Formal verification of infinite state systems with unbounded agents

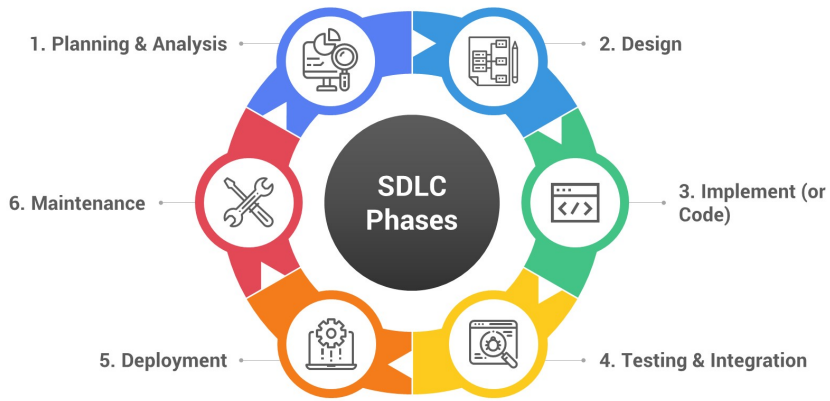
Tephilla Prince
Indian Institute of Technology Dharwad

Supervisors:
Prof. Ramchandra Phawade
Prof. S. Sheerazuddin

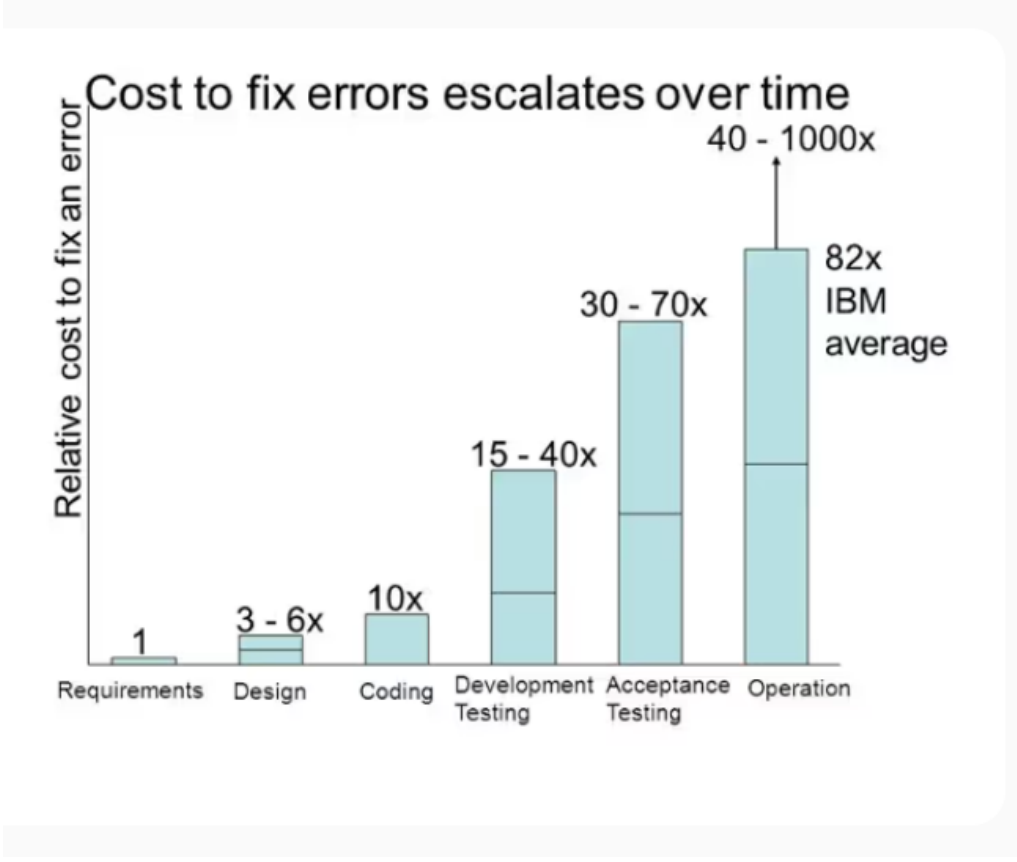
27/06/2025

Software Development Life Cycle (SDLC) Phases

Software Development Life Cycle (SDLC) Phases



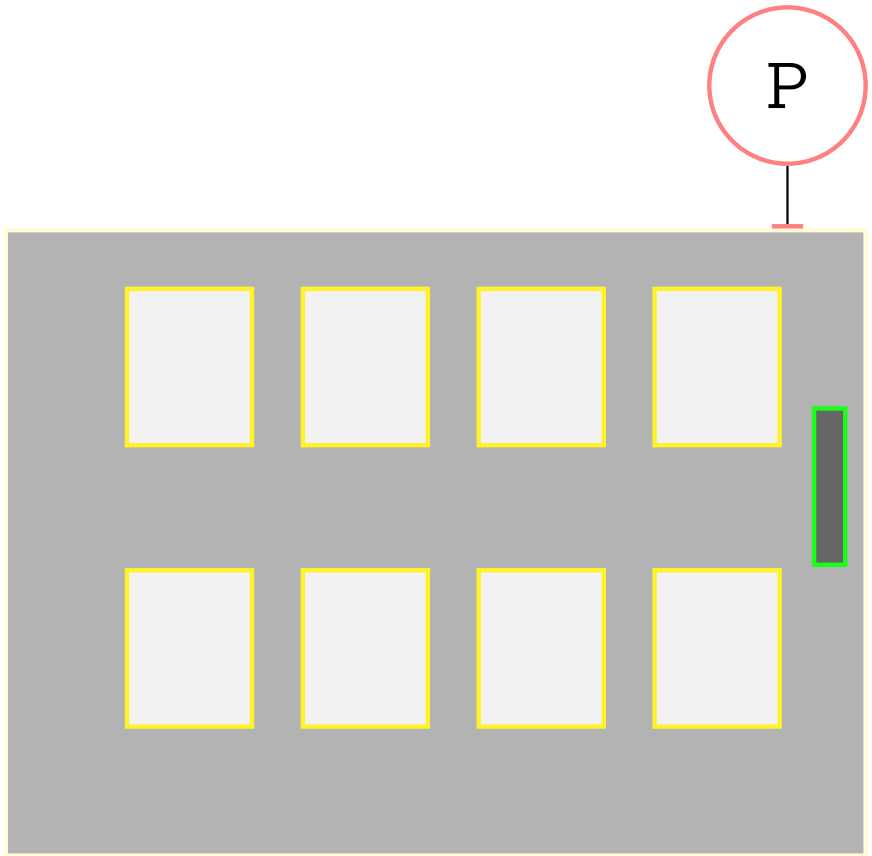
Error Costs



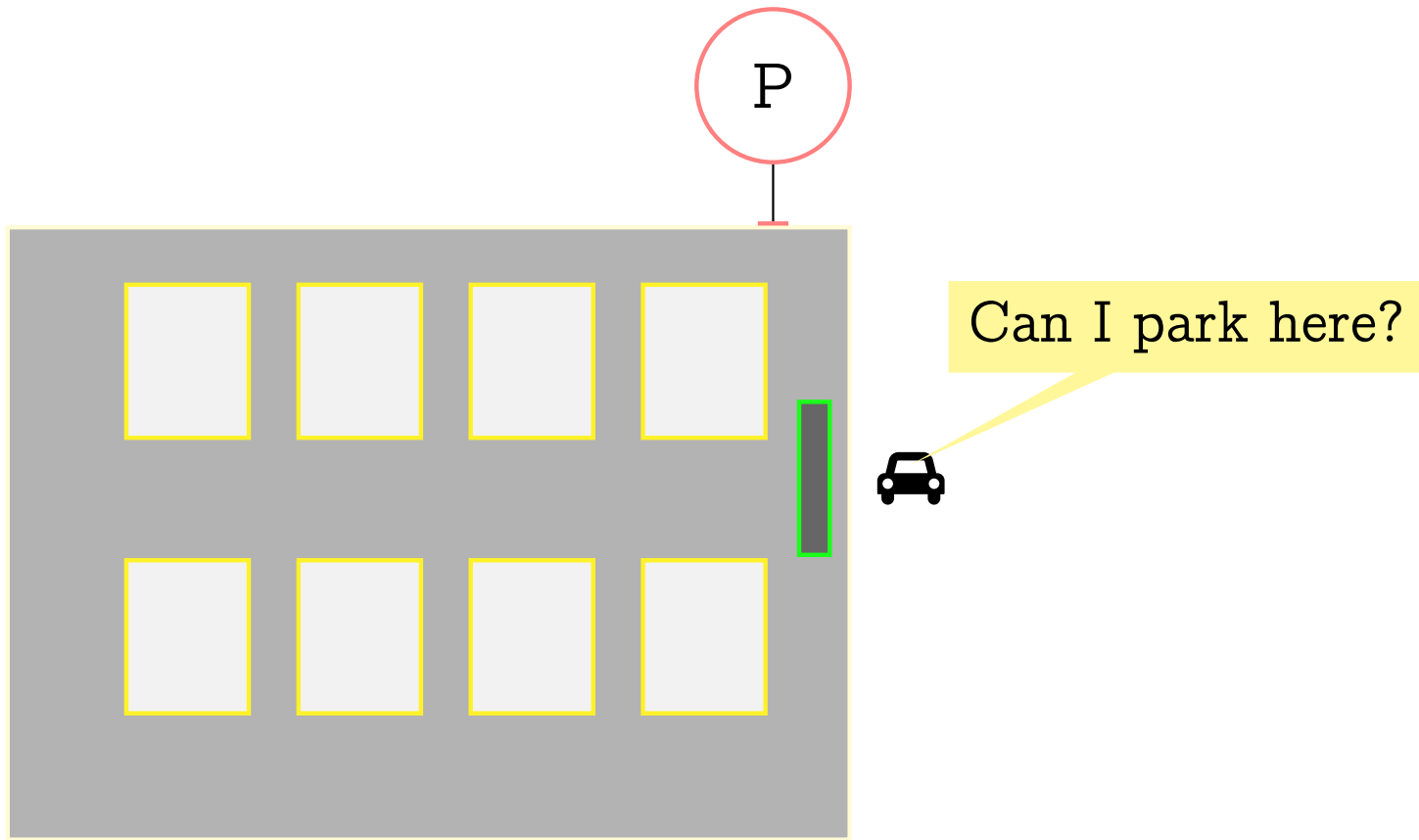
J. Stecklein, "Error cost escalation through the project life cycle, 2010

