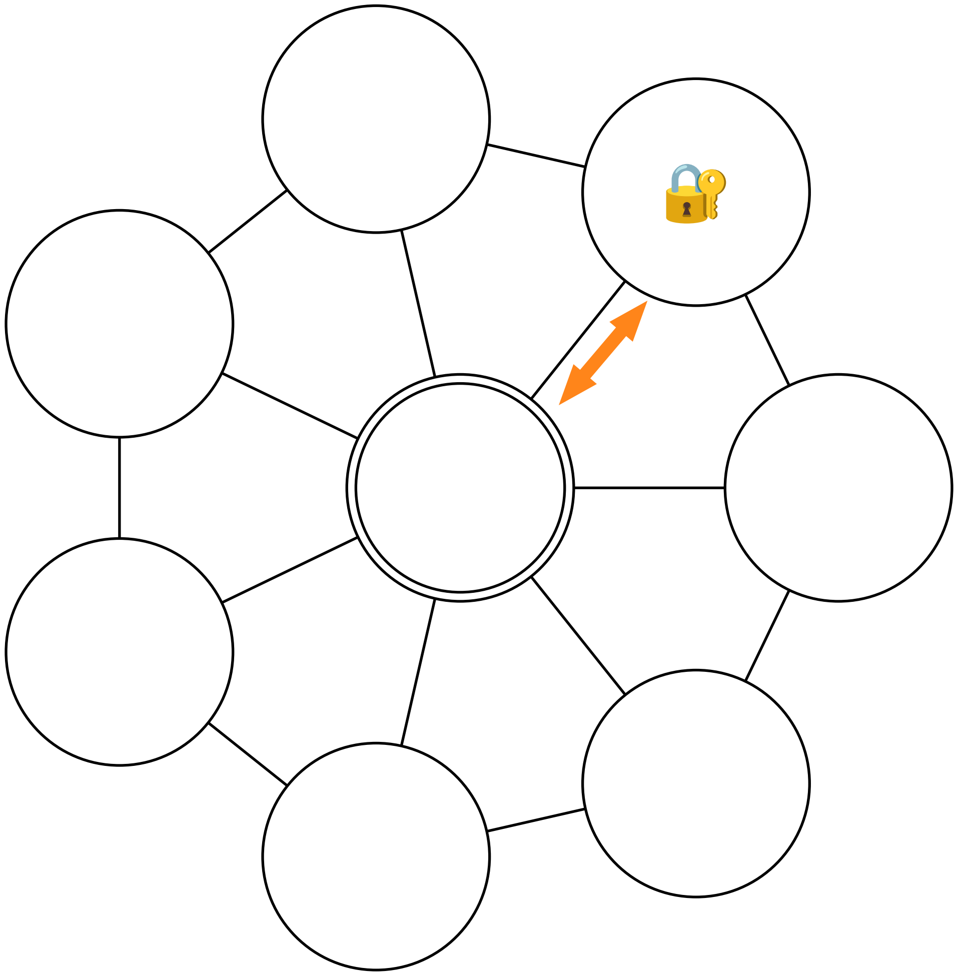
Token ring with resource



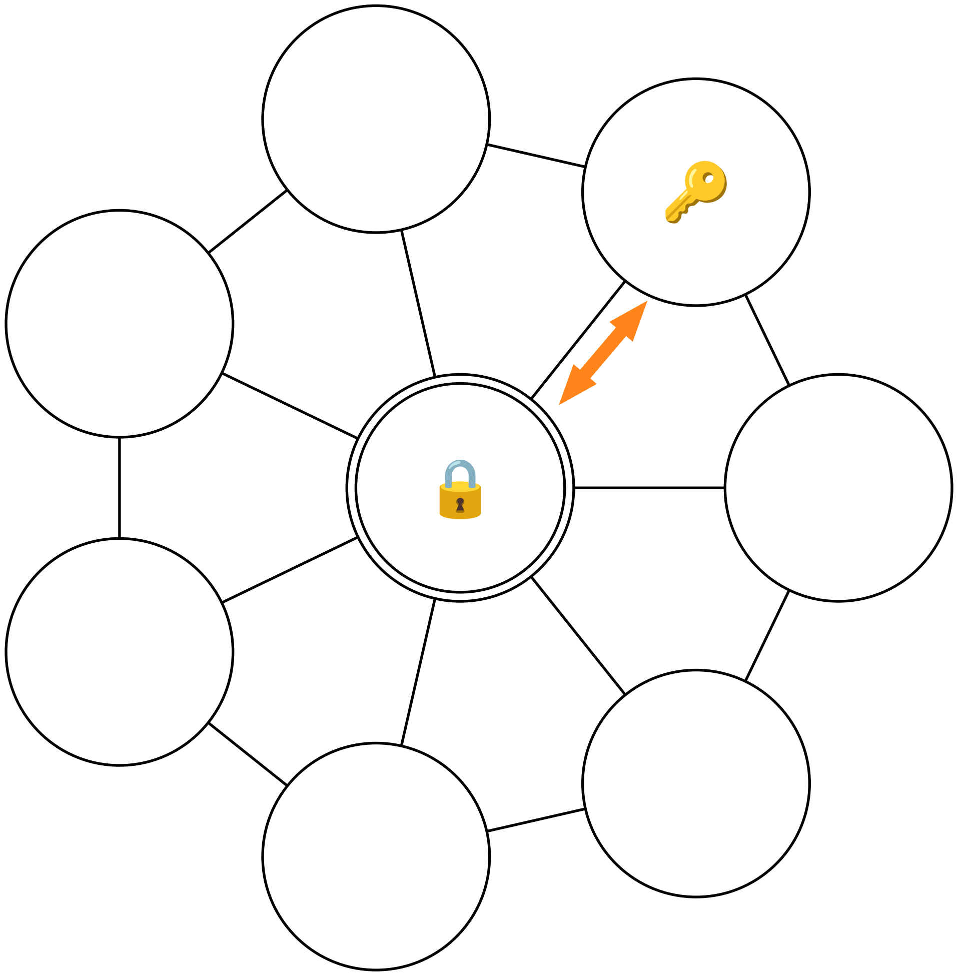
Token ring with resource



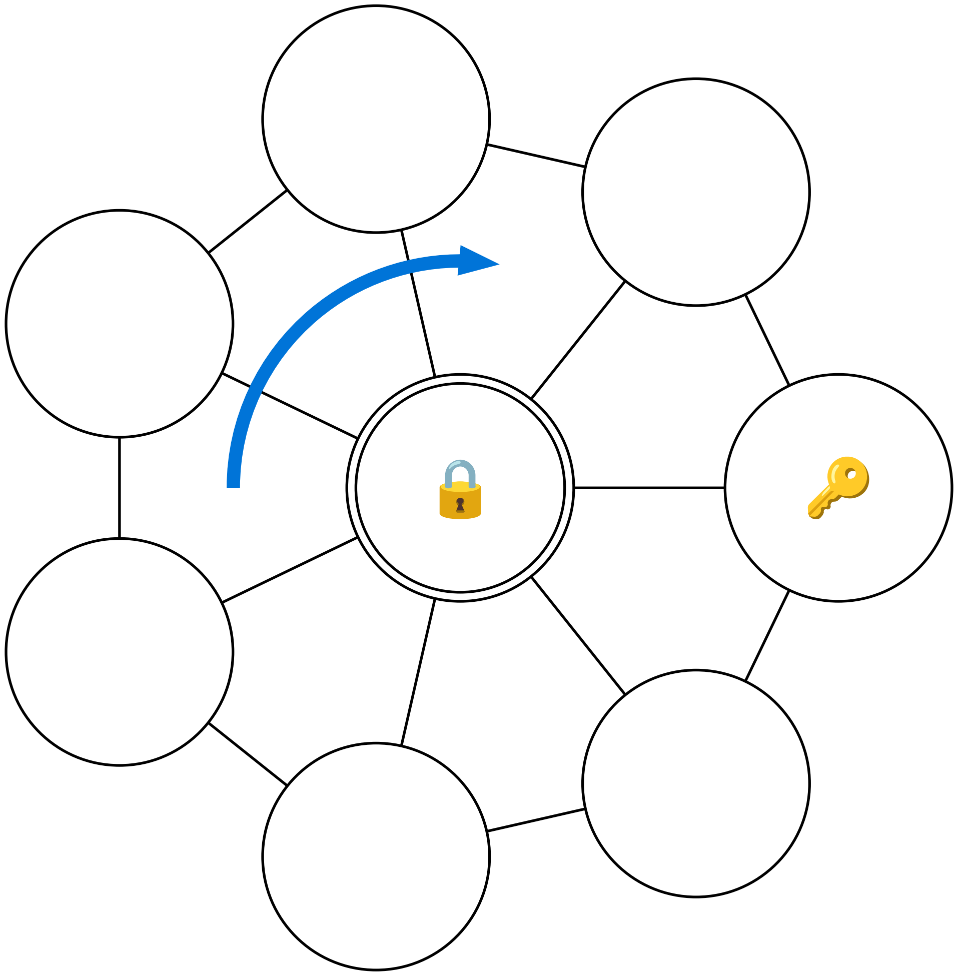
Token ring with resource



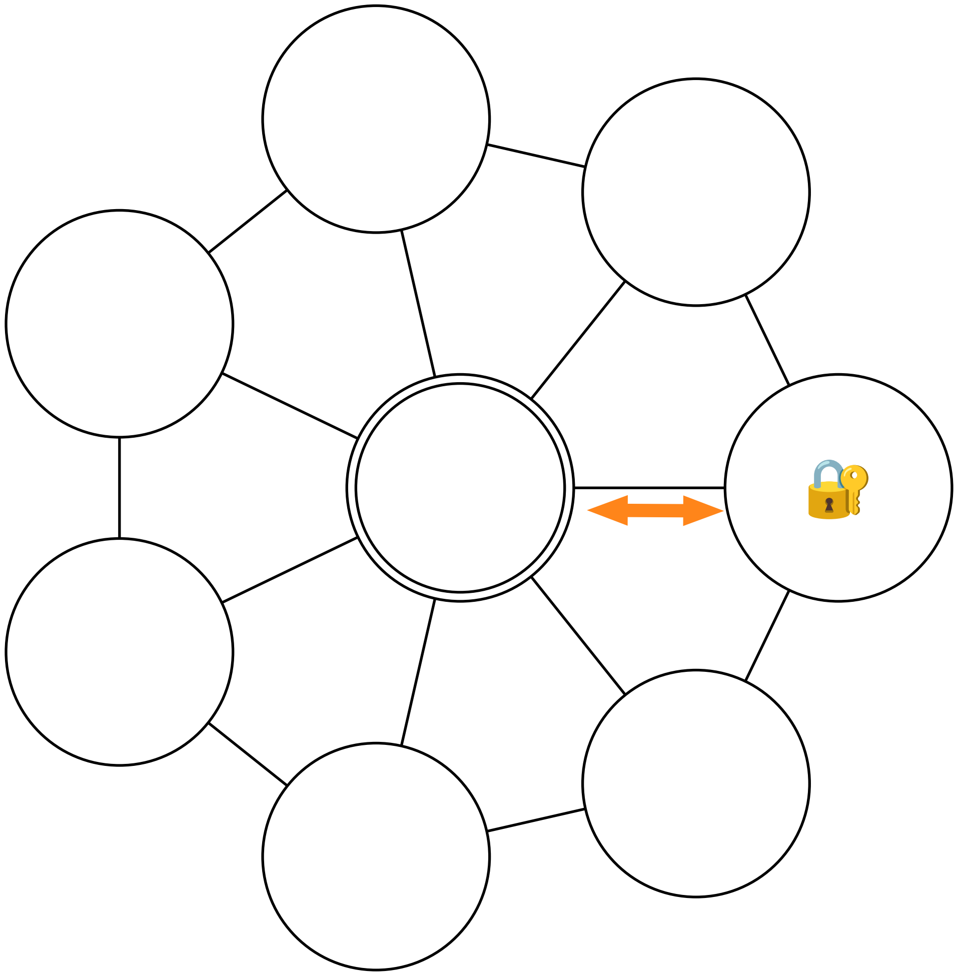
Token ring with resource



Token ring with resource

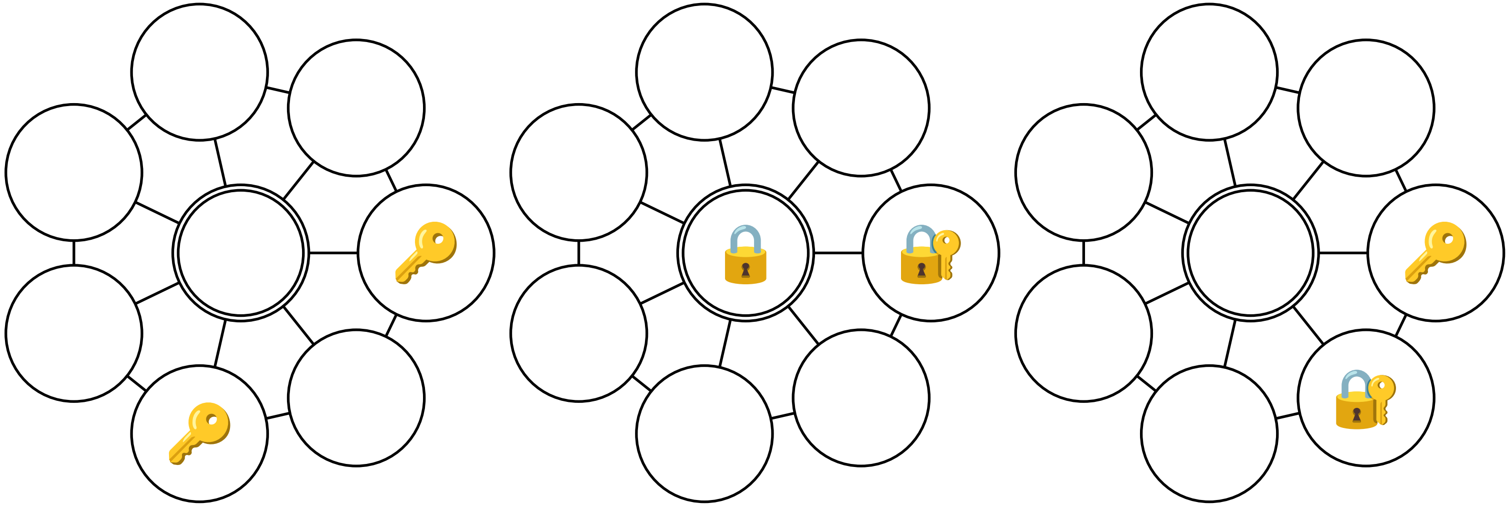


Token ring with resource

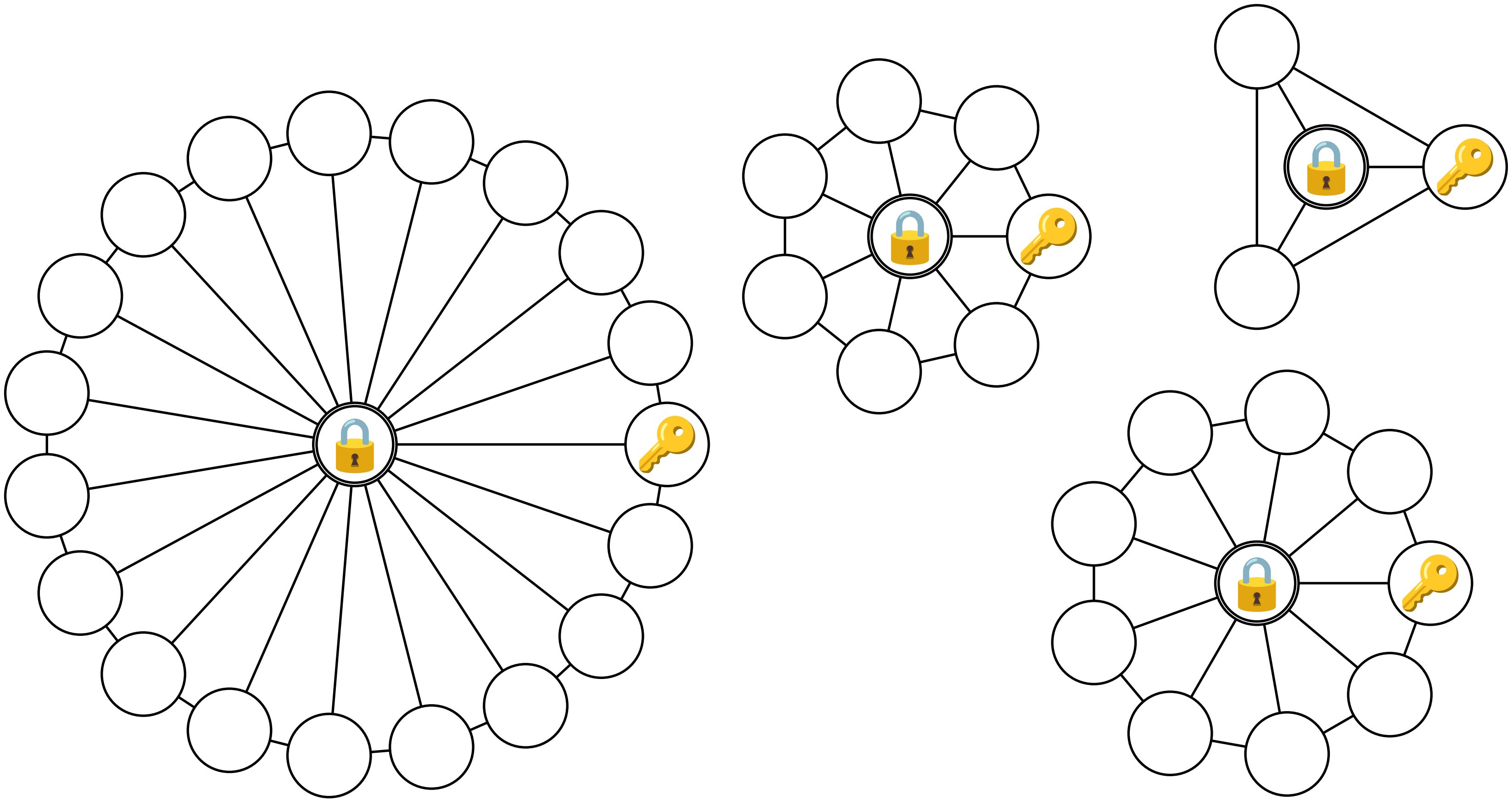


Some undesirable configurations