Running example

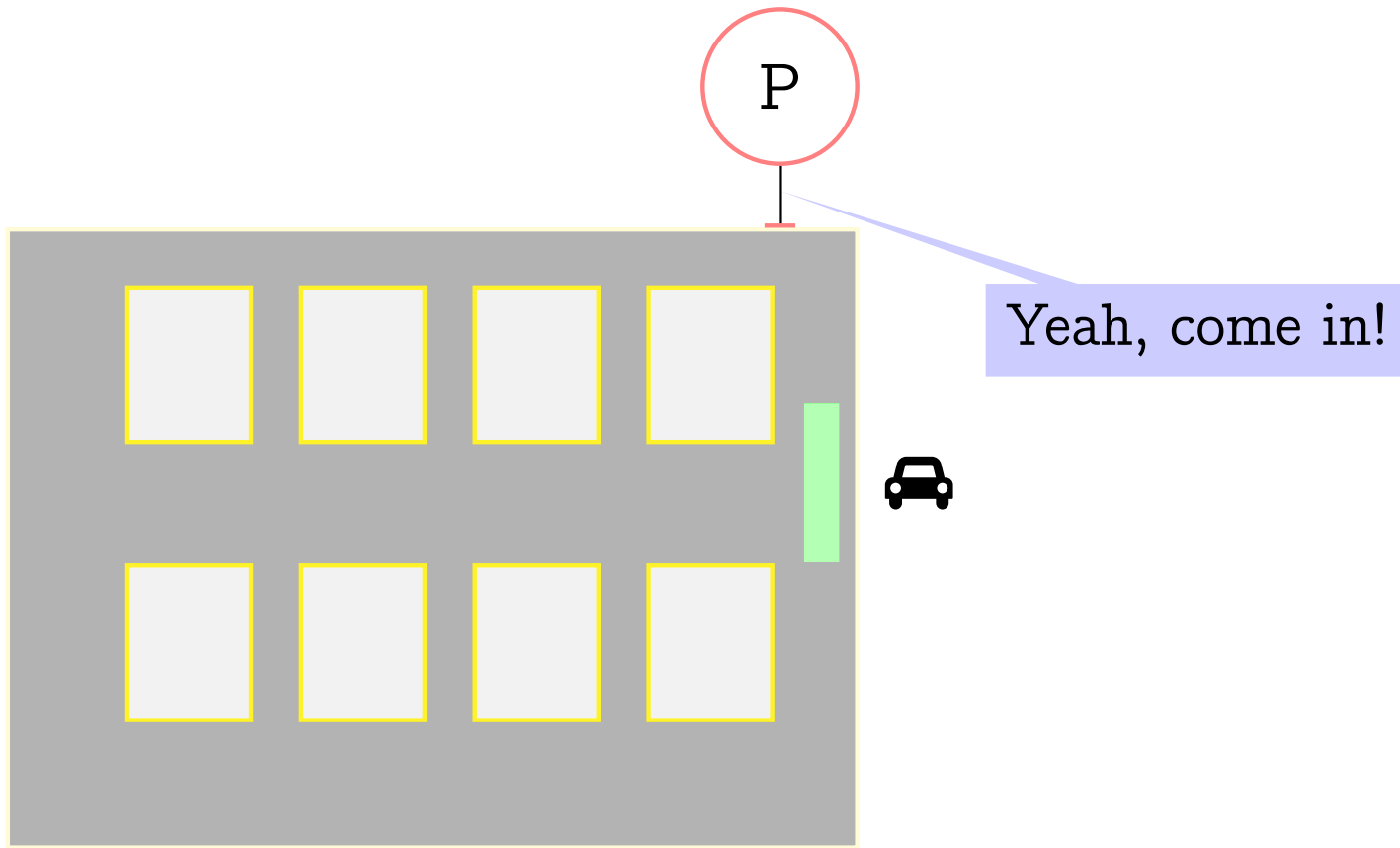
An empty Parking Lot waits for vehicles



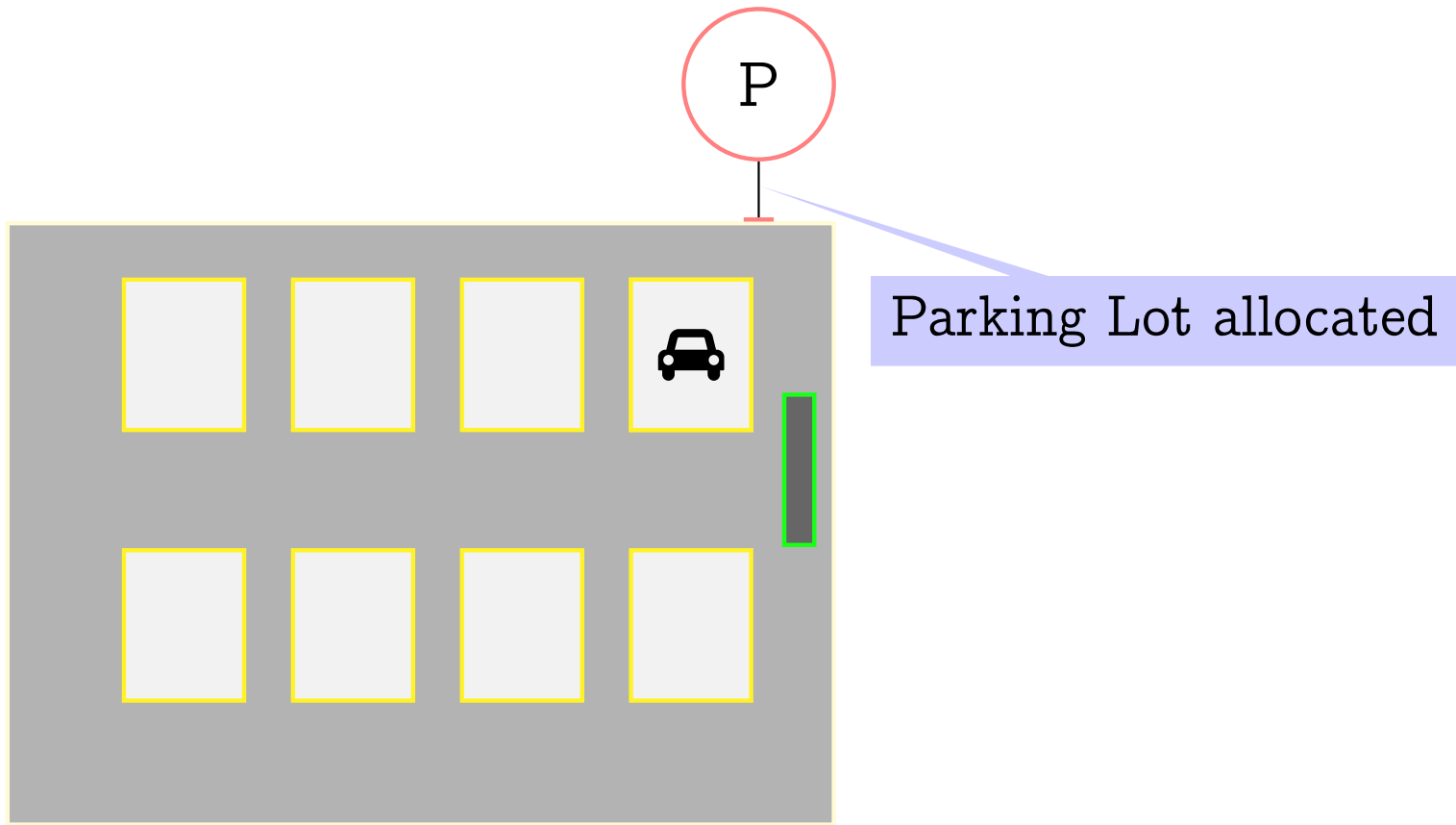
Running example



Running example

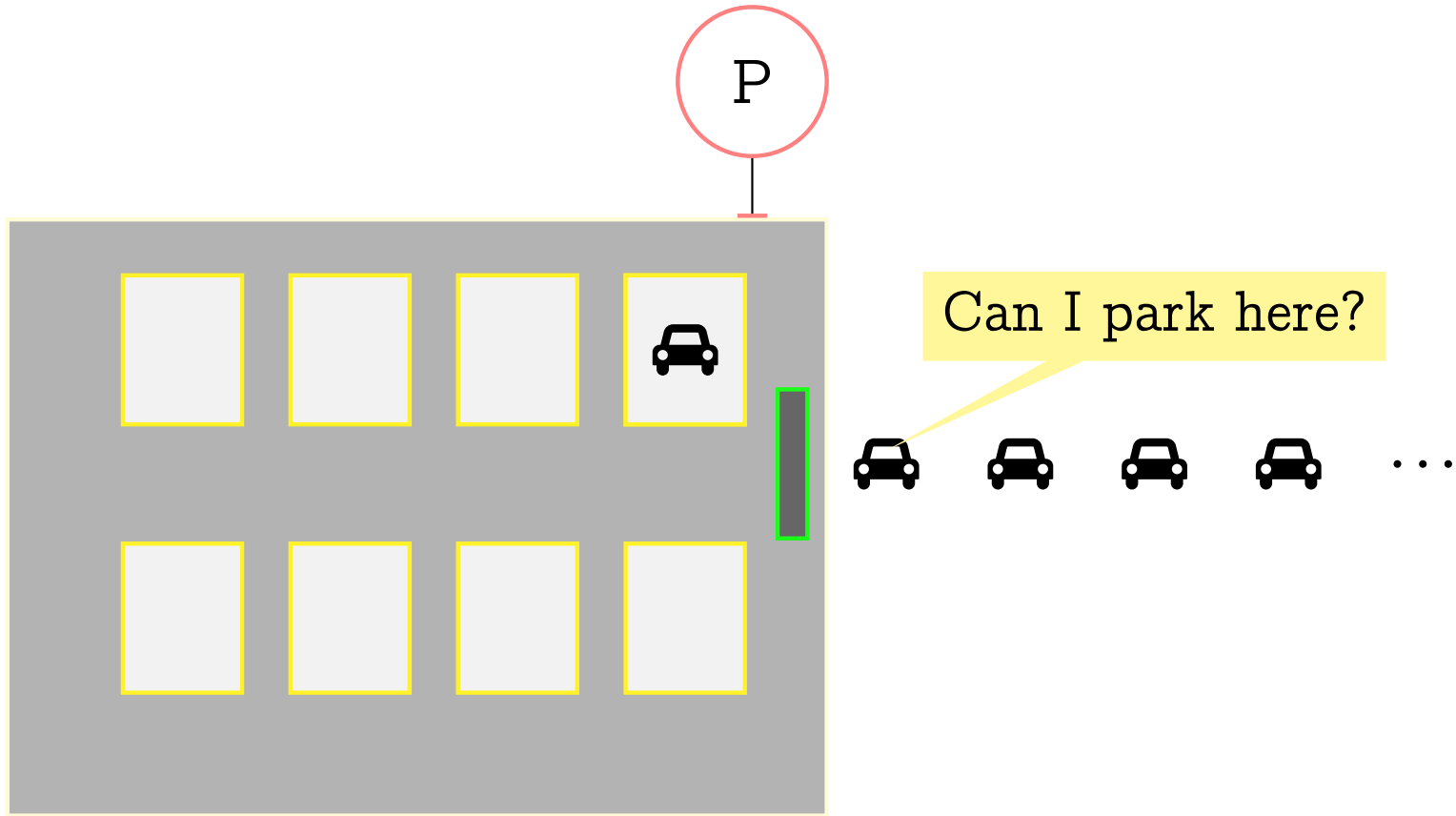


Running example



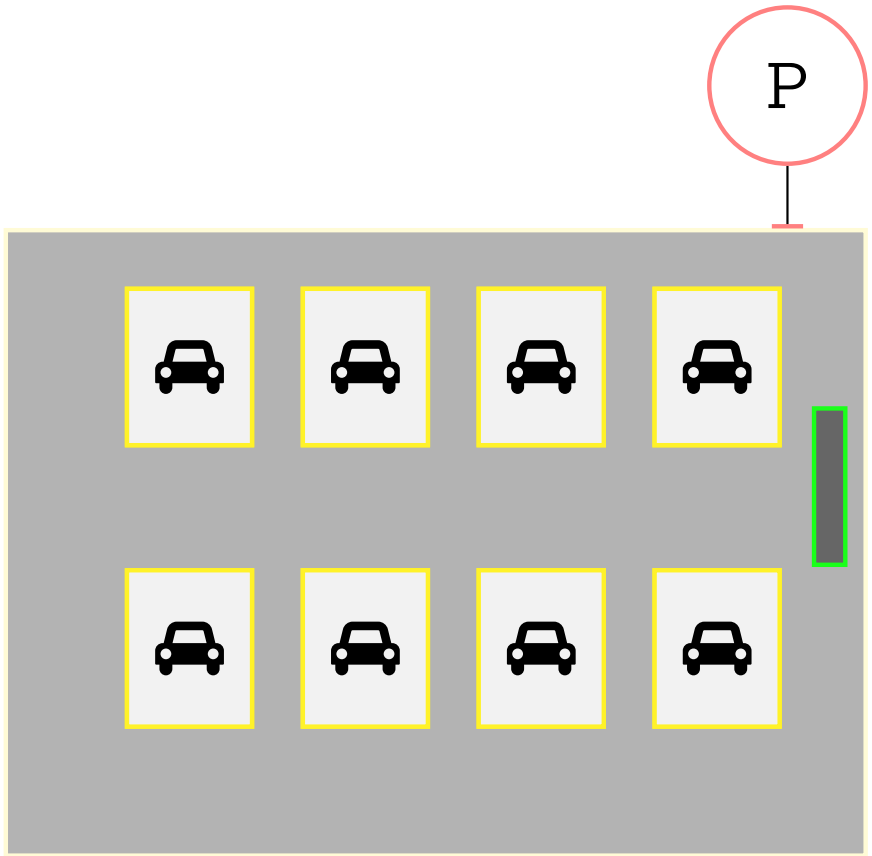