Safety: only one process at a time claims to own the unique resources



Parameterization



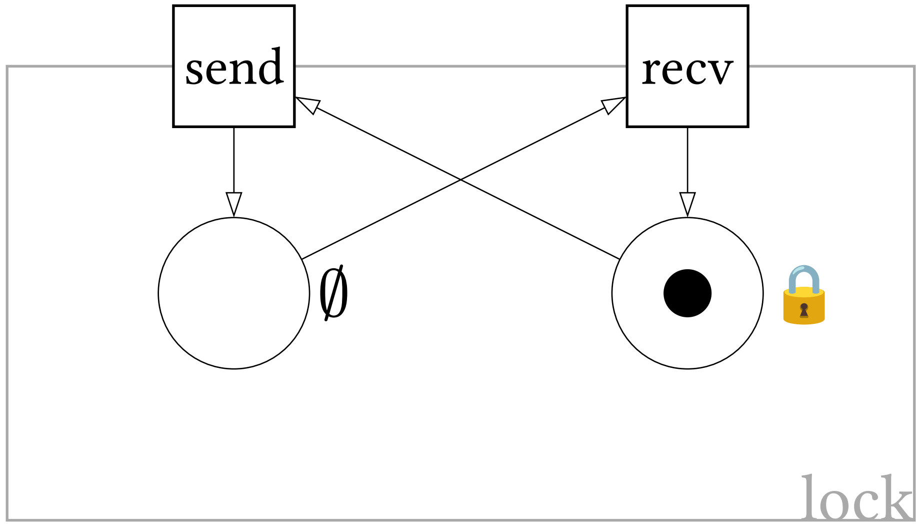
$$\forall n \geq 2$$

Requirements

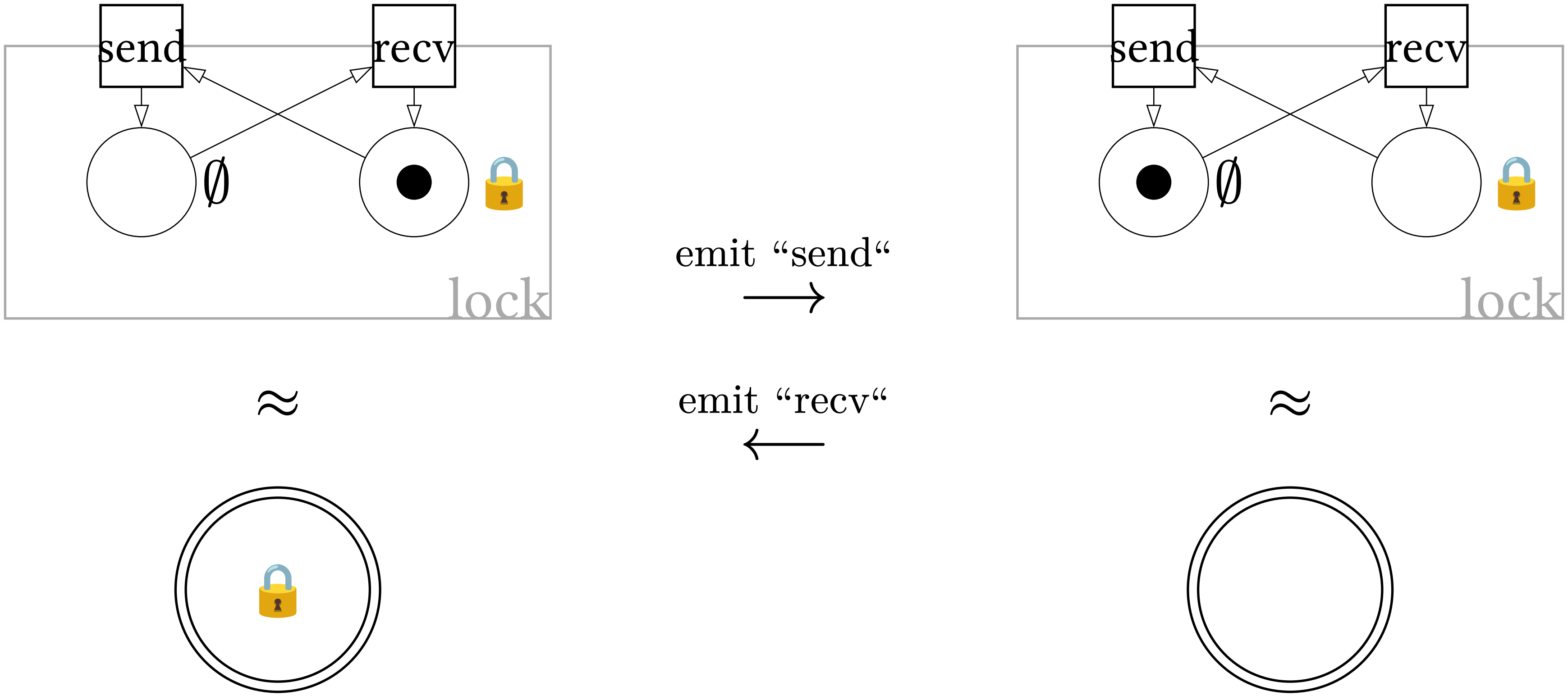
- an encoding of the **implementation** of processes
- a description of the **interactions** and **architectures** of arbitrary size
- a **specification language** for safety properties
- **approximation techniques** that work on infinite families
- a **decidable** problem to reduce to

2. Implementation

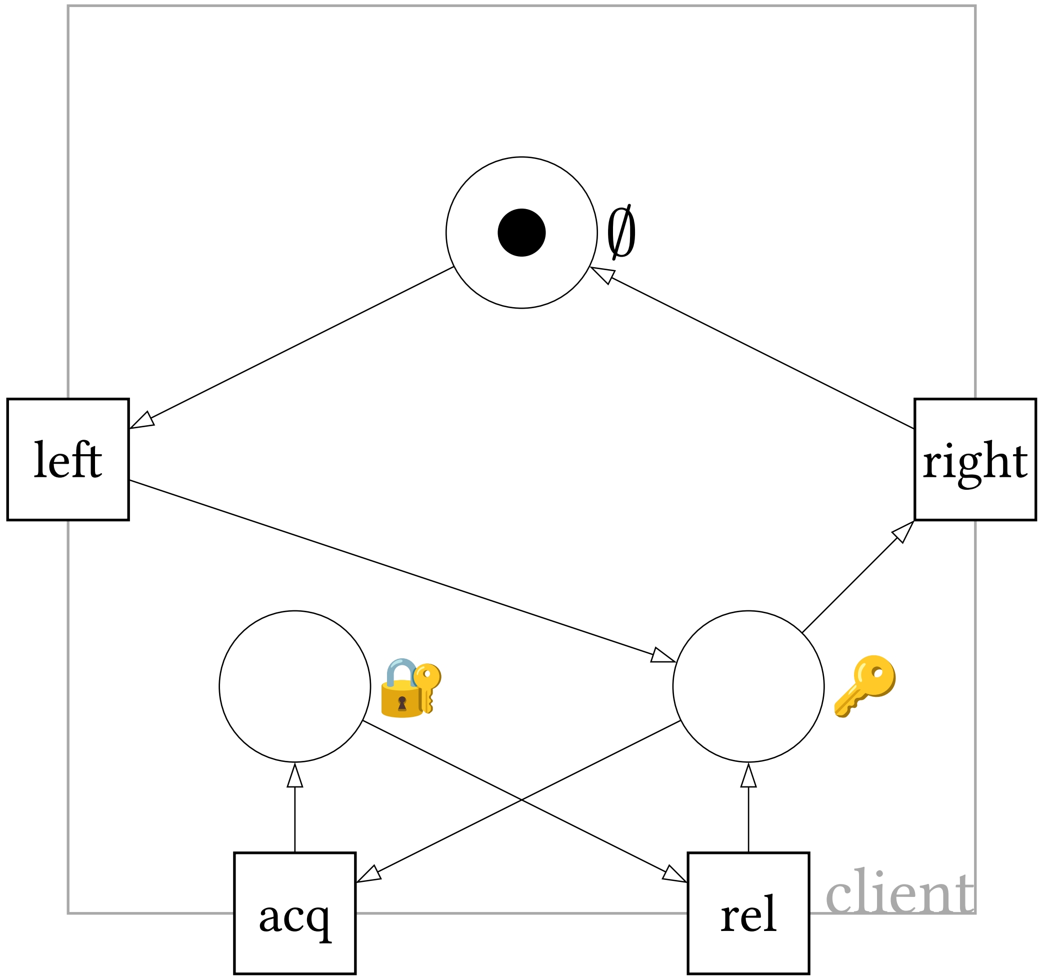
Lock



Lock

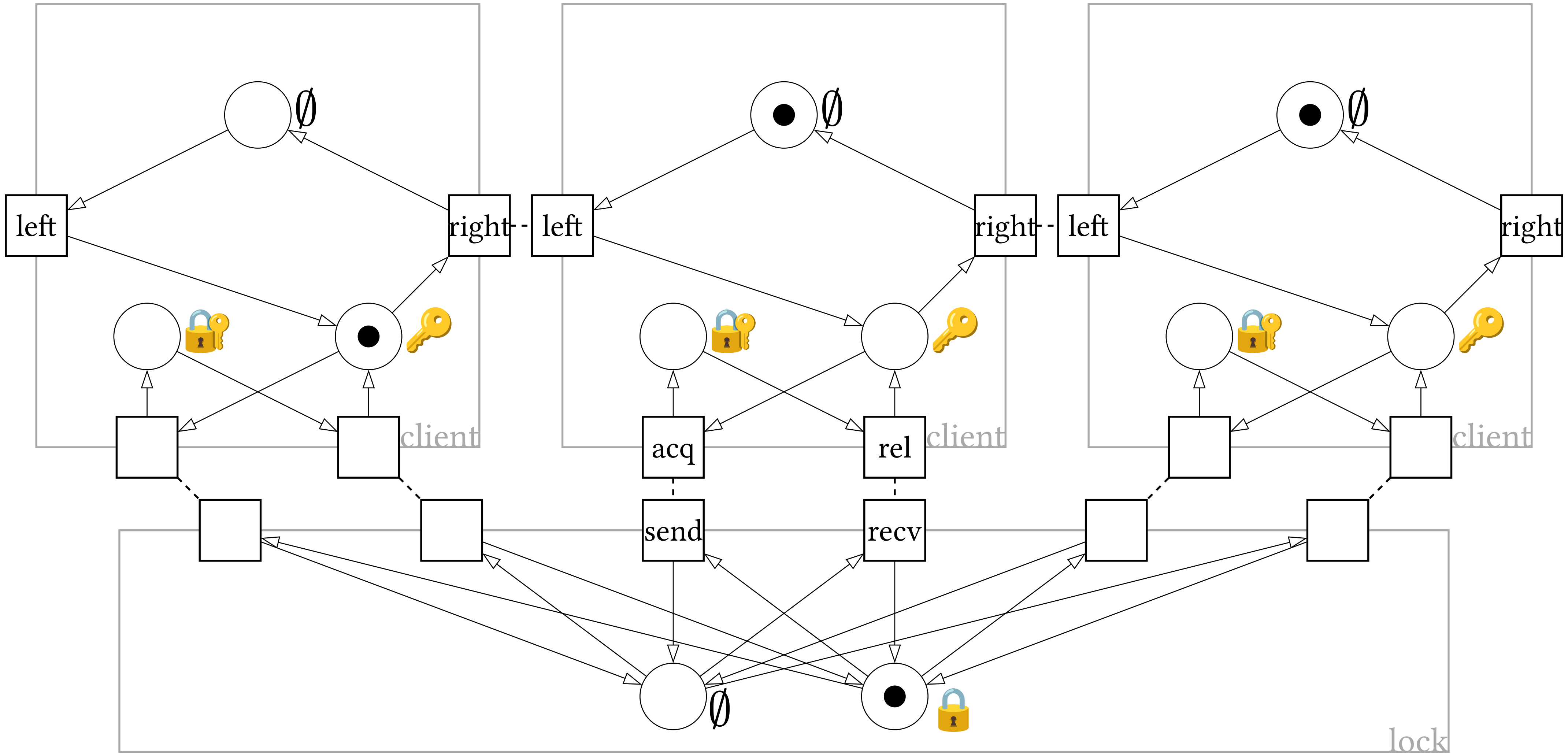


Client

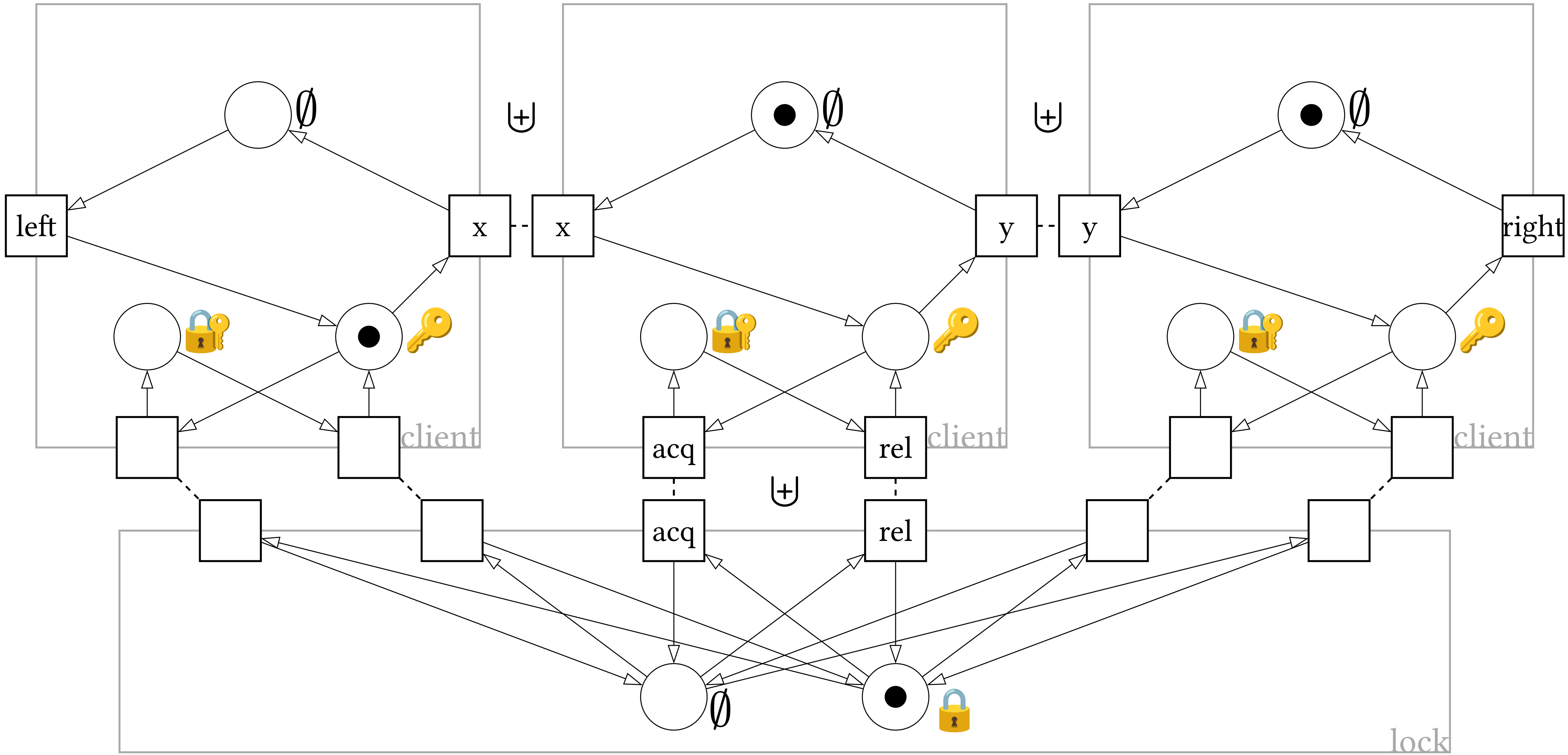


3. Interactions

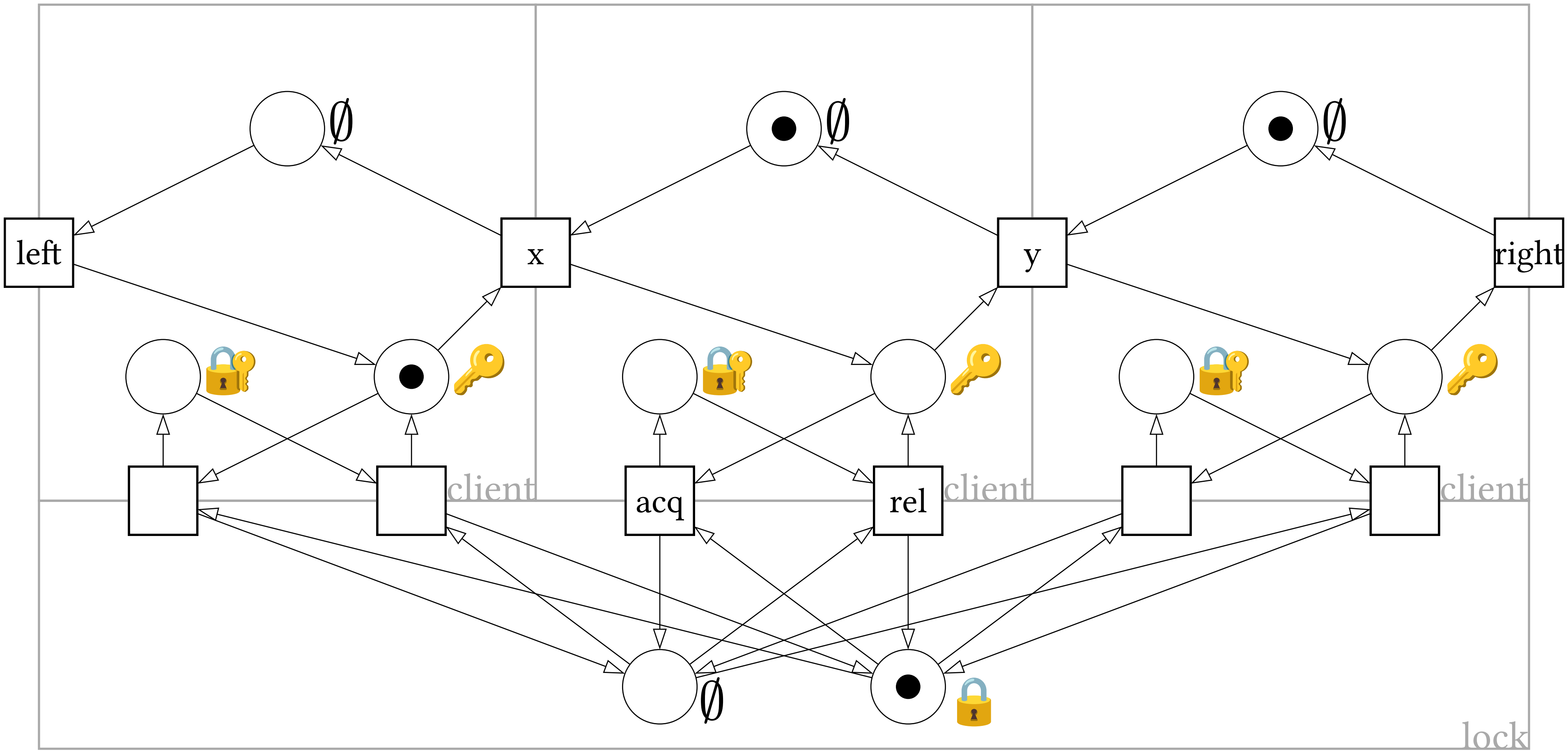
Interactions through disjoint composition



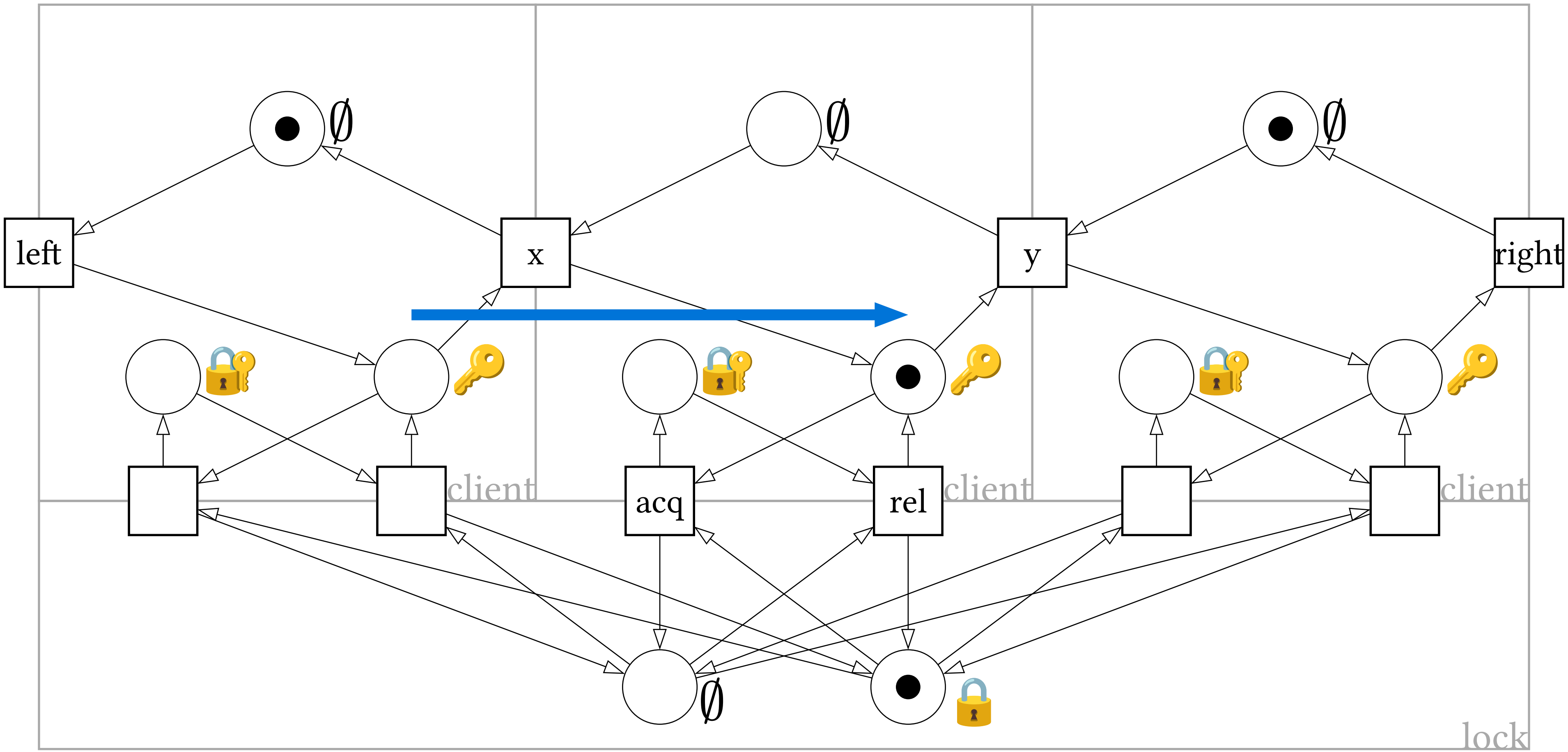
Interactions through disjoint composition