Running example

Unbounded number of parking requests

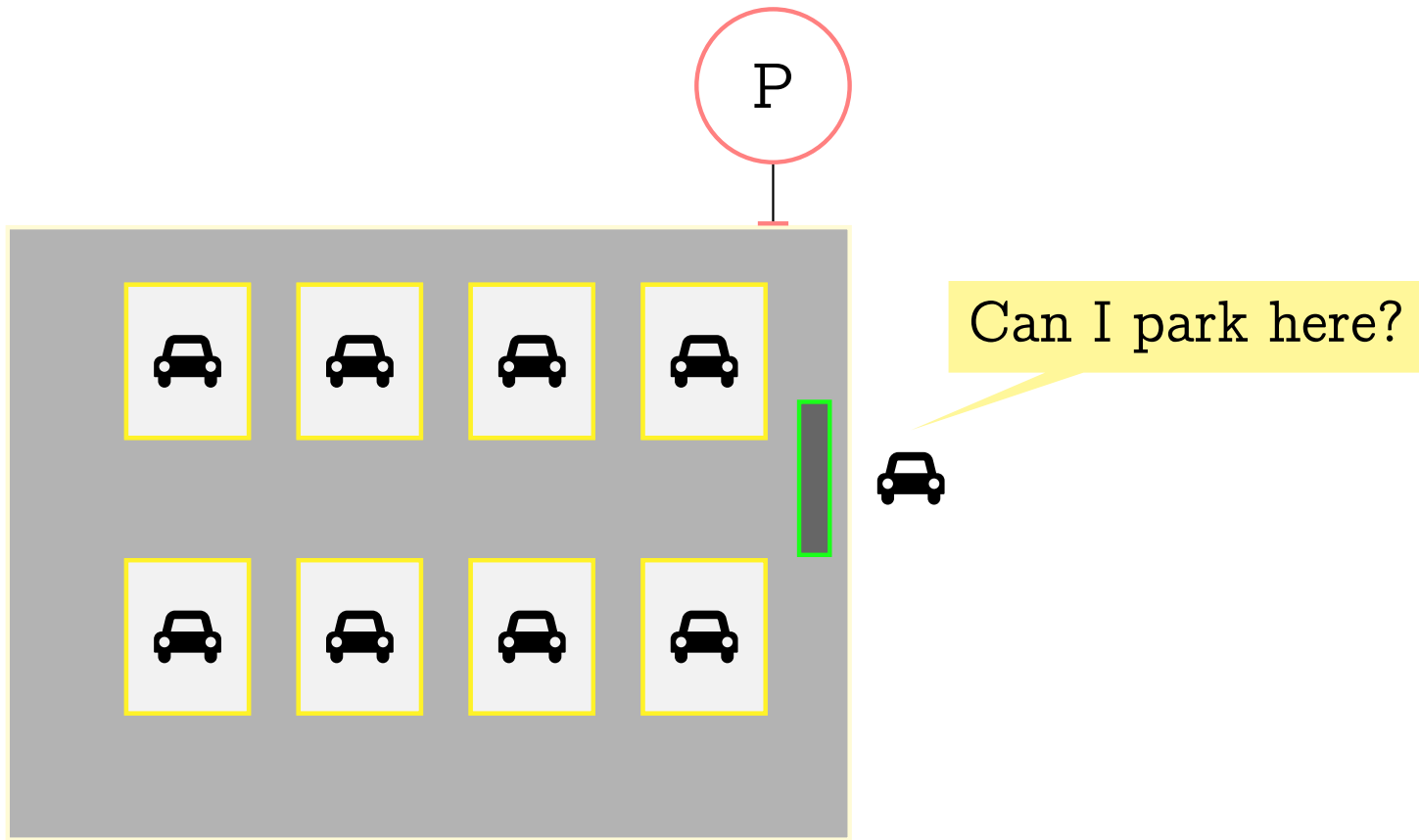


Running example

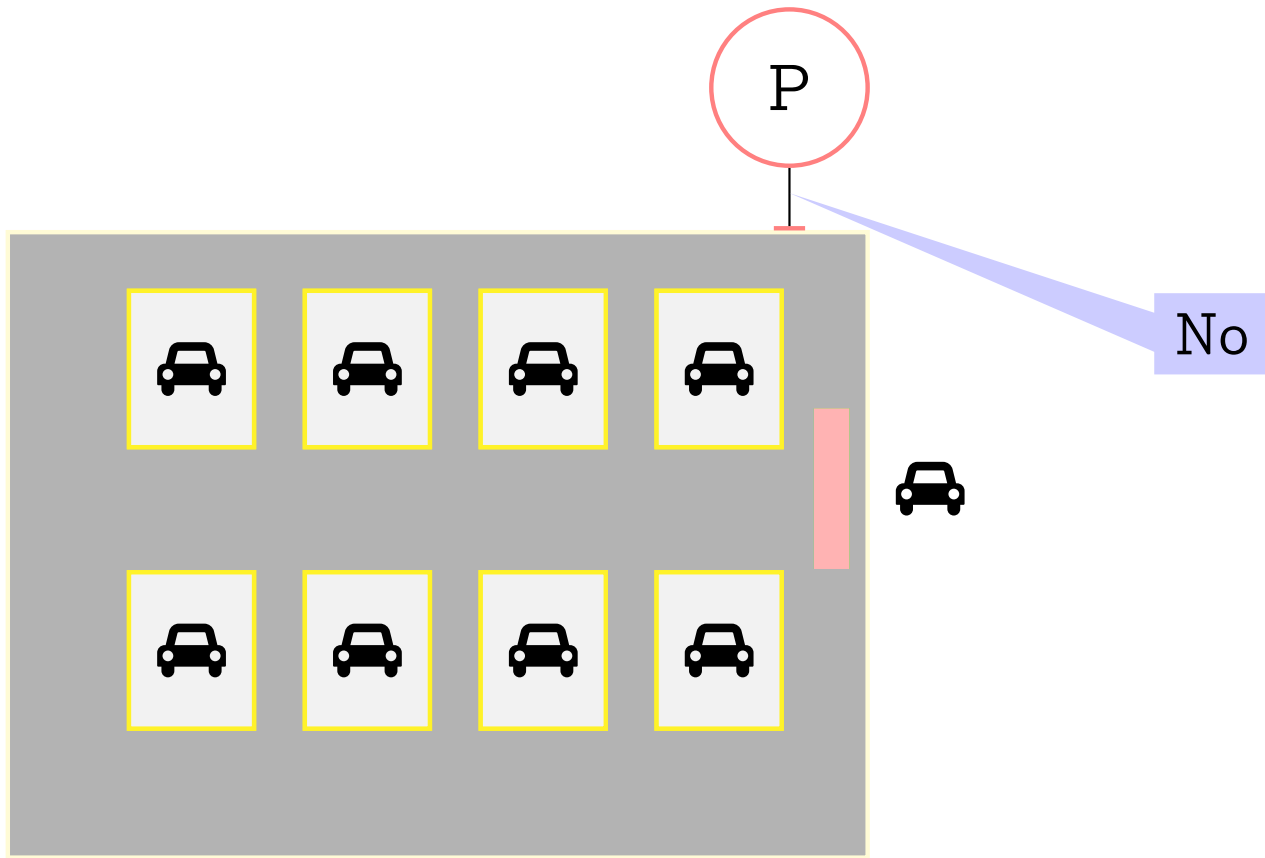
No parking space



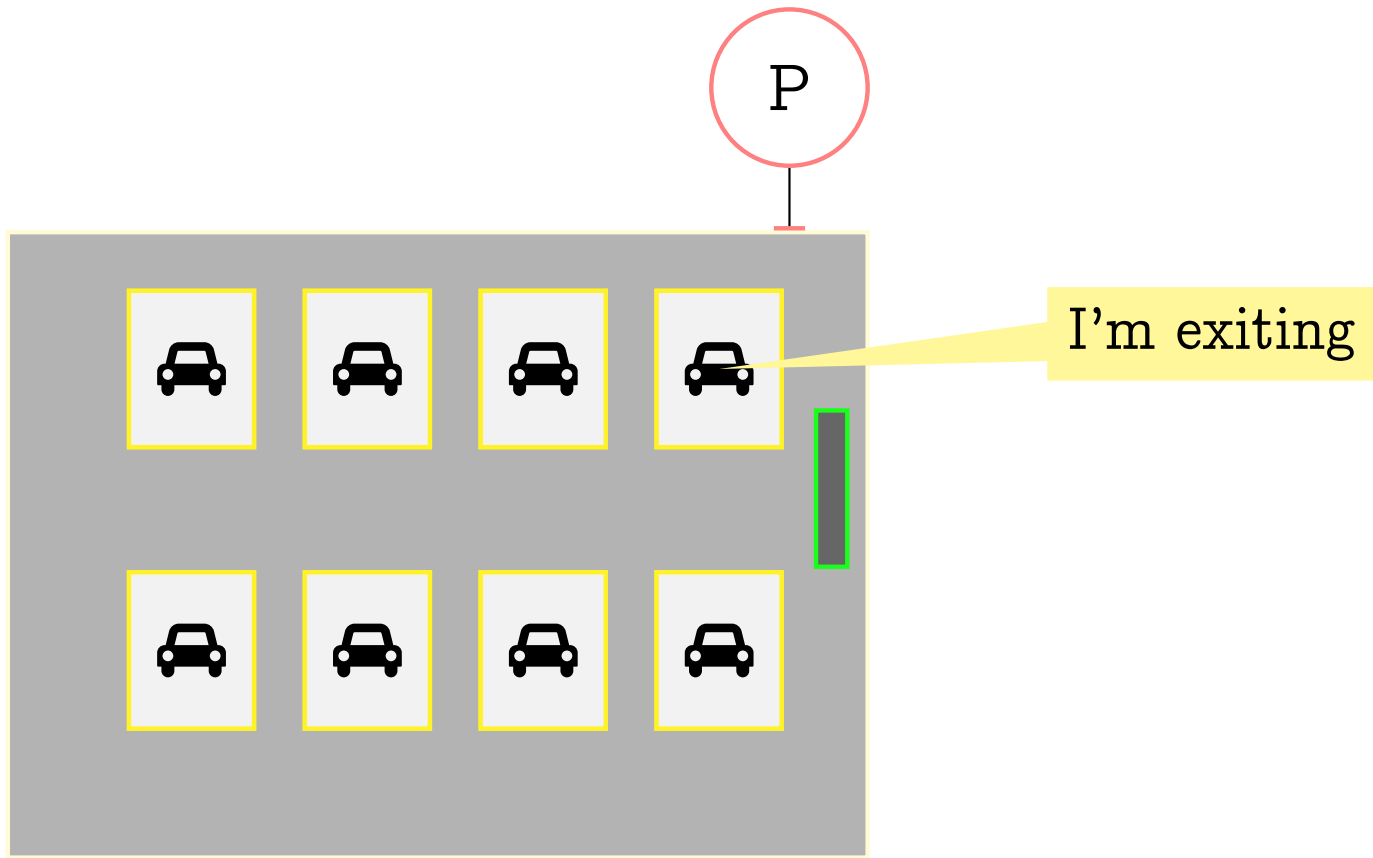
Running example



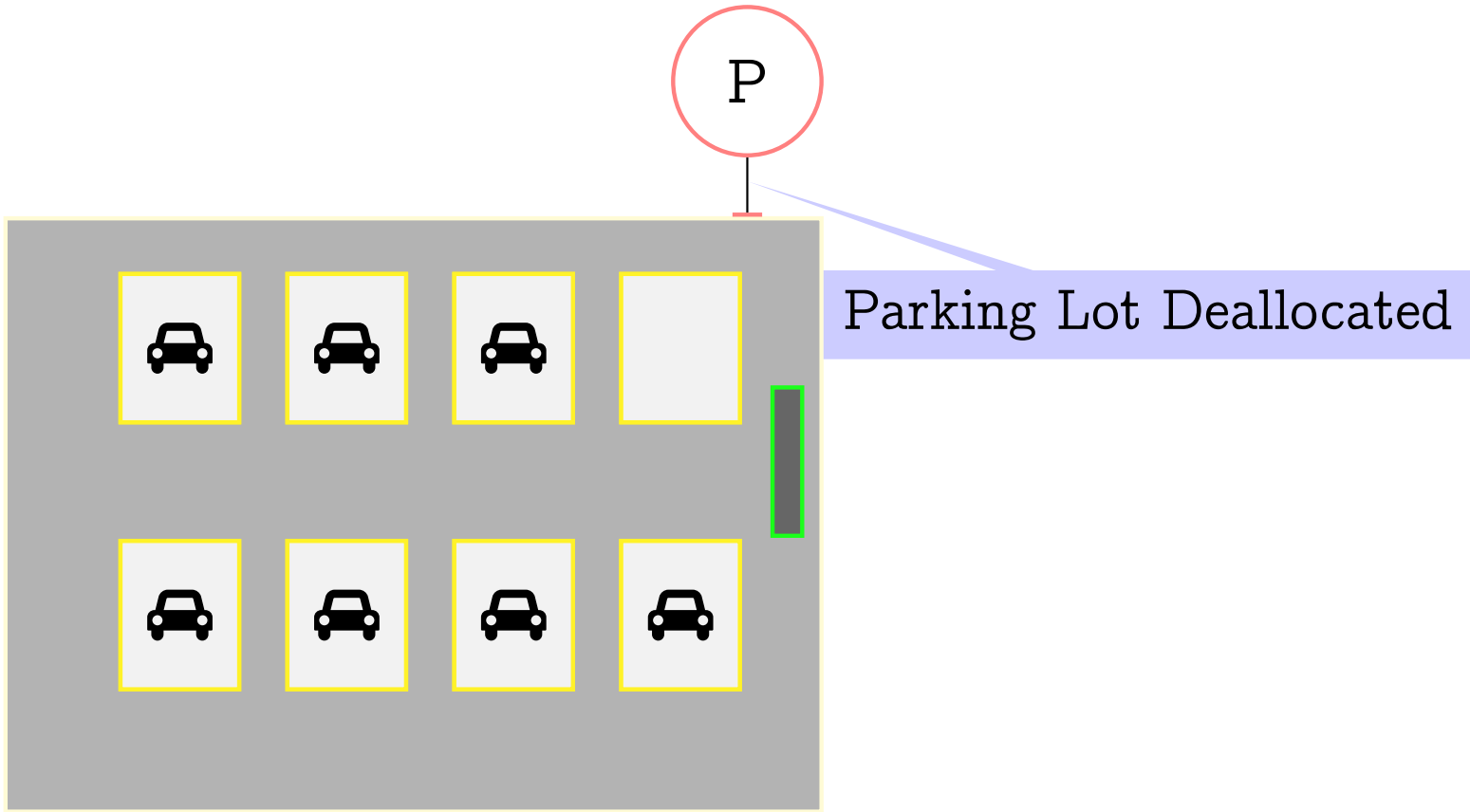