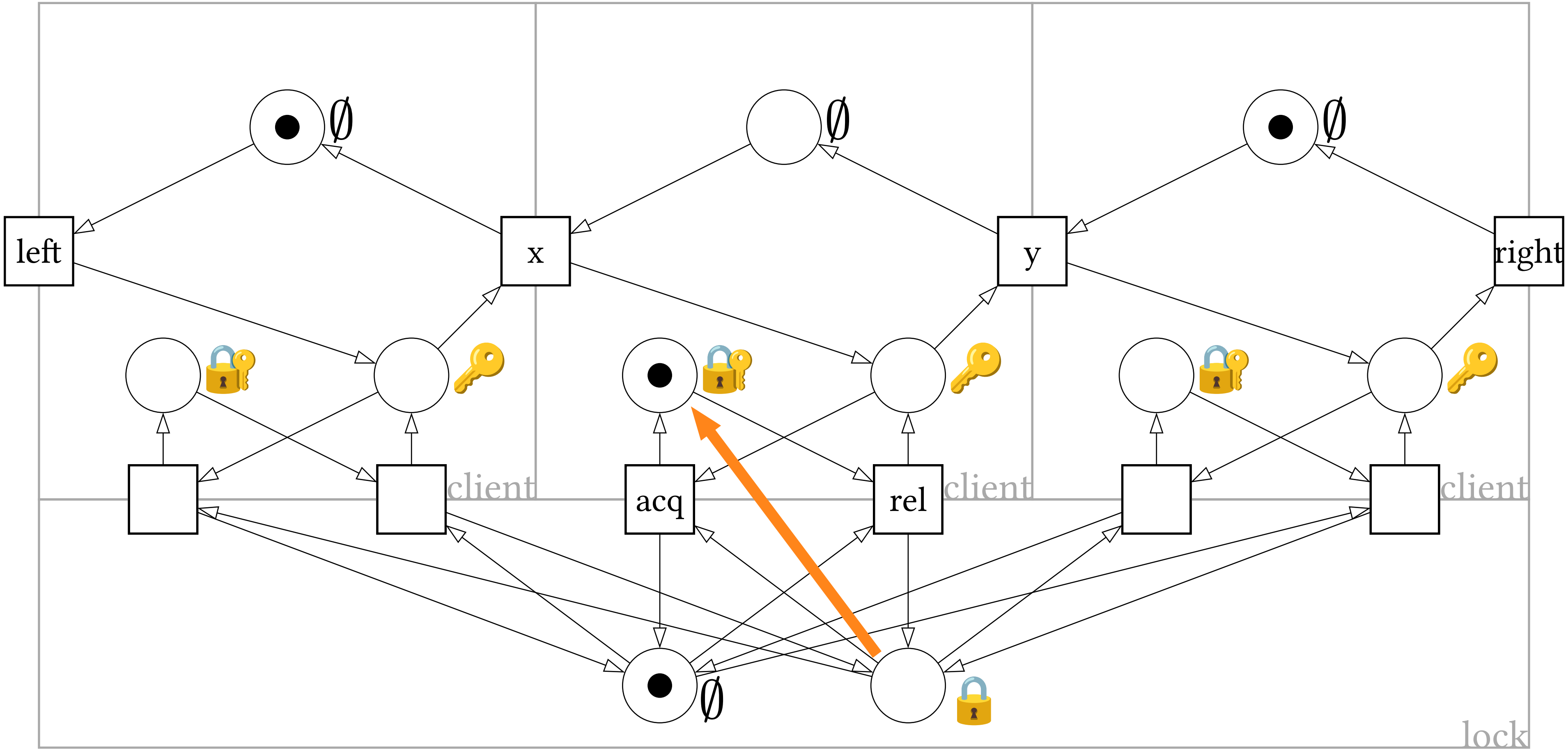
Interactions through disjoint composition



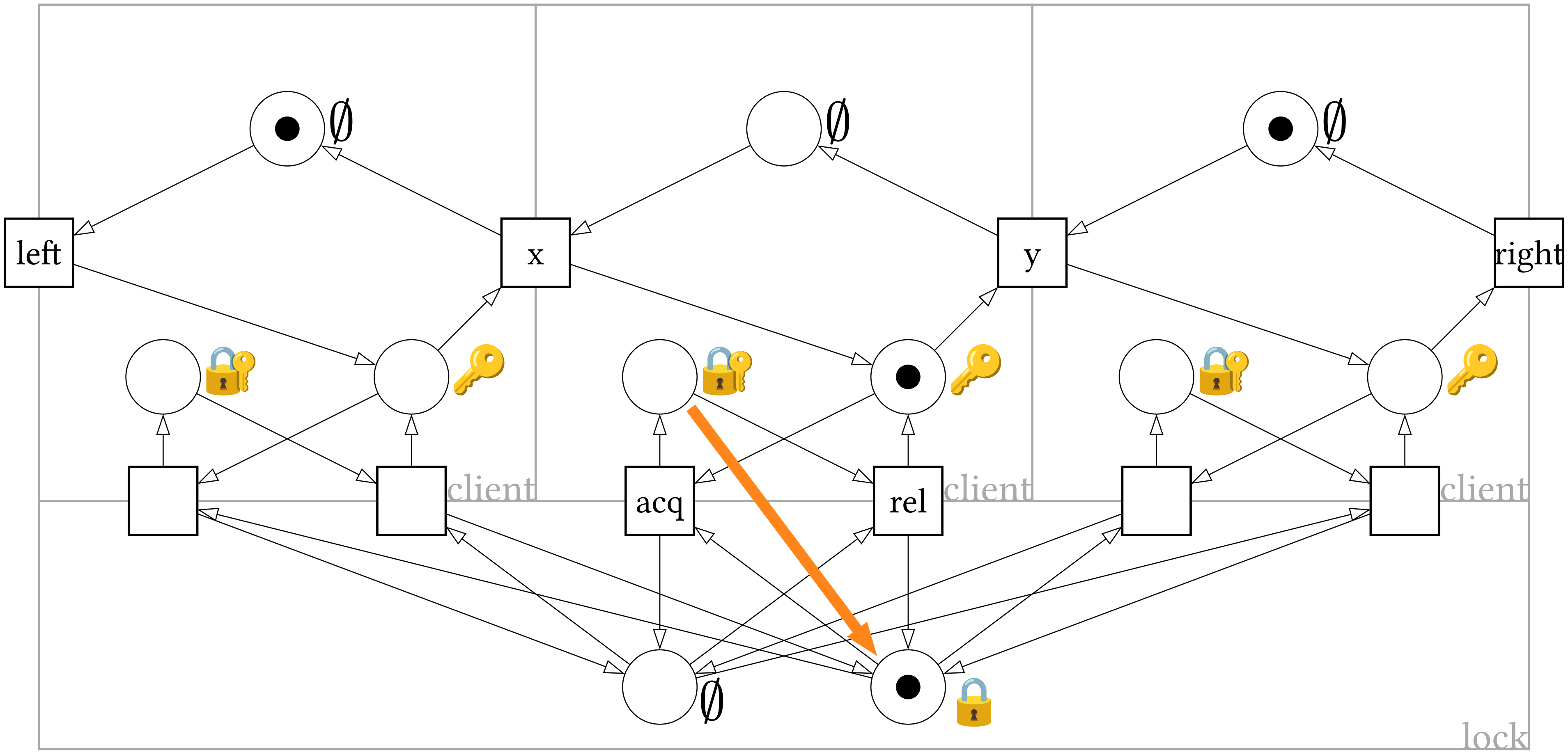
Interactions through disjoint composition



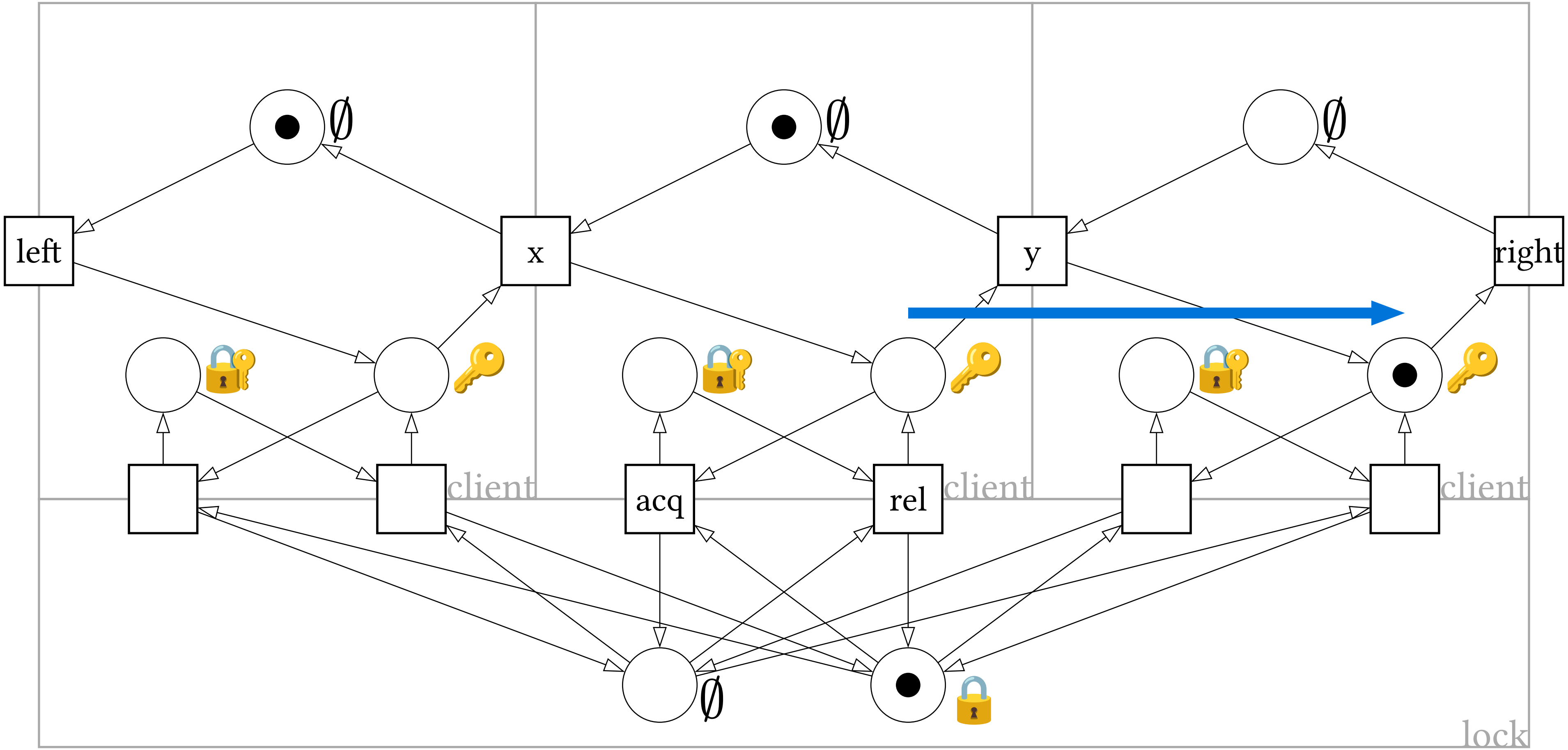
Interactions through disjoint composition



Interactions through disjoint composition

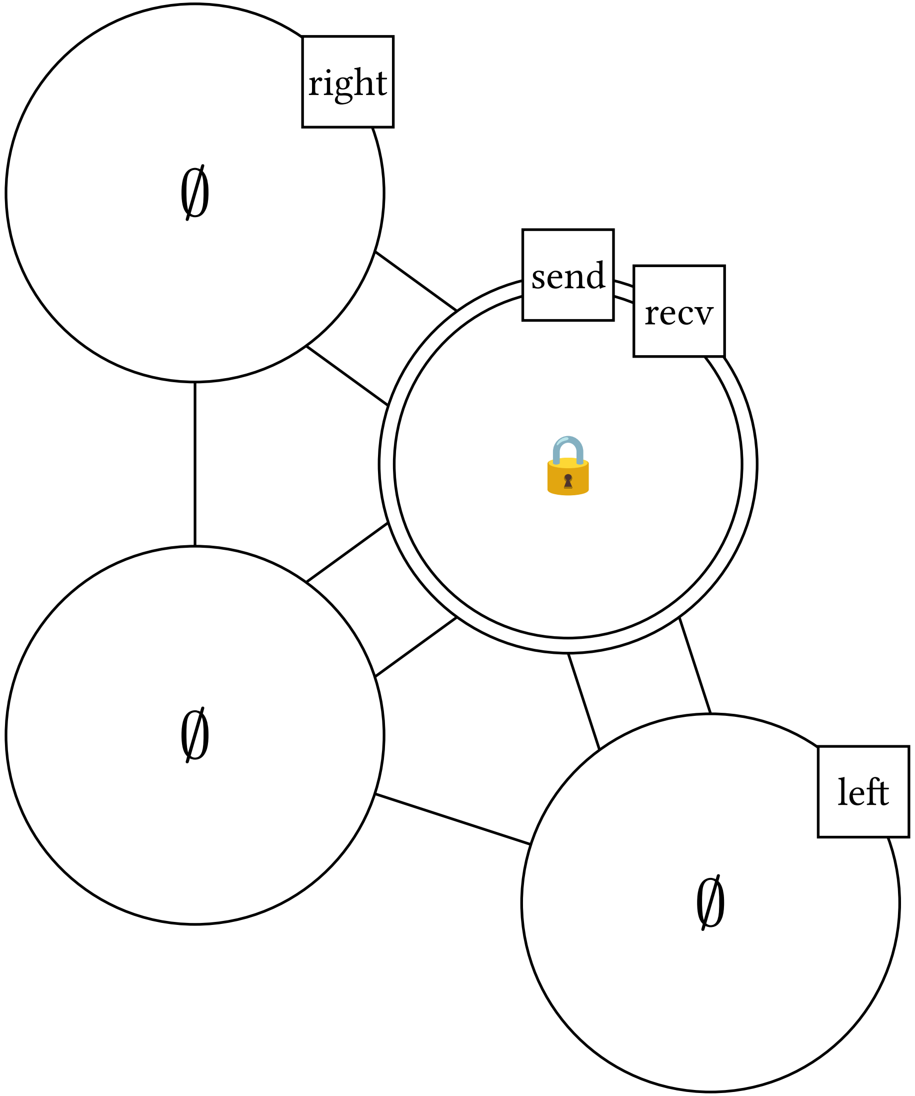


Interactions through disjoint composition

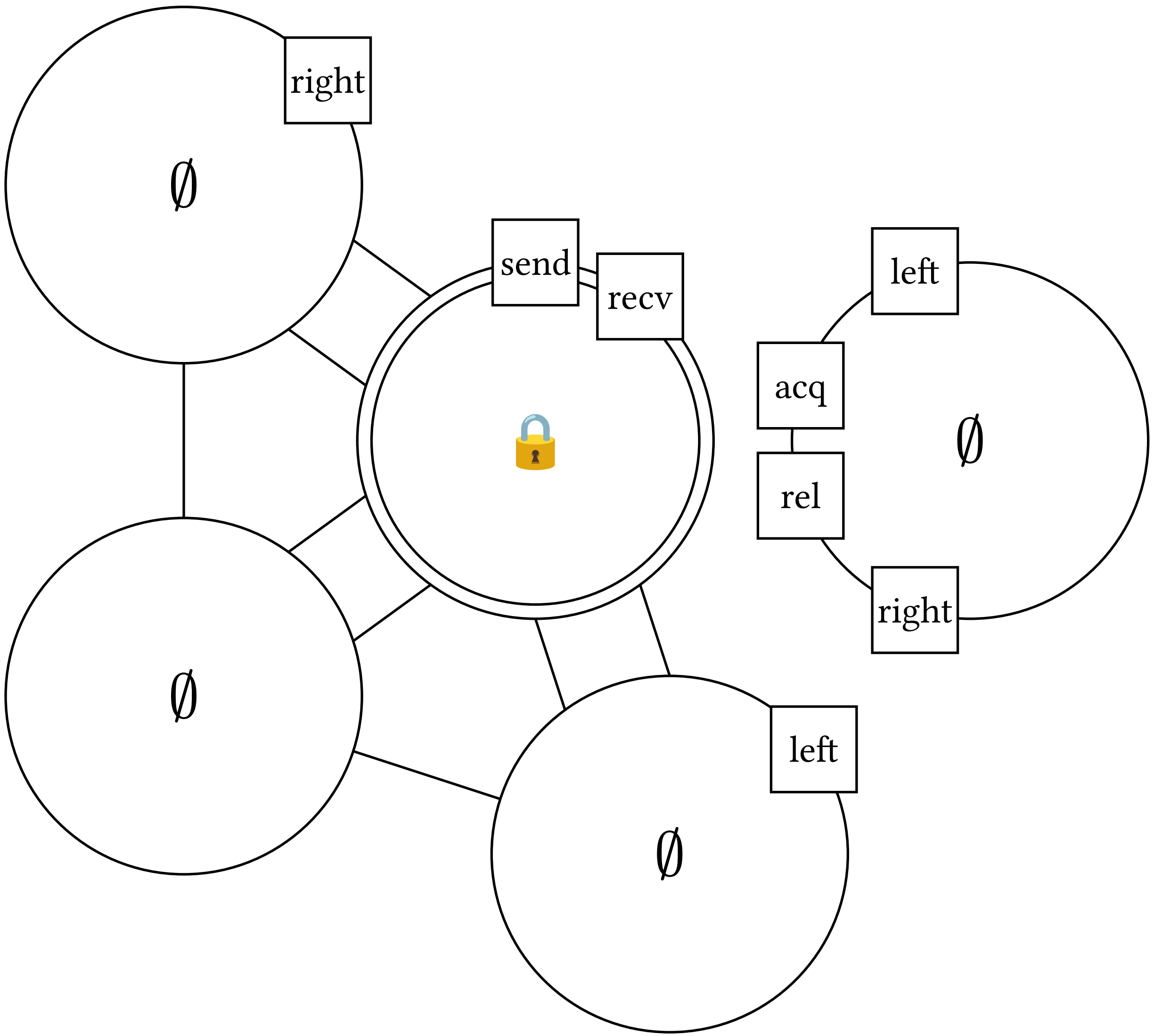


4. Architecture

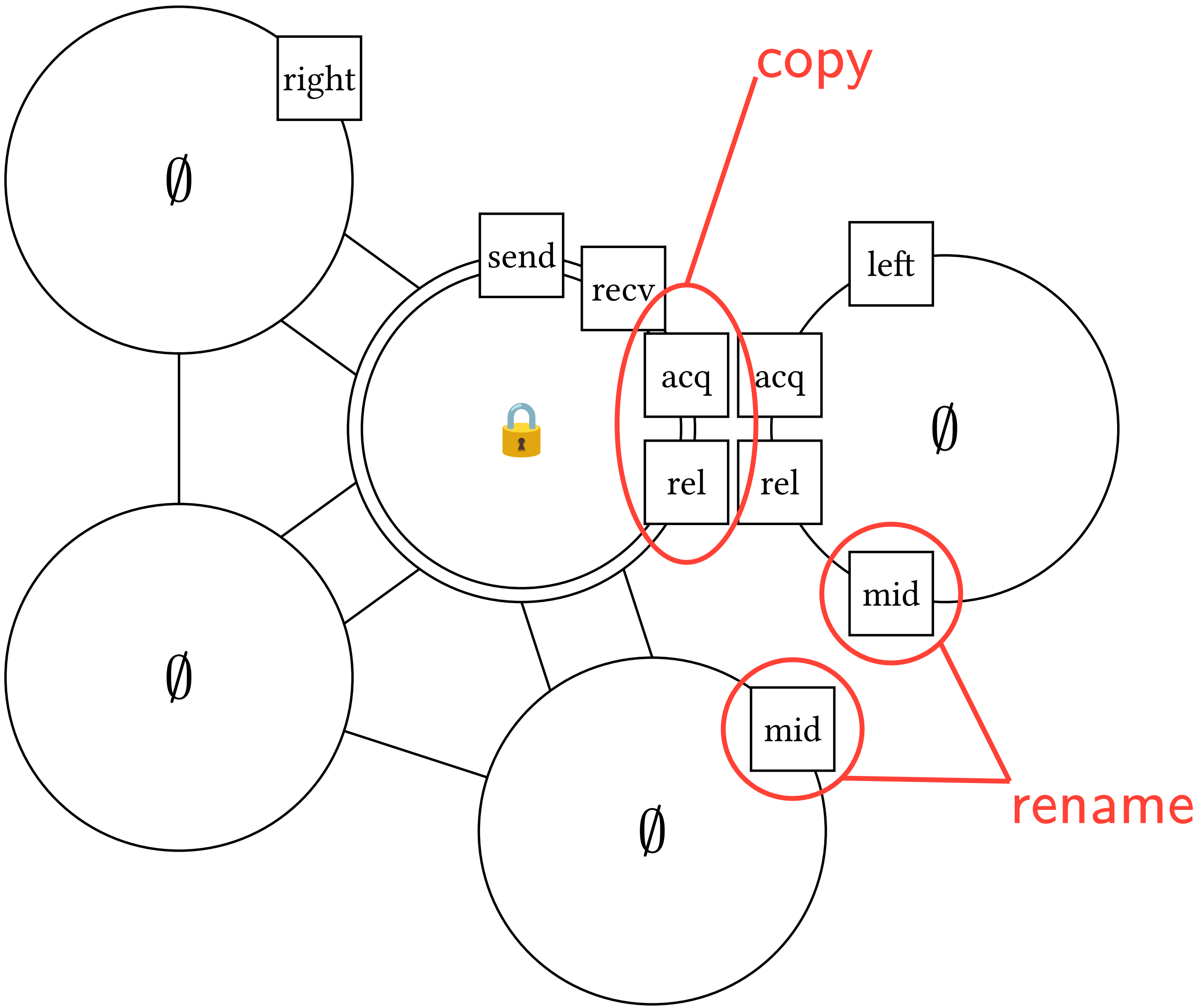
Structure: inductive step



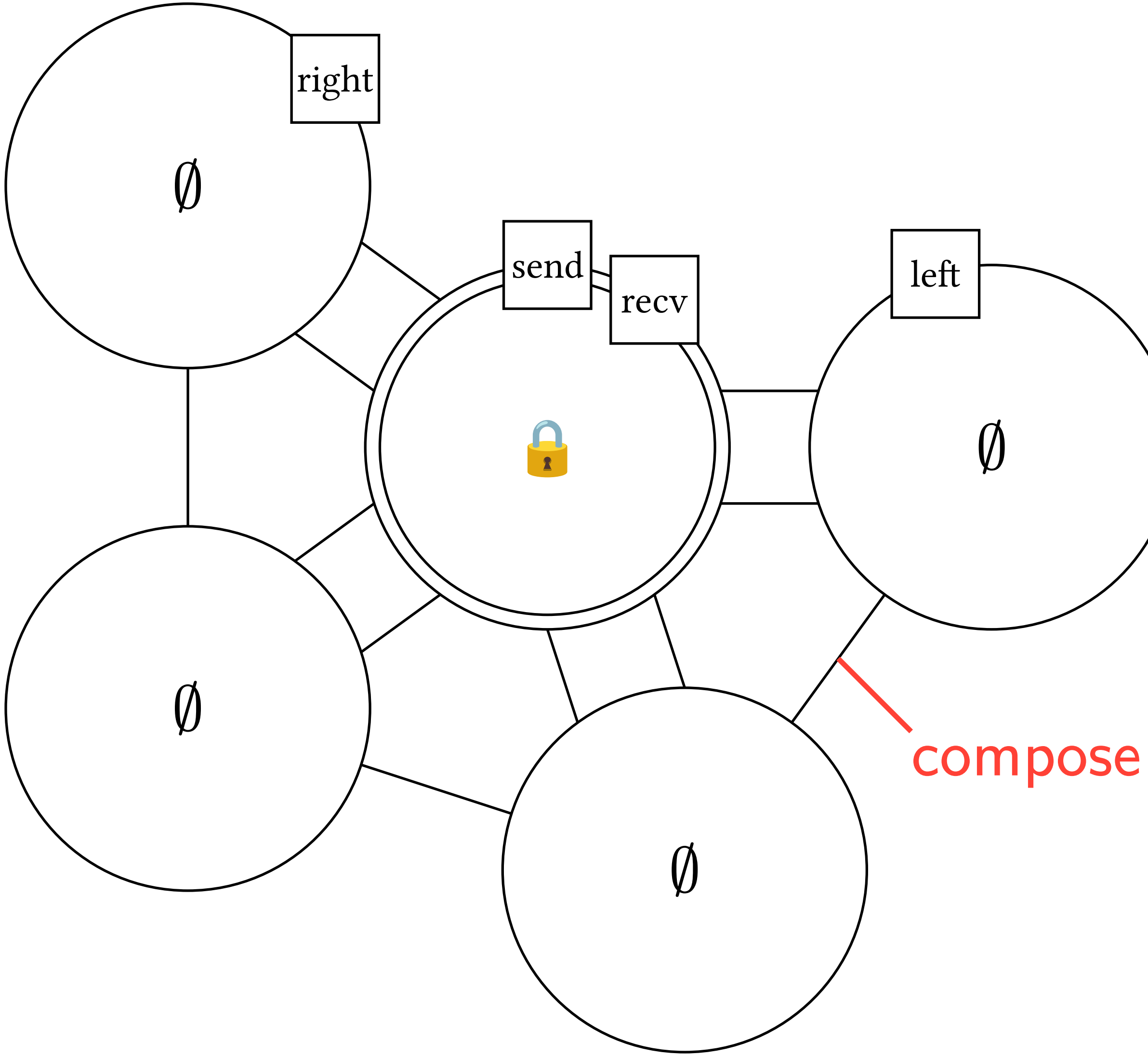
Structure: inductive step



Structure: inductive step



Structure: inductive step



Structure represented by a grammar

$$X \longrightarrow \mathsf{compose}\Big(\mathsf{rename}_{\mathrm{left}\mapsto\mathrm{mid}}\big(\mathsf{copy}_{\mathrm{send}\rightsquigarrow\mathrm{acq},\mathrm{recv}\rightsquigarrow\mathrm{rel}}(X)\big),\ \mathsf{rename}_{\mathrm{right}\mapsto\mathrm{mid}}(\mathrm{proc})\Big)$$

Structure represented by a grammar

$$\mathrm{Sys} \longrightarrow \mathsf{compose}\big(X, \mathsf{rename}_{\mathrm{left}\mapsto\mathrm{right},\ \mathrm{right}\mapsto\mathrm{left}}(\mathrm{proc}')\big)$$

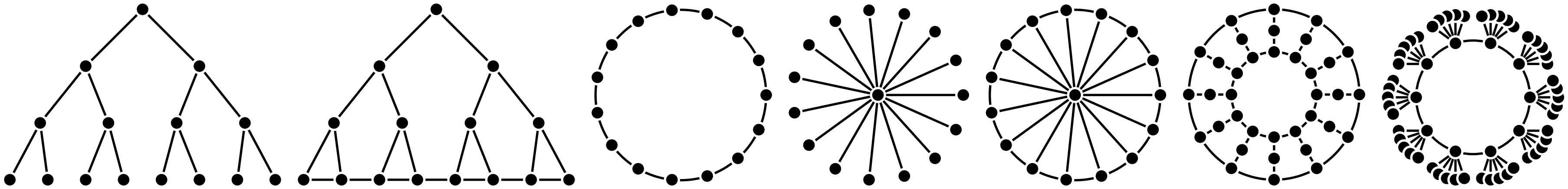
$$\begin{aligned} X \longrightarrow \mathsf{compose}\big(& \\ & \mathsf{rename}_{\mathrm{left}\mapsto\mathrm{mid}}\big(\mathsf{copy}_{\mathrm{send}\rightsquigarrow\mathrm{acq},\mathrm{recv}\rightsquigarrow\mathrm{rel}}(X)\big), \\ & \mathsf{rename}_{\mathrm{right}\mapsto\mathrm{mid}}(\mathrm{proc}) \\ \big)& \end{aligned}$$

$$X \longrightarrow \mathsf{compose}\big(\mathsf{copy}_{\mathrm{send}\rightsquigarrow\mathrm{acq},\mathrm{recv}\rightsquigarrow\mathrm{rel}}(\mathrm{lock}), \mathrm{proc}\big)$$

Representable architectures

These manipulations are encoded in a form similar to CFG for graphs.