Running example



Running example

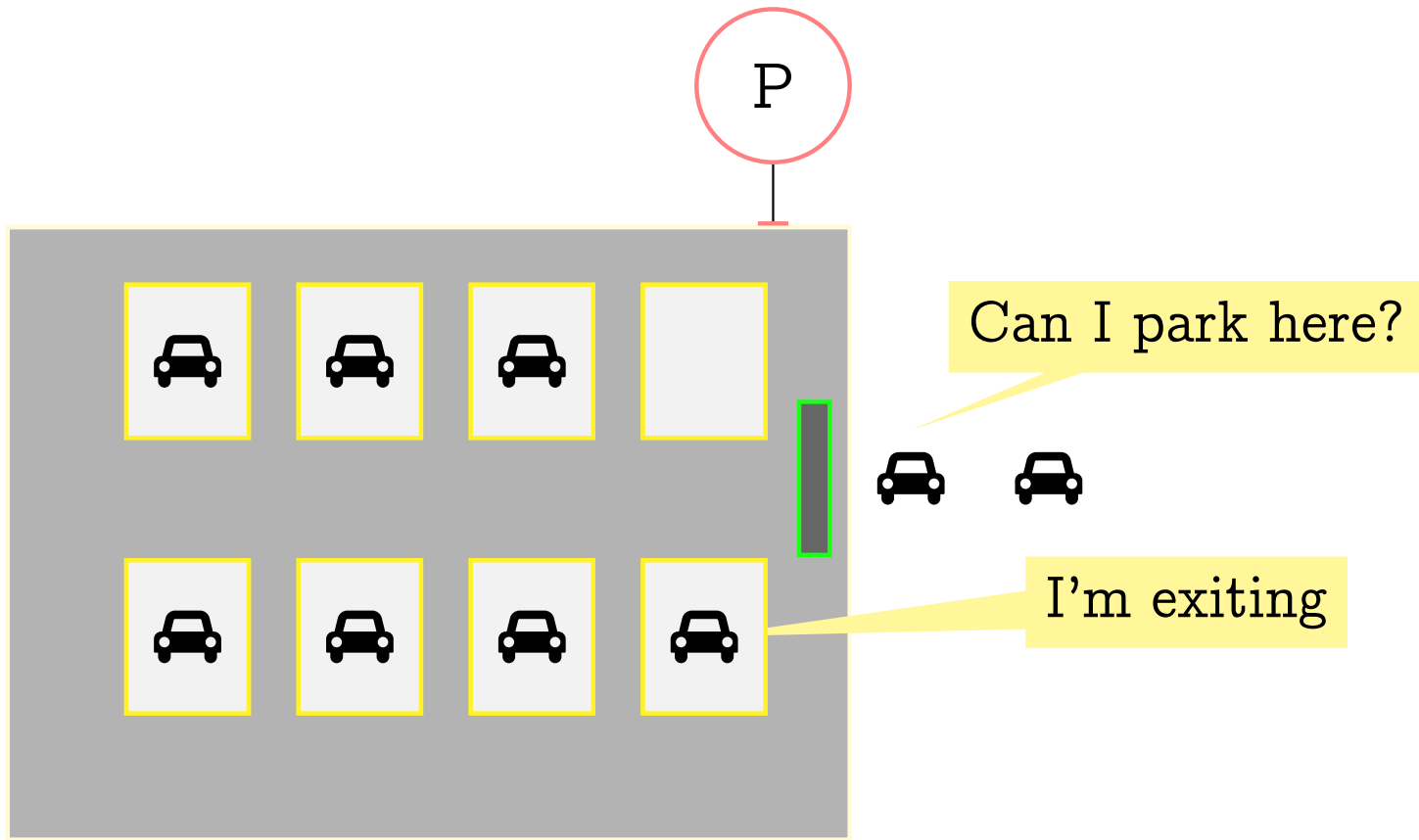


Running example

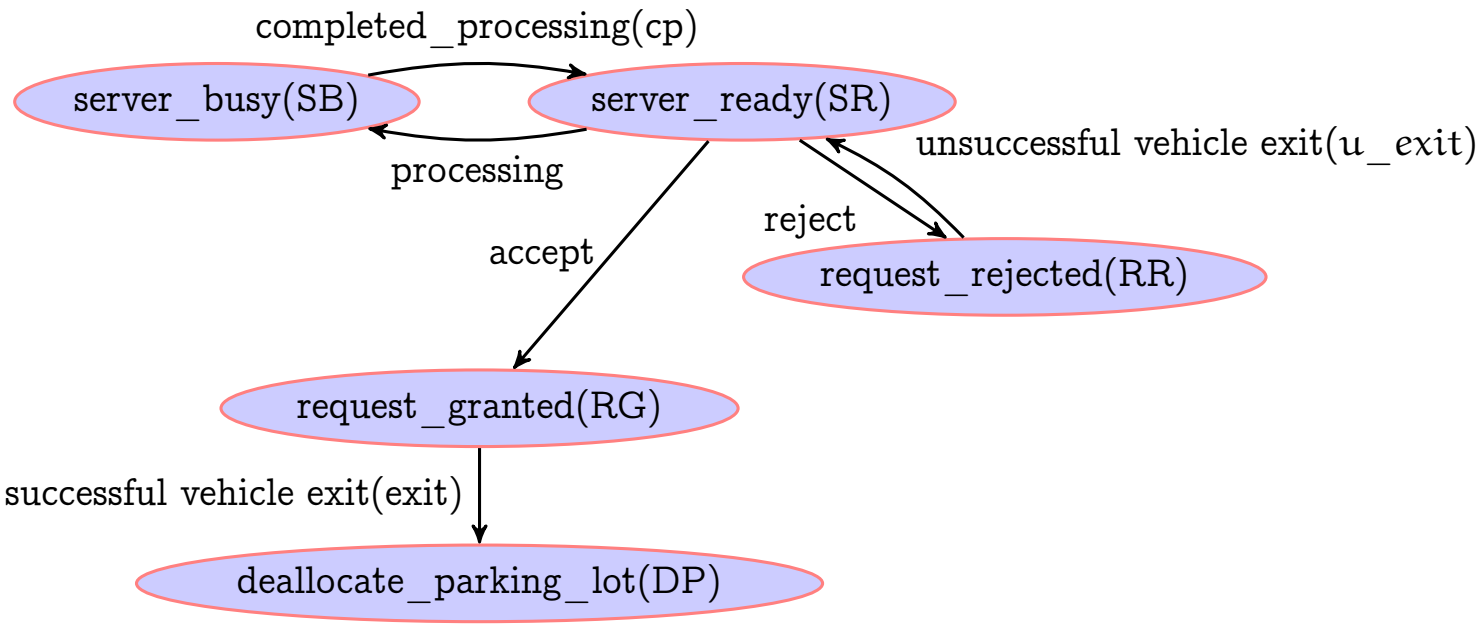


Running example

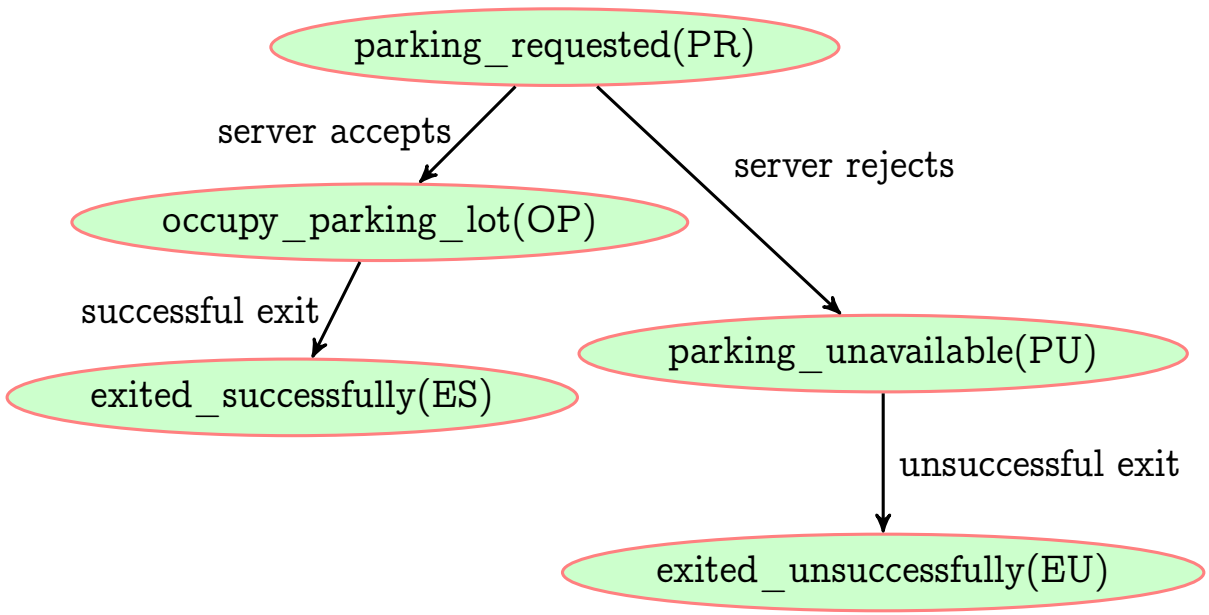
- single server-multiple client system
- unbounded clients of the same type