- language of a grammar is an (infinite) set of Petri nets
- many families of networks of bounded tree-width are representable
- missing: ~~grids~~, ~~cliques~~



1 3 2 1 3 4 2

5. Safety specification

Safety properties

#(🔑): number of tokens on 🔑
$\sim$ number of clients who claim to own the key

If any size of the system has a reachable configuration with
#(🔑) + #(🔐) > 1, there is a bug in the specification.

Proving safety $\approx$ solving a reachability problem in an infinite family of Petri nets

Other undesirable configurations: #(🔒) + #(🔐) > 1

Expressible properties

- **mutual exclusion**
  “at most k processes can enter a critical section simultaneously”
- **uniqueness**
  “the entire system contains at most k instances of a resource”
- **unreachability**
  “no process can reach a bad state”

Examples: leader election, locks and semaphores, dining philosophers, …

Missing: ~~liveness~~, ~~deadlock freedom~~

6. Verification

Using an abstraction

$$\binom{\text{Implementation}}{\text{Petri nets}} + \binom{\text{Architecture}}{\text{Grammar } \Gamma} + \binom{\text{Specification}}{\text{Formula } \varphi}$$

$$\rightsquigarrow \text{Do all systems generated by } \Gamma \text{ avoid bad configurations } \varphi\text{?}$$

$$(\text{written } \Gamma \nvDash \varphi)$$

Undecidable !

Using an abstraction

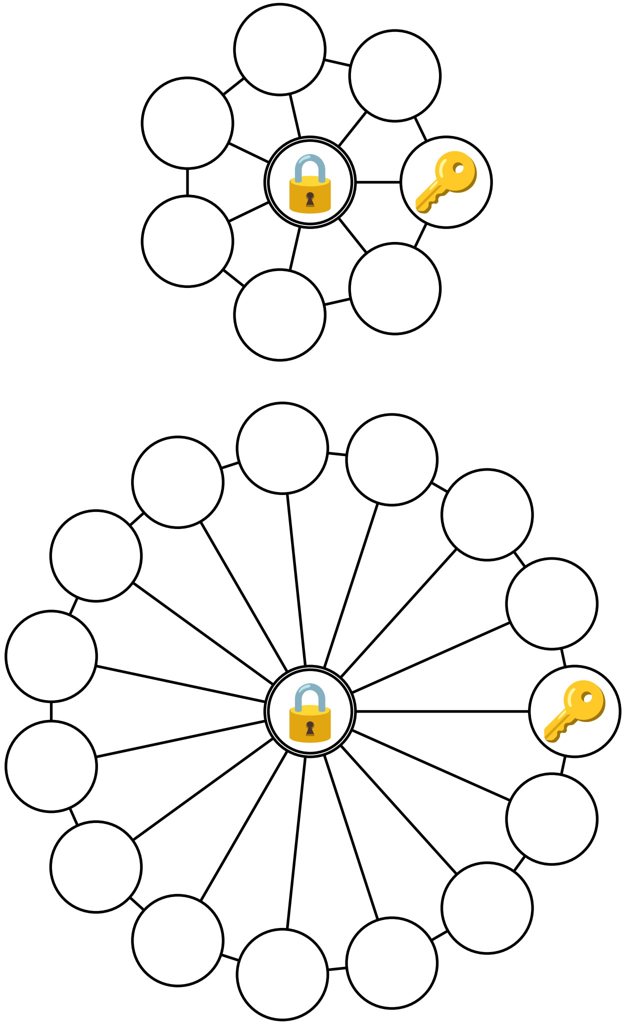
$$\begin{pmatrix}\text{Implementation}\\ \text{Petri nets}\end{pmatrix} + \begin{pmatrix}\text{Abstract architecture}\\ \text{Grammar } \alpha(\Gamma)\end{pmatrix} + \begin{pmatrix}\text{Specification}\\ \text{Formula } \varphi\end{pmatrix}$$

$\rightsquigarrow$ Do all systems generated by $\alpha(\Gamma)$ avoid bad configurations φ?

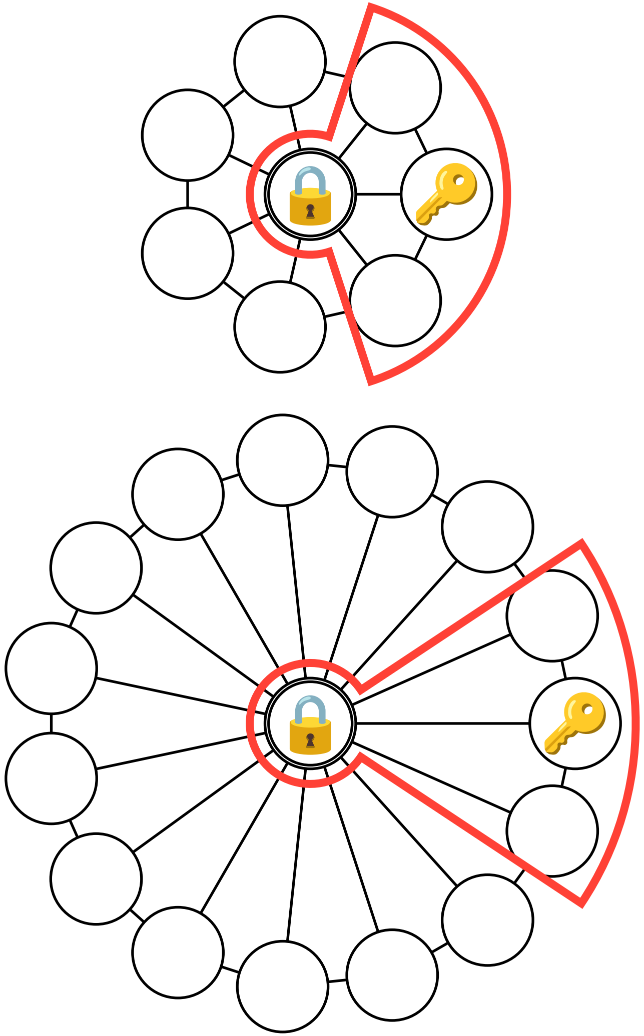
(written $\alpha(\Gamma) \nvDash \varphi$)

- $\alpha(\Gamma)$ finite $\to$ coverability solvable on $\alpha(\Gamma)$
- α should preserve violations of safety properties

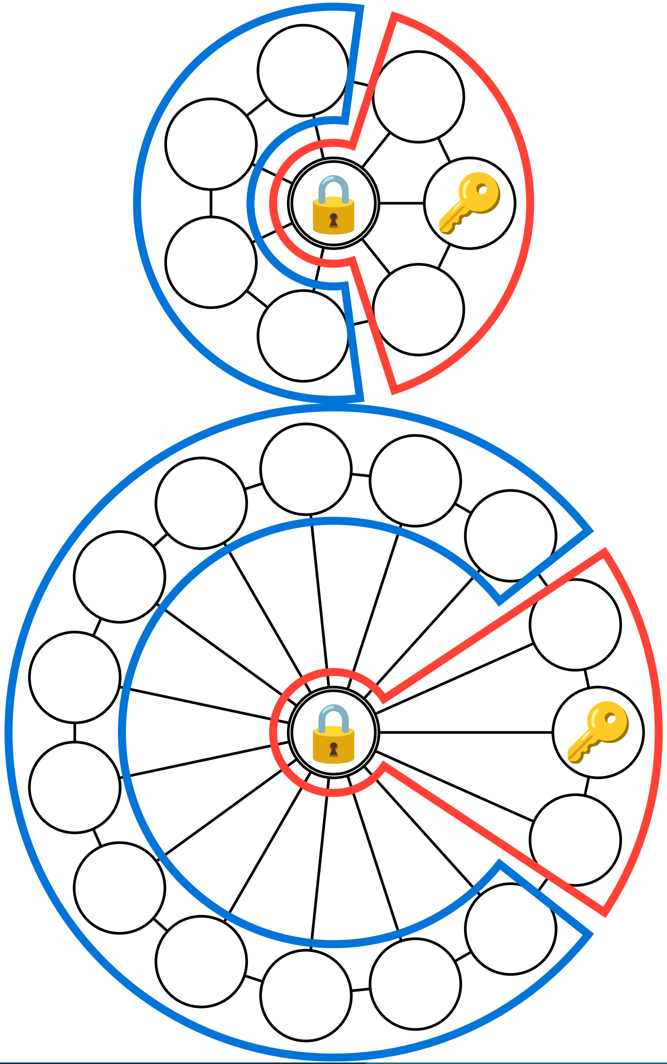
Counting abstraction (folding)



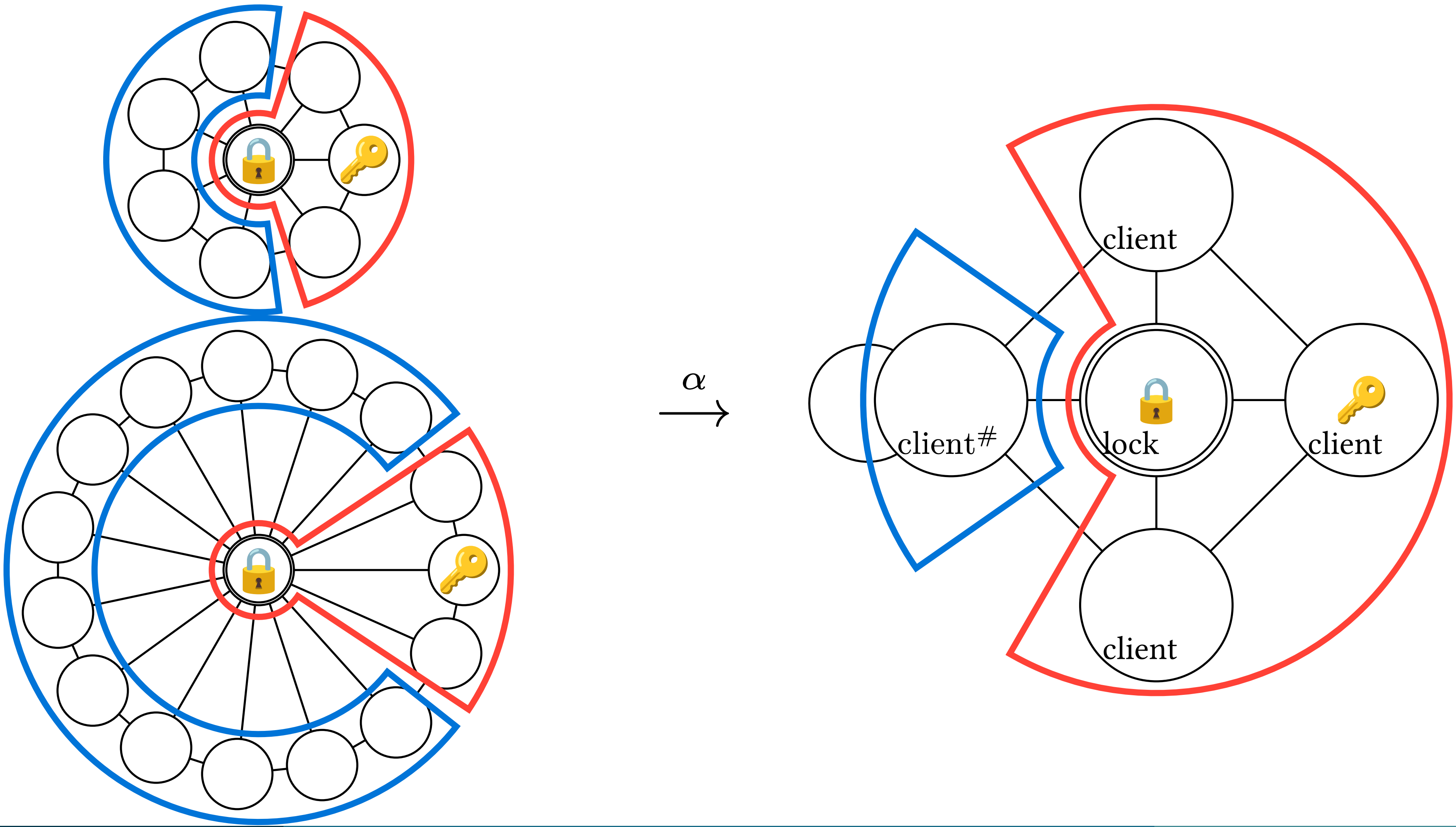
Counting abstraction (folding)



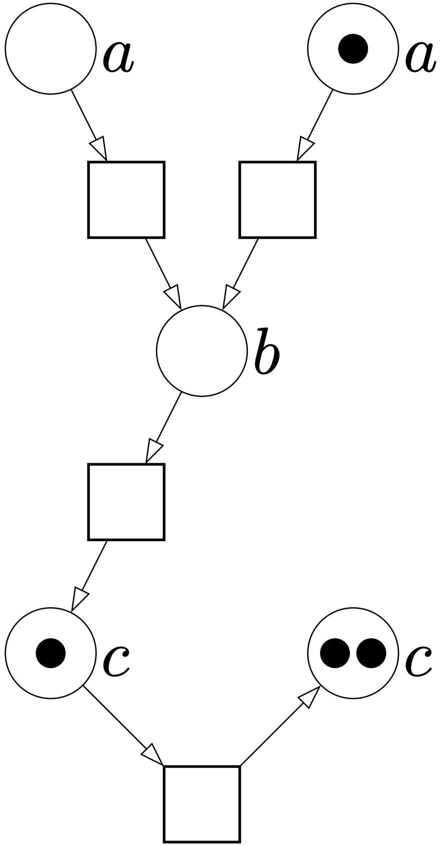
Counting abstraction (folding)



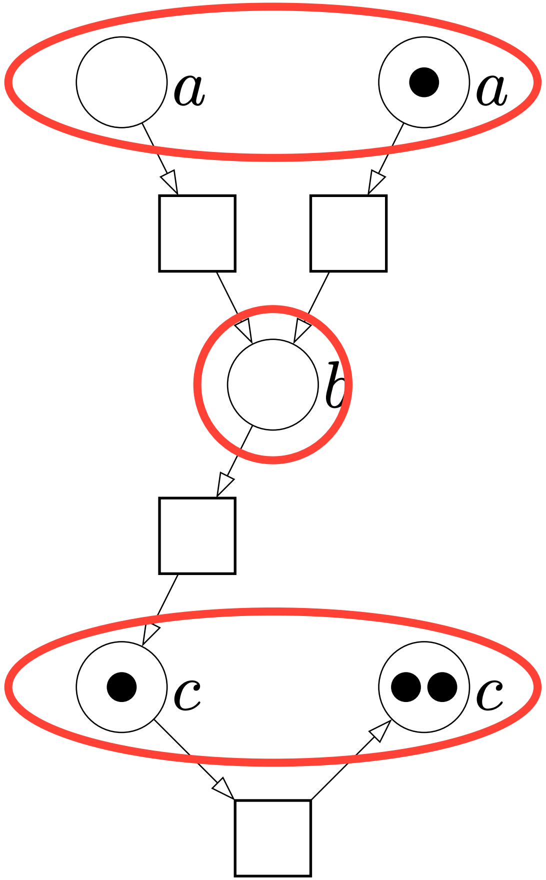
Counting abstraction (folding)



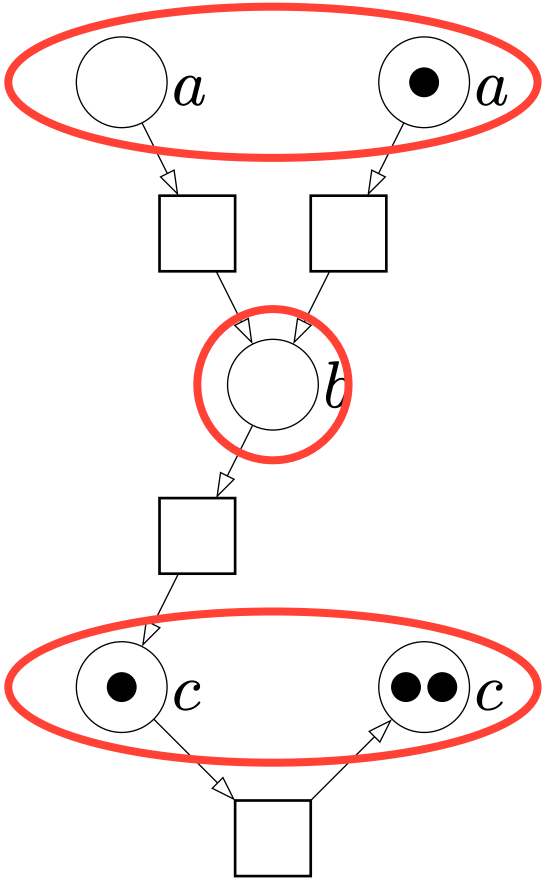
Implementing α



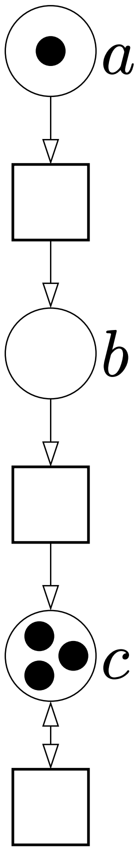
Implementing α



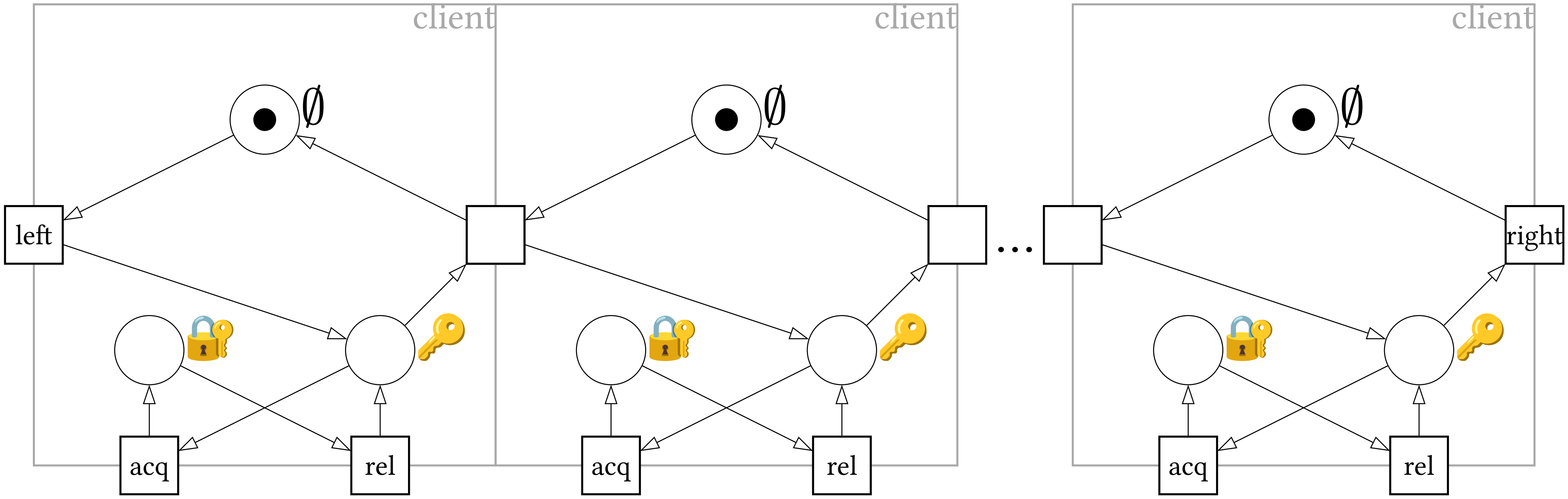
Implementing α



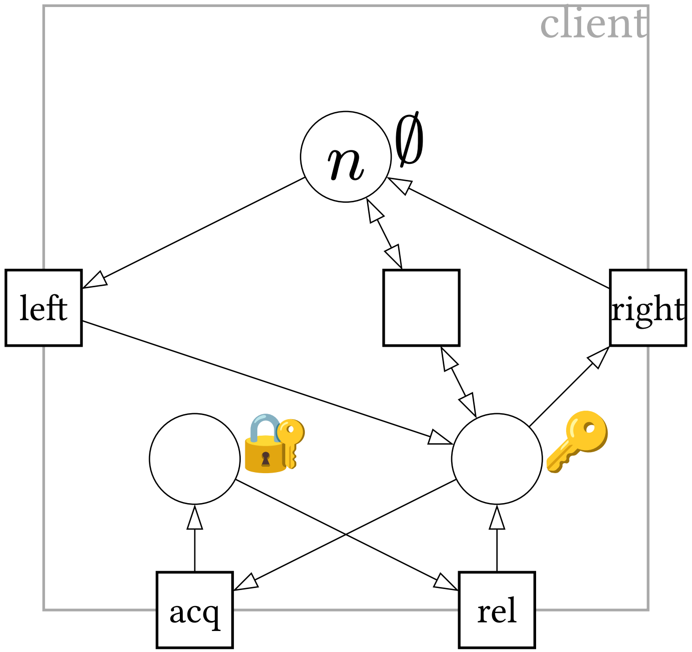
$\xrightarrow{\alpha}$



What does $\text{client}^{\#}$ look like ?

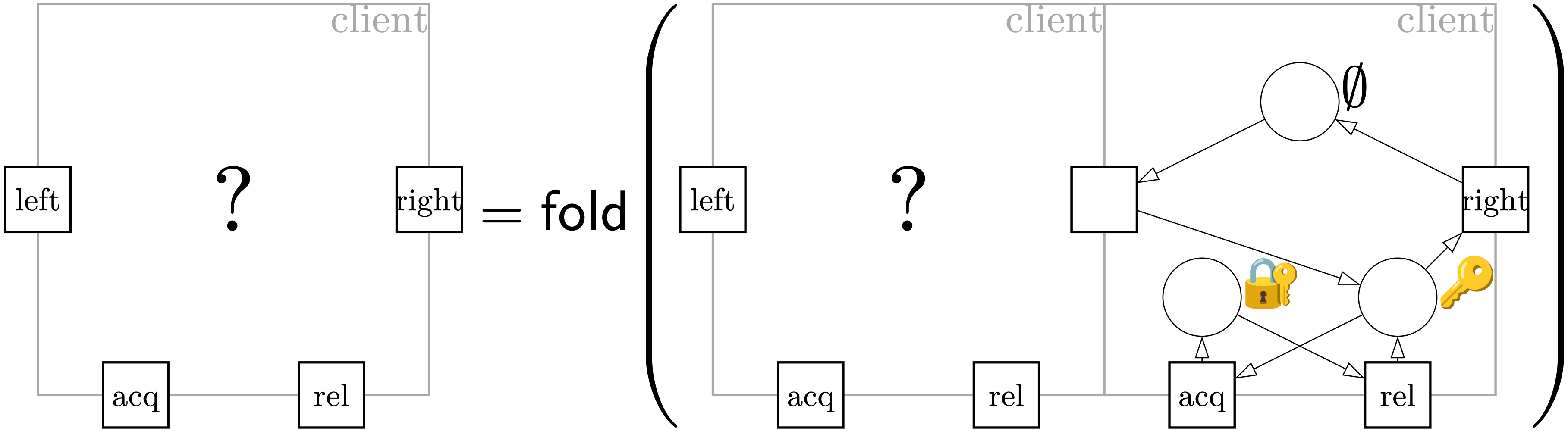


What does client$^{\#}$ look like ?



Practical computation of client$^{\#}$

Find a least fixed point of the equation



In practice: bottoms-up application of the rules of the grammar (finite domain).