Case Study Parking System: Server behaviour



Case Study Parking System: Client behaviour



Research Questions

1. How do we formally verify UCS?
    1. How do we formally model UCS?

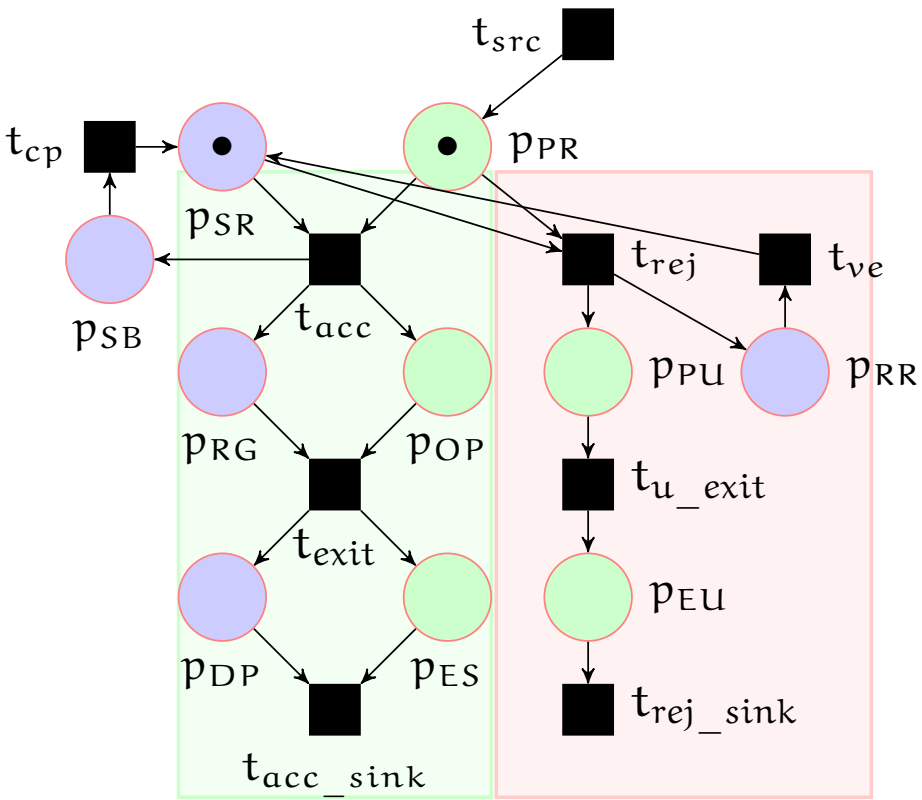
Research Questions

1. How do we formally verify UCS?
    1. How do we formally model UCS?
2. How do we express properties of client-server systems naturally?
    1. Do existing logics suffice?
    2. What additional properties are interesting in the context of unbounded client-server systems?

Research Questions

1. How do we formally verify UCS?
    1. How do we formally model UCS?
2. How do we express properties of client-server systems naturally?
    1. Do existing logics suffice?
    2. What additional properties are interesting in the context of unbounded client-server systems?
3. Are there formal verification tools for UCS? Can we add to the repertoire?
    1. How are existing tools different, unique? Is another tool necessary?

Part 1: Modelling an Unbounded Client-Server (UCS) using Petri Nets



Finite sets of atomic client propositions P_c and server propositions P_s:

$$P_c = \{\text{parking\_requested}(PR),\ \text{occupy\_parking\_lot}(OP),\ \text{parking\_unavailable}(PU),\ \text{exit\_successfully}(ES),\ \text{exited\_unsuccessfully}(EU)\}$$

$$P_s = \{\text{server\_ready}(SR),\ \text{server\_busy}(SB),\ \text{request\_granted}(RG),\ \text{request\_rejected}(RR),\ \text{deallocate\_parking\_lot}(DP)\}$$

Question: What properties of the system are we interested in?

Properties

1. Is the number of clients occupying parking lots = number of requests granted always?

   $G(\#p_{OP} = \#p_{RG})$

Properties

1. Is the number of clients occupying parking lots = number of requests granted always?

   $G(\#p_{OP} = \#p_{RG})$

2. Consider the property, $p_{PR} + p_{OP} + p_{ES} = 1$, which is an invariant.

   $G(\#p_{PR} + \#p_{OP} + \#p_{ES} = 1)$.

Properties

1. Is the number of clients occupying parking lots = number of requests granted always?

   $G(\#p_{OP} = \#p_{RG})$

2. Consider the property, $p_{PR} + p_{OP} + p_{ES} = 1$, which is an invariant.

   $G(\#p_{PR} + \#p_{OP} + \#p_{ES} = 1)$.

3. Is the number of rejected requests greater than the number of accepted requests at some point?

   $F(\#p_{EU} > \#p_{ES})$.

2D-BMC Algorithm

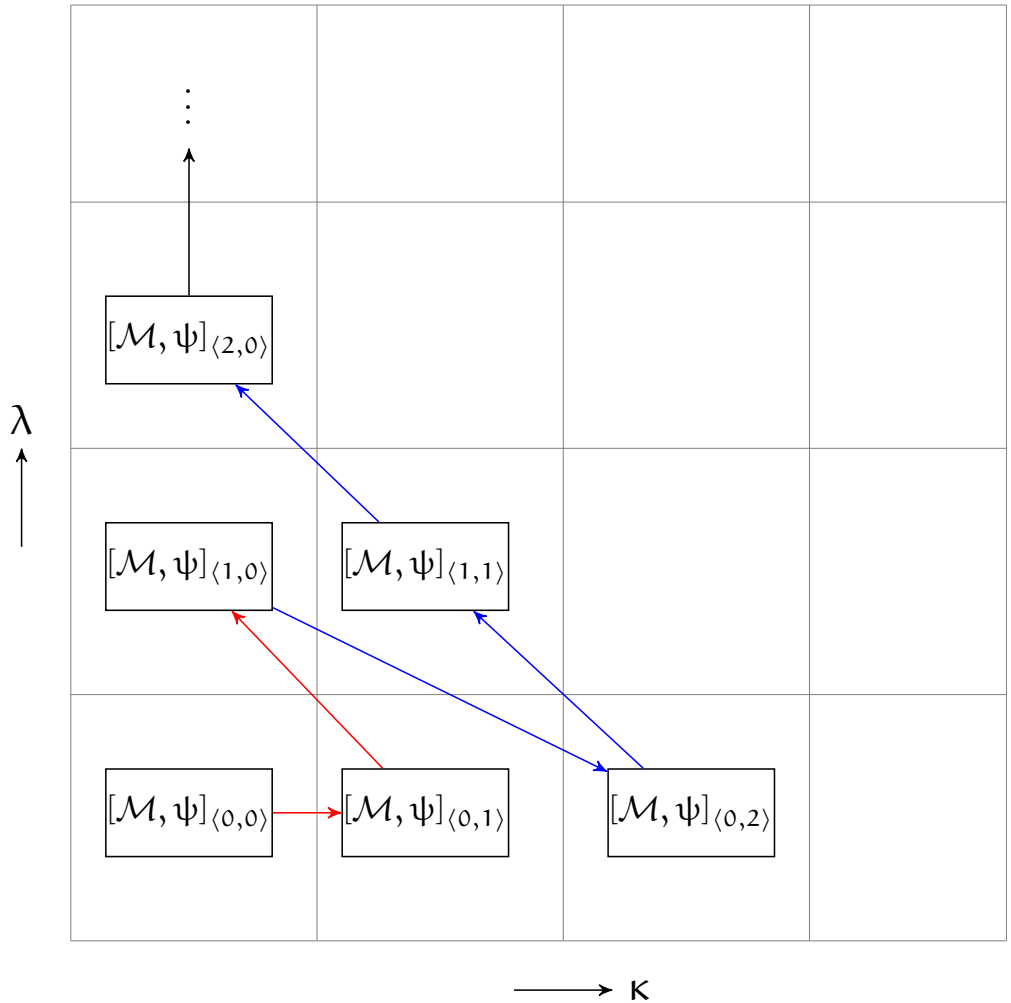


Figure: Unfolding of the encoded formula $[\mathcal{M},\psi]_{(\lambda,\kappa)}$ with respect to λ (execution steps) and κ (number of tokens)

Tool Architecture

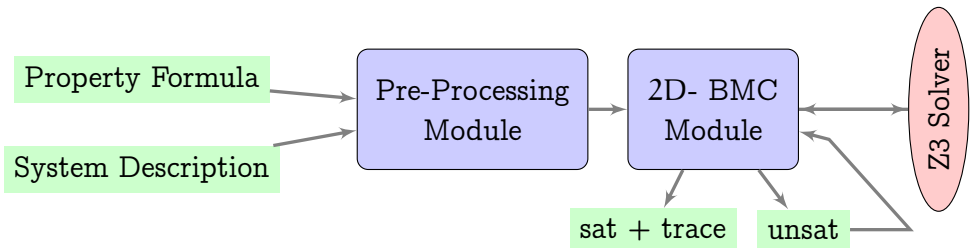


Figure: DCModelChecker architecture

Experimental Results

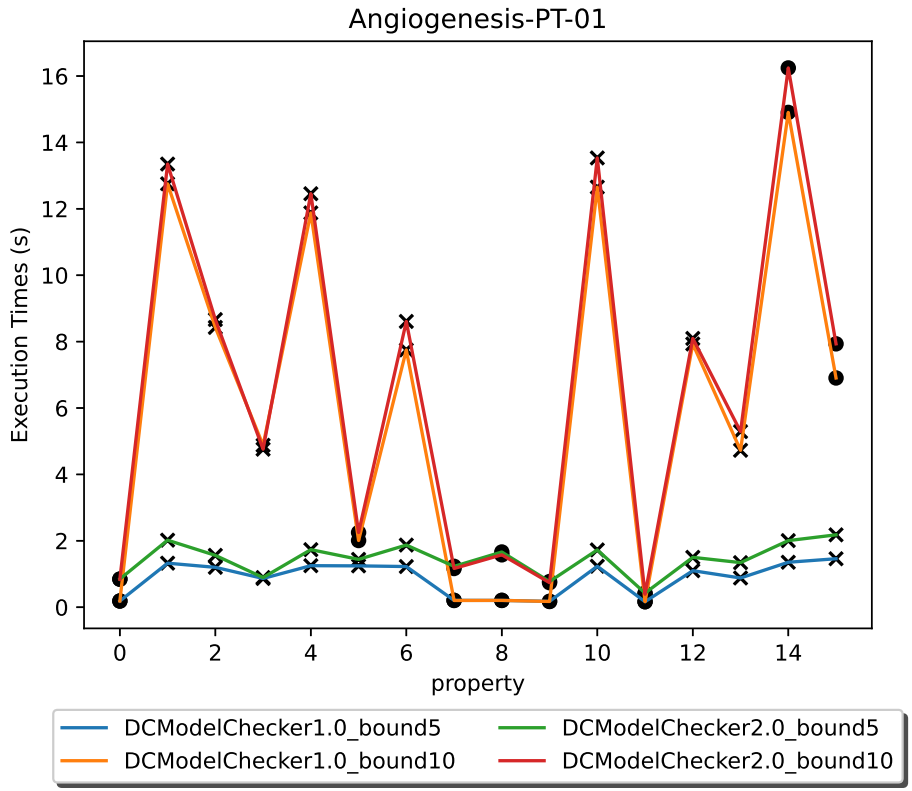


Figure: Verification of FireabilityCardinality Properties with bound 5 and 10

Verified against 16 Benchmarks from Model Checking Contest 2025

Experimental Results

Model	Execution Times(ms)			
	DCModelChecker 1.0	DCModelChecker 2.0	ITS-Tools	Tapaal
Parity	**0.010**	0.40	0.933	0.02
PGCD	**0.019**	0.98	1.487	0.07
Process	0.033	0.85	1.201	**1e-05**
CryptoMiner	0.036	0.64	0.957	**7e-06**
Murphy	0.031	0.83	1.161	**8e-06**