Initial marking

$$\mathrm{Sys} \longrightarrow \mathsf{compose}\Big(X, \mathsf{rename}_{\mathrm{left}\mapsto\mathrm{right},\ \mathrm{right}\mapsto\mathrm{left}}(\mathrm{proc}')\Big)$$

$$X \longrightarrow \mathsf{compose}\Big($$

$$\mathsf{rename}_{\mathrm{left}\mapsto\mathrm{mid}}\Big(\mathsf{copy}_{\mathrm{send}\rightsquigarrow\mathrm{acq},\mathrm{recv}\rightsquigarrow\mathrm{rel}}(X)\Big),$$

$$\mathsf{rename}_{\mathrm{right}\mapsto\mathrm{mid}}(\mathrm{proc})$$

$$\Big)$$

$$X \longrightarrow \mathsf{compose}\Big(\mathsf{copy}_{\mathrm{send}\rightsquigarrow\mathrm{acq},\mathrm{recv}\rightsquigarrow\mathrm{rel}}(\mathrm{lock}), \mathrm{proc}\Big)$$

Initial marking

$$\mathrm{Sys} \longrightarrow \mathsf{compose}\left(X, \mathsf{rename}_{\mathrm{left}\mapsto\mathrm{right},\ \mathrm{right}\mapsto\mathrm{left}}\left(\mathrm{proc}'\right)\right)$$

$$X \longrightarrow \mathsf{compose}\Big(\mathsf{rename}_{\mathrm{left}\mapsto\mathrm{mid}}\left(\mathsf{copy}_{\mathrm{send}\rightsquigarrow\mathrm{acq},\mathrm{recv}\rightsquigarrow\mathrm{rel}}\left(X\right)\right), \mathsf{rename}_{\mathrm{right}\mapsto\mathrm{mid}}\left(\mathrm{proc}\right) \Big)$$

$$X \longrightarrow \mathsf{compose}\left(\mathsf{copy}_{\mathrm{send}\rightsquigarrow\mathrm{acq},\mathrm{recv}\rightsquigarrow\mathrm{rel}}\left(\mathrm{lock}\right), \mathrm{proc}\right)$$

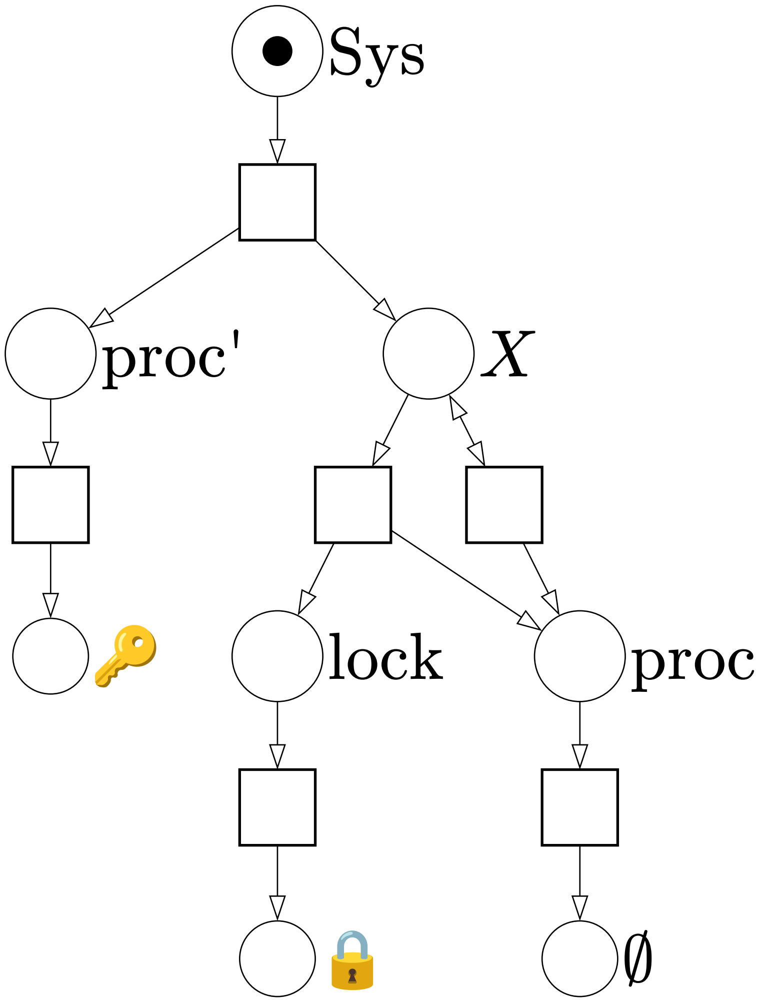
Initial marking

From the grammar

$$\begin{aligned} \mathrm{Sys} &\longrightarrow X, \mathrm{proc}' \\ X &\longrightarrow X, \mathrm{proc} \\ X &\longrightarrow \mathrm{lock}, \mathrm{proc} \end{aligned}$$

From the initial states

$$\begin{aligned} \mathrm{proc}' &\longrightarrow 🔑 \\ \mathrm{proc} &\longrightarrow \emptyset \\ \mathrm{lock} &\longrightarrow 🔒 \end{aligned}$$



Soundness

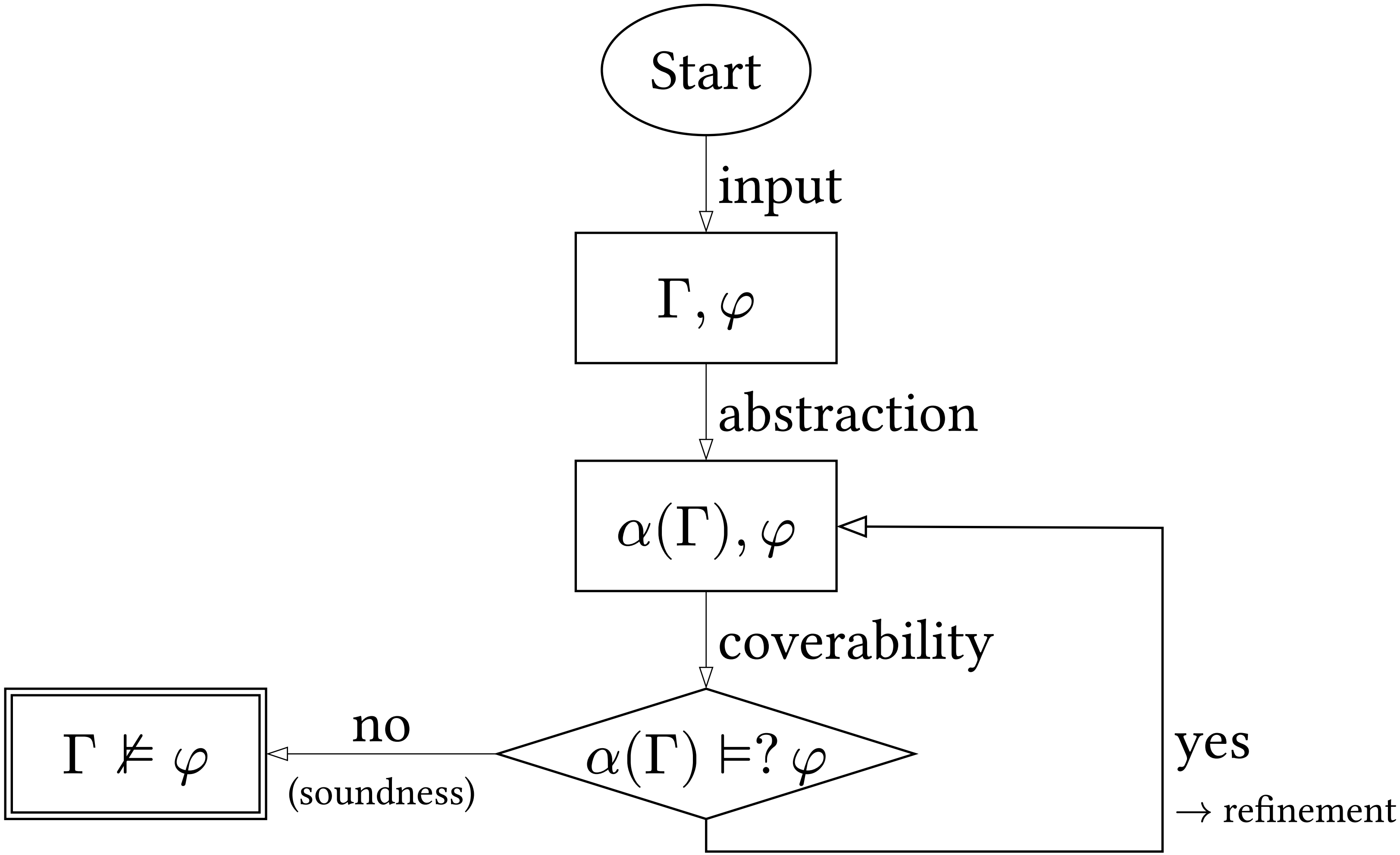
Counting abstraction is sound:

- if Γ contains undesirable behaviors then $\alpha(\Gamma)$ too
- contrapositive:
  if $\alpha(\Gamma) \not\models \varphi$ (abstract system is safe)
  then $\Gamma \not\models \varphi$ (concrete system is safe).

Reciprocal implication does not hold

- undecidability
- false positives

Automation loop



Implementation

Input: text file describing the grammar and the safety properties

- computes the abstraction
- offloads the coverability problem to a specialized solver
- ~ 7500 lines of OCaml

This example:

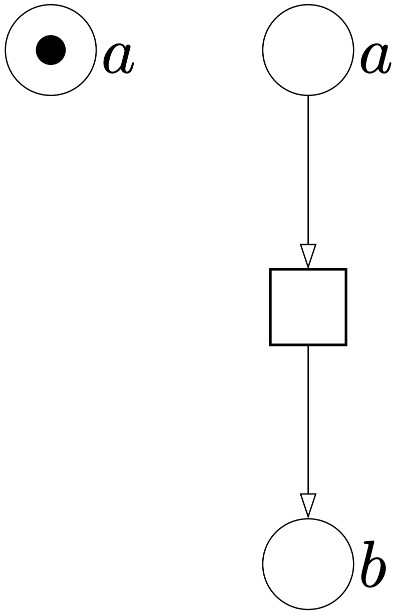
- specification in 40 lines
- 4 safety properties in 200ms

Other case studies:

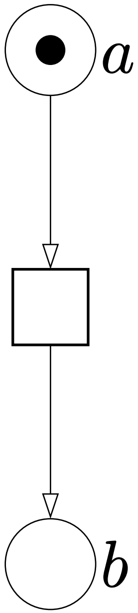
- 15 examples, 7 architectures, 27 safety properties

7. Refinement

A false positive



$$\xrightarrow{\text{fold}}$$

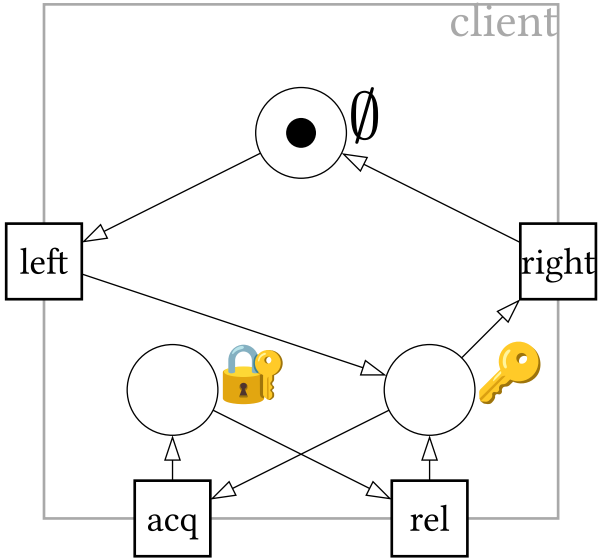


$b \geq 1$ is **not** reachable

$b \geq 1$ **is** reachable

Contracts

- formula that restricts firing sequences
- boolean contract: $\neg t$ means "no admissible firing sequence fires t"
- problem becomes reachability through **only firing sequences that satisfy the contract**



Contract:

$$(\text{acq} \vee \text{right} \Rightarrow \text{left}) \wedge (\text{rel} \Rightarrow \text{acq})$$

Composing contracts

- Finite domain (boolean formulas, bounded number of variables)
- If C_1,C_2 are contracts for N_1,N_2,
  then $C_1 \wedge C_2$ is a contract for $\mathsf{compose}(N_1, N_2)$.
- Fixed point is computable.

The construction is lossy, but can be more accurate than folding without contracts.

Conclusion

- technique to reduce infinite systems to finite instances
- in part architecture-agnostic
- observed efficient in practice

Future work

- explore completeness
- improve refinements
- encode more complex systems (infinite behaviors, reconfigurations)

8. Appendix

Full grammar

```
term lock(send, recv) ::= {
    (emp) -> [recv] -> (locked) -> [send] -> (emp);
    token (locked);
}

term client(left, right, acq, rel) ::= {
    (emp) -> [left] -> (key) -> [acq] -> (unlocked)
      -> [rel] -> (key) -> [right] -> (emp);
    token (emp);
}

term once(t) ::= {
    (p) -> [t];
    token (p);
}
```

Full grammar


```
gram gamma ::= {
    start Sys();

    sys: Sys() ->
        Arc(left, right, send, recv)
        || client(right, left, send, recv);

    rec: Arc(left, right, send, recv) ->
        Arc(left, mid, send!acq, recv!rel)
        || client(mid, right, acq, rel);

    ini: Arc(left, right, send, recv) ->
        lock(send!acq, recv!rel)
        || client(left, right, acq, rel);
}
```

Full grammar


```
with gamma do {
    do {
        safety EF (client.(key) > 1);
        safety EF (client.(unlocked) + lock.(locked) > 1);
    }
    do {
        choose client*i;
        safety EF (client*i.(unlocked) > 0 /\ lock.(locked) > 0);
    }
    do {
        choose client*j;
        choose client*k;
        safety EF (client*j.(key) > 0 /\ client*k.(key) > 0);
    }
}
```

Case studies

Filename (`.gram`)	Architecture	Property	Result	Count	Depth	Runtime (ms) excl. oracle	Runtime (ms) incl. oracle
`philos`	Ring	Mutual exclusion	Negative	2	4	98 ± 8	105 ± 10
`philos-asym`	Ring	Mutual exclusion	Negative	4	4	132 ± 7	176 ± 15
`ring`	Ring	Global uniqueness	Negative	2, 8	3, 4	62 ± 1	119 ± 17
`leader-election`	Ring	Mutual exclusion	Negative	2	2	46 ± 2	119 ± 5
`server-loop`	Ring of stars	Mutual exclusion	Negative	40	7	903 ± 74	1533 ± 478
`star`	Star	Global uniqueness	Negative	2	2	54 ± 6	78 ± 16
`star-ring`	Linked star	Global uniqueness	Negative	2	2	54 ± 3	89 ± 23
`tree-dfs`	Binary tree	Global uniqueness	Mixed	5	5	75 ± 8	129 ± 40
`tree-down`	Binary tree	Global uniqueness	Negative	3	5	47 ± 2	63 ± 16
`tree-halves`	Binary tree	Mutual exclusion	Negative	4	4	93 ± 13	362 ± 46
`tree-nav`	Linked tree	Global uniqueness	Negative	2, 12	4, 5	130 ± 8	201 ± 33
`coverapprox`	Ring	Unreachability	Mixed	2	2	81 ± 26	254 ± 51
`propagation`	Ring	Unreachability	Mixed	2	3	136 ± 52	1041 ± 156
`lock`	Star	Mutual exclusion	Mixed	2	2	52 ± 3	75 ± 27
`open`	Double ring	Unreachability	Unknown	2	3	95 ± 11	911 ± 127