Table: Results of Comparative Experiments on Unbounded PNs

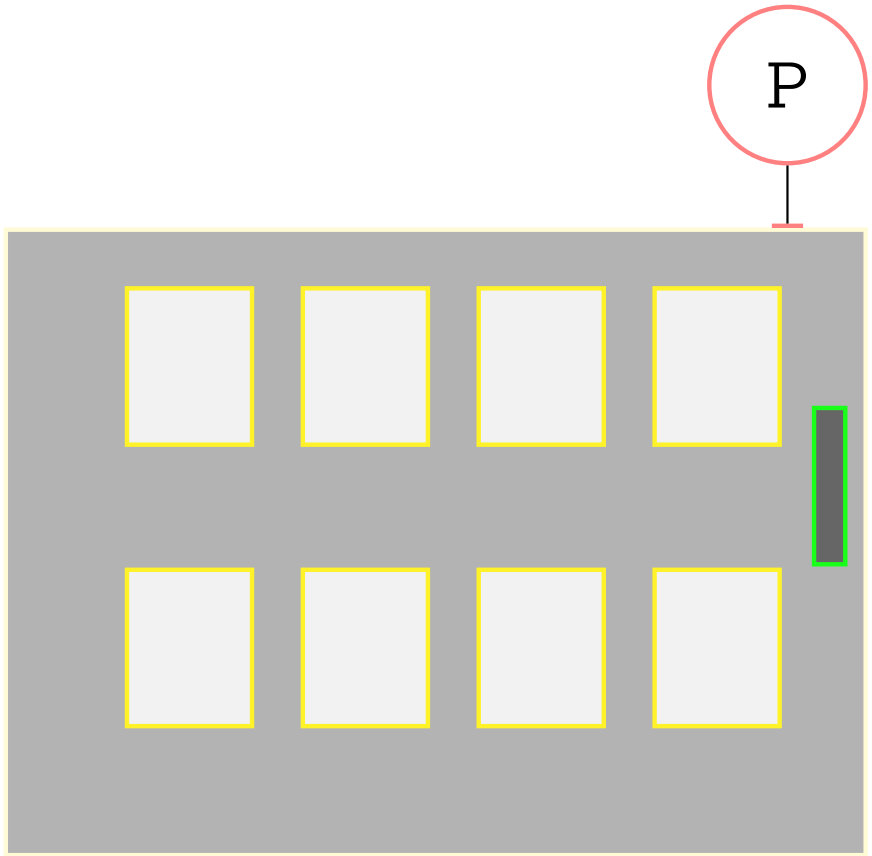
Contributions

- Part 1: Verification of Unbounded Client-Server (UCS) using Petri Nets and Linear Temporal Logic with integer arithmetic using 2D-BMC

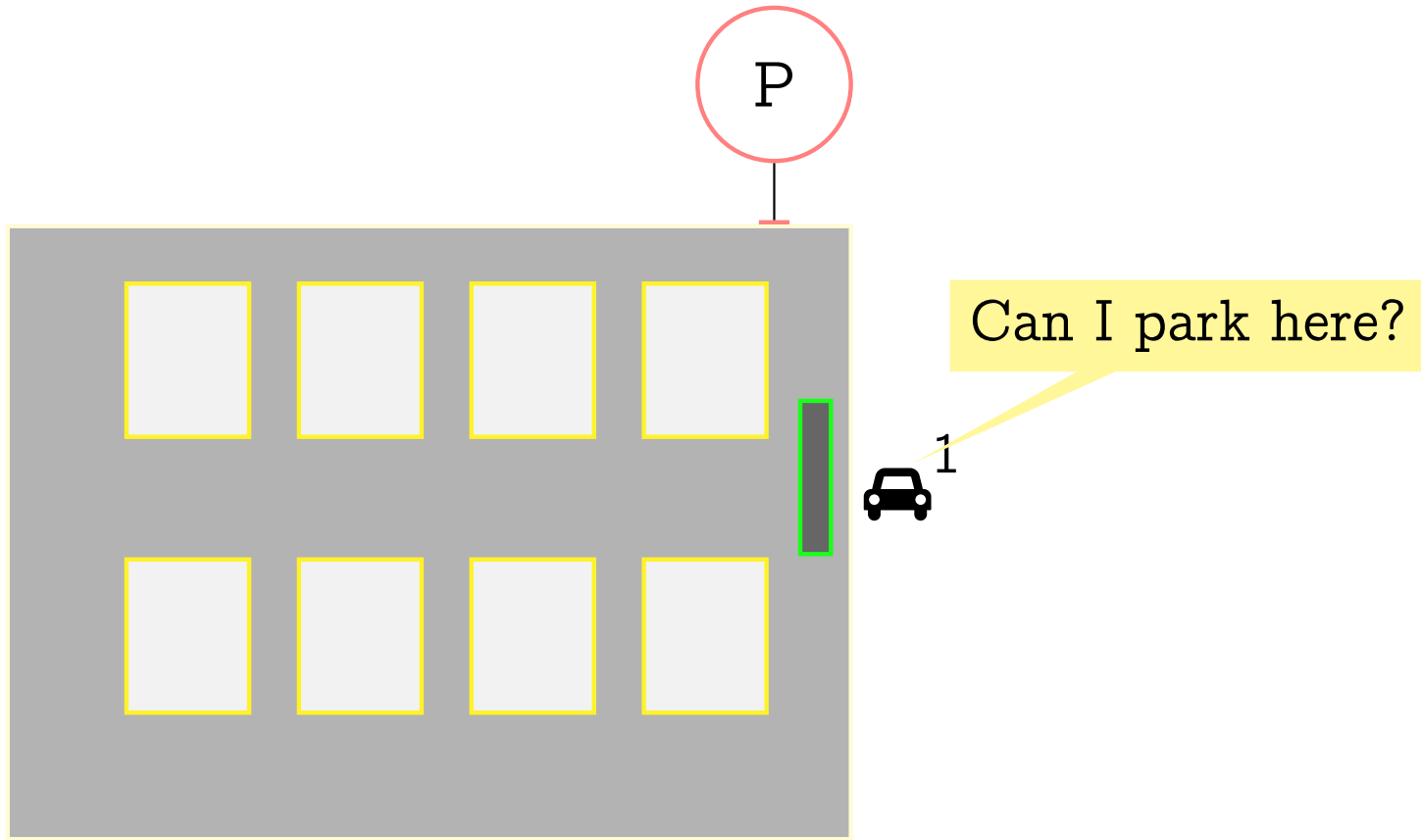
What if clients needed to be distinguished?

Running example

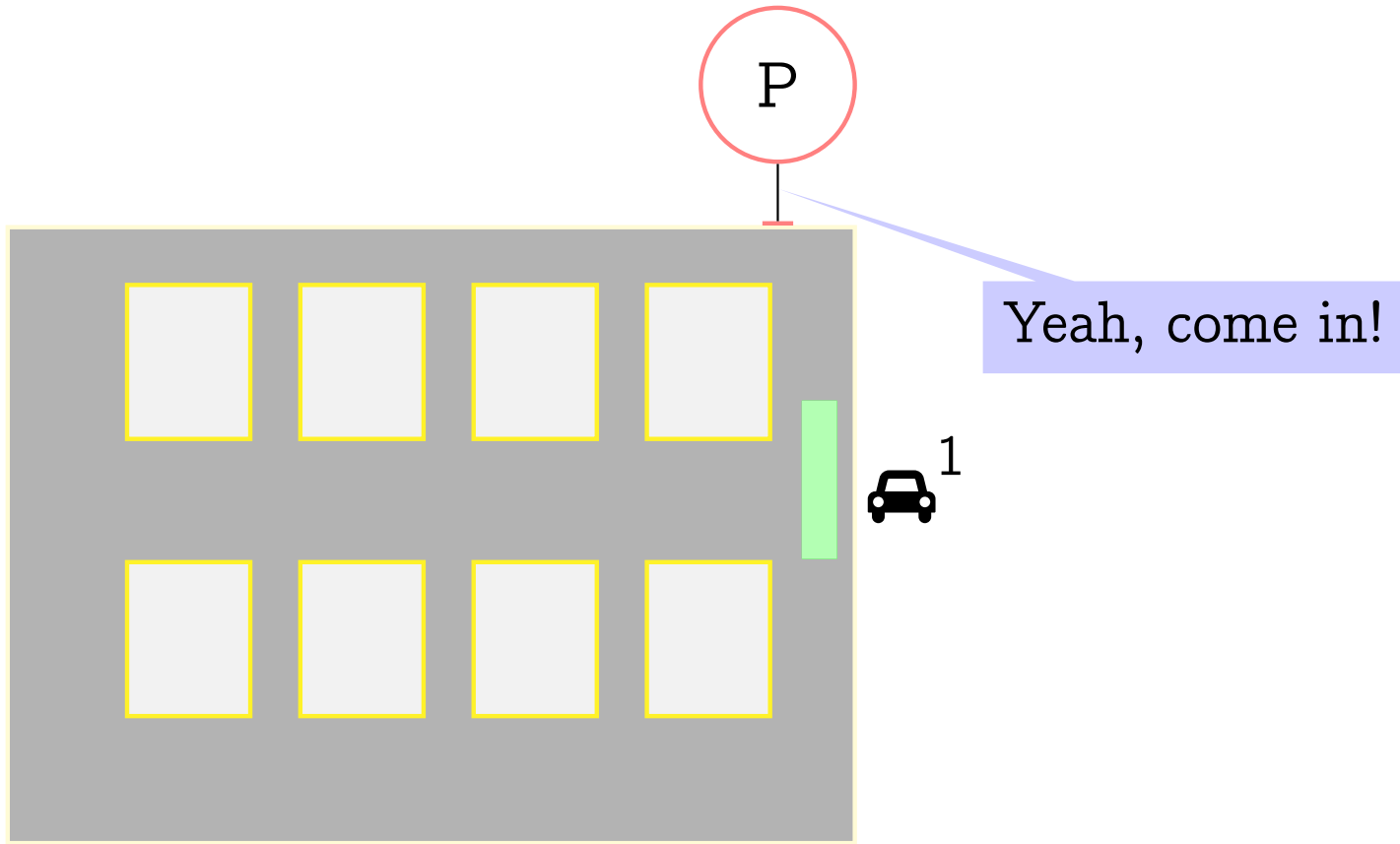
An empty Parking Lot waits for vehicles



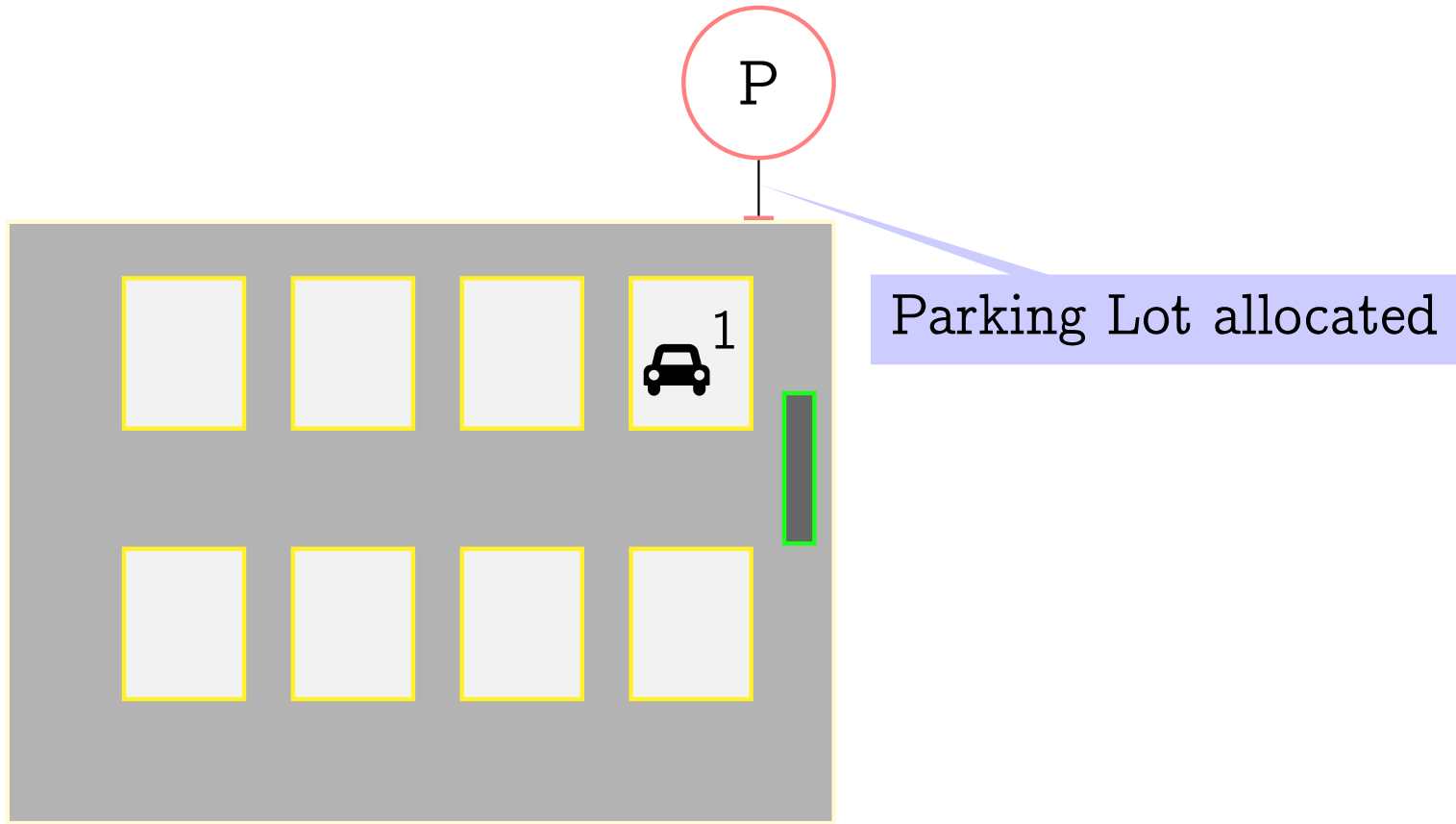
Running example



Running example

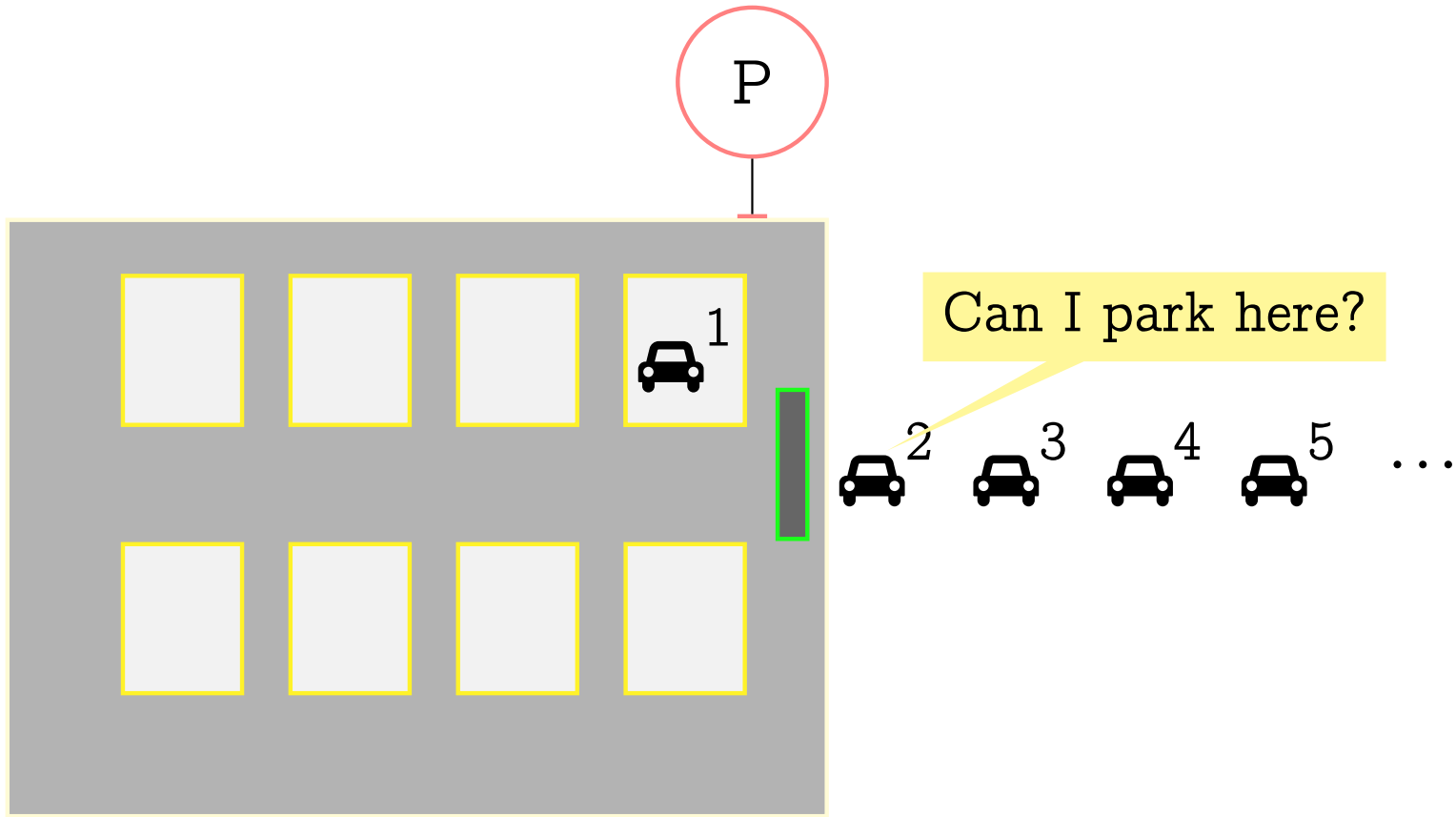


Running example



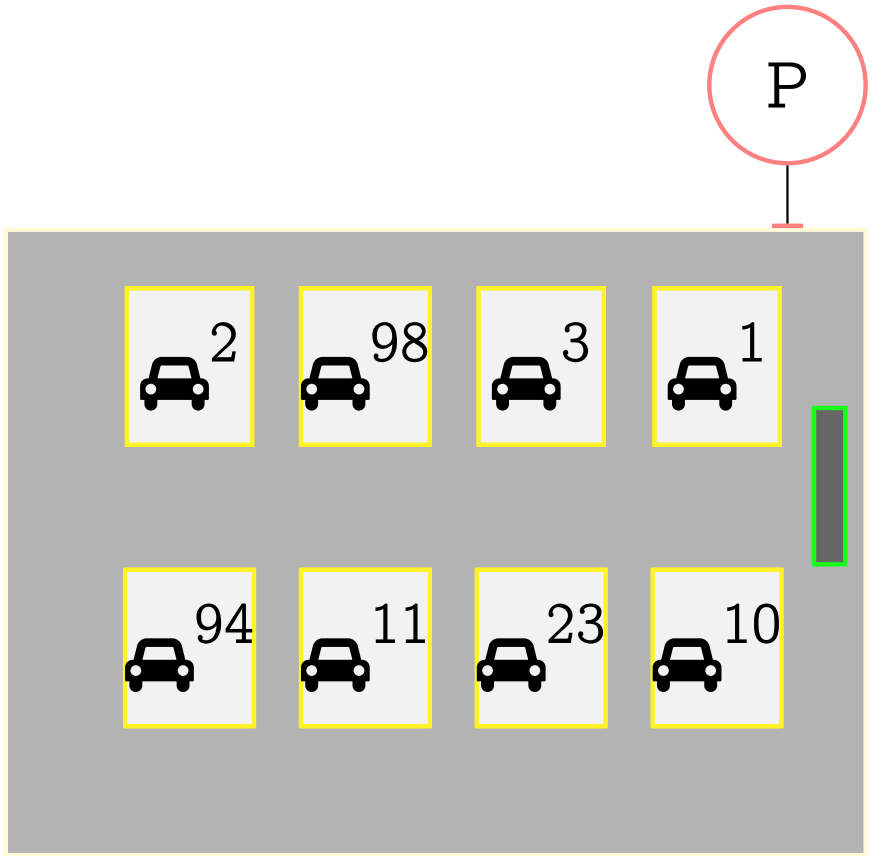
Running example

Unbounded number of parking requests

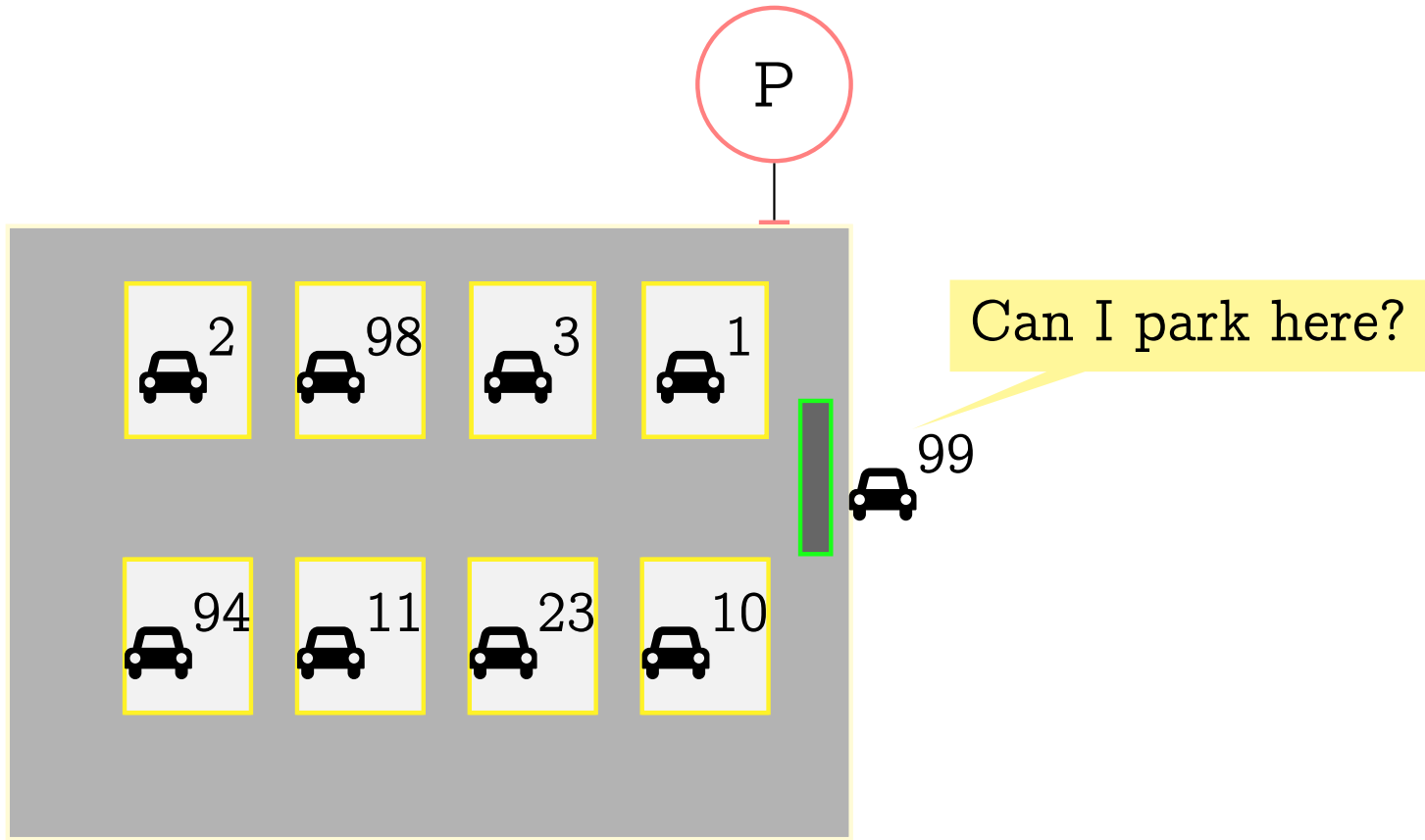


Running example

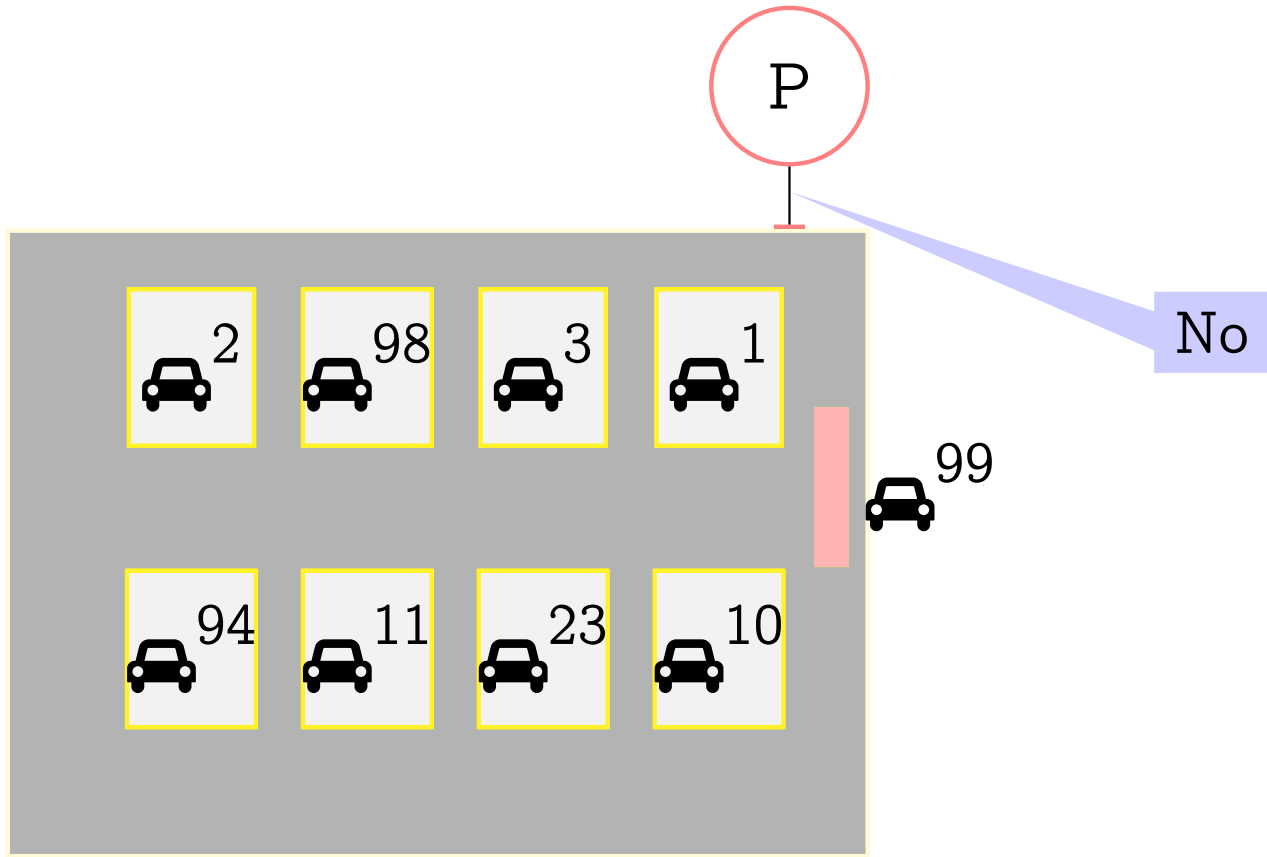
No parking space



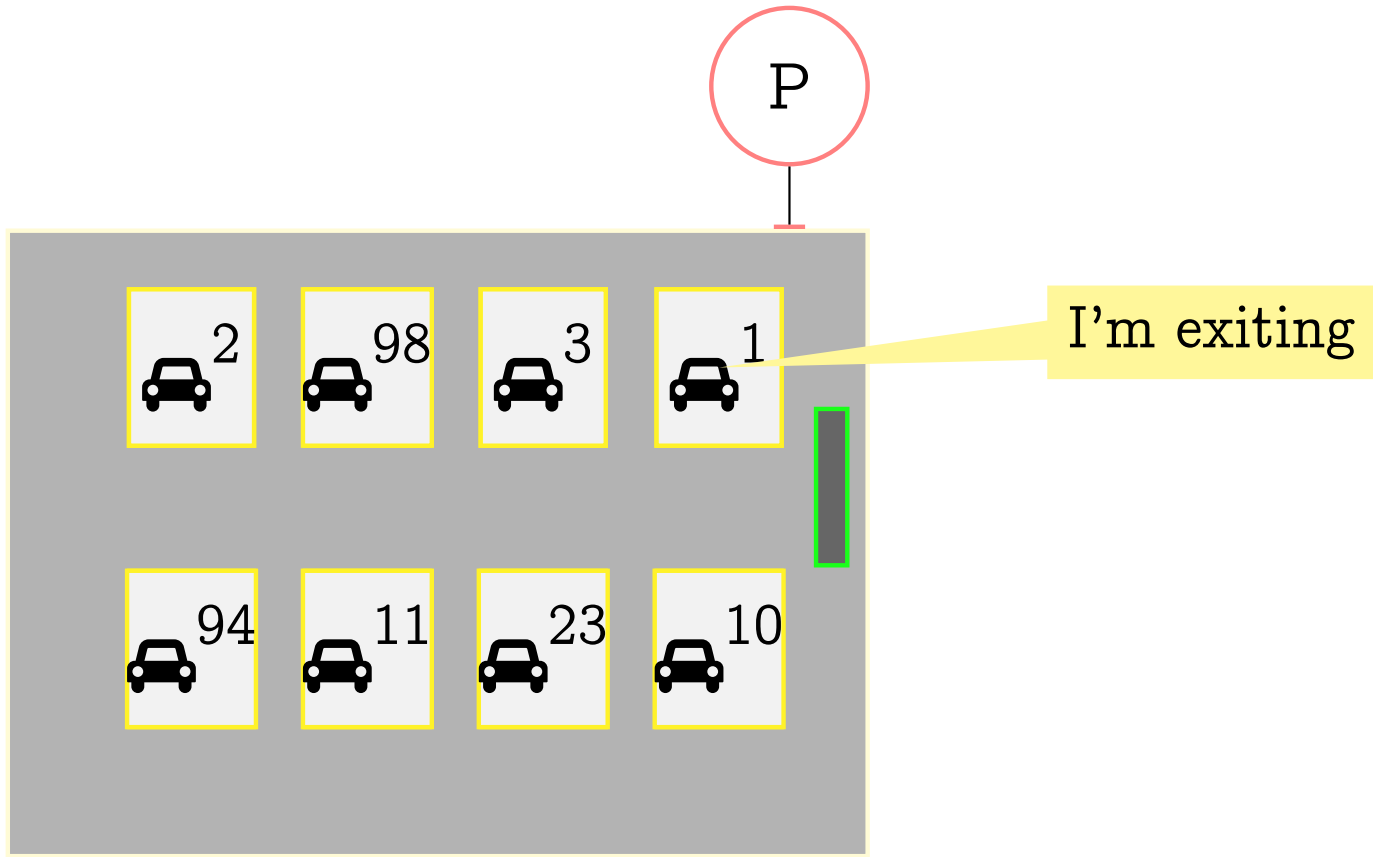
Running example



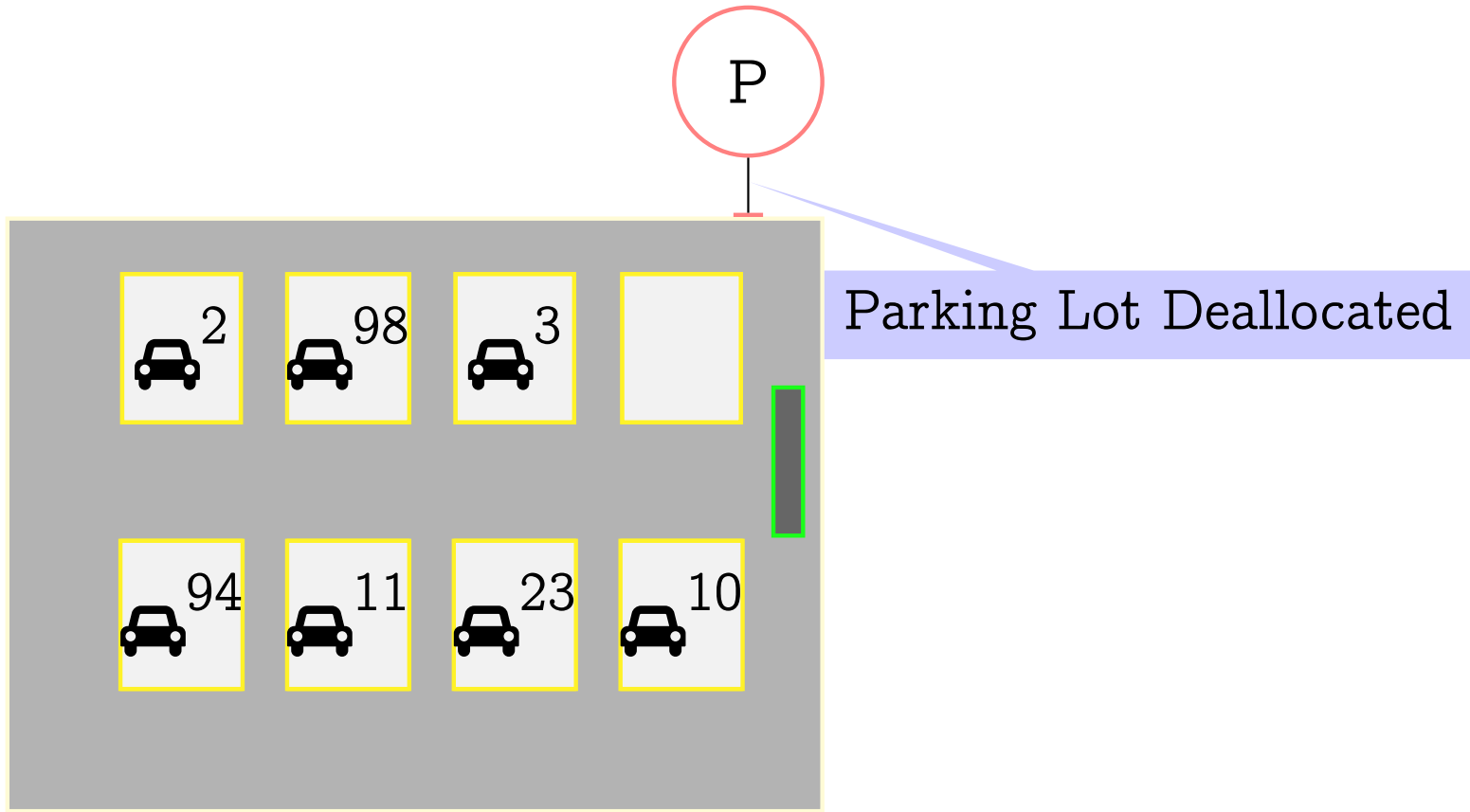
Running example



Running example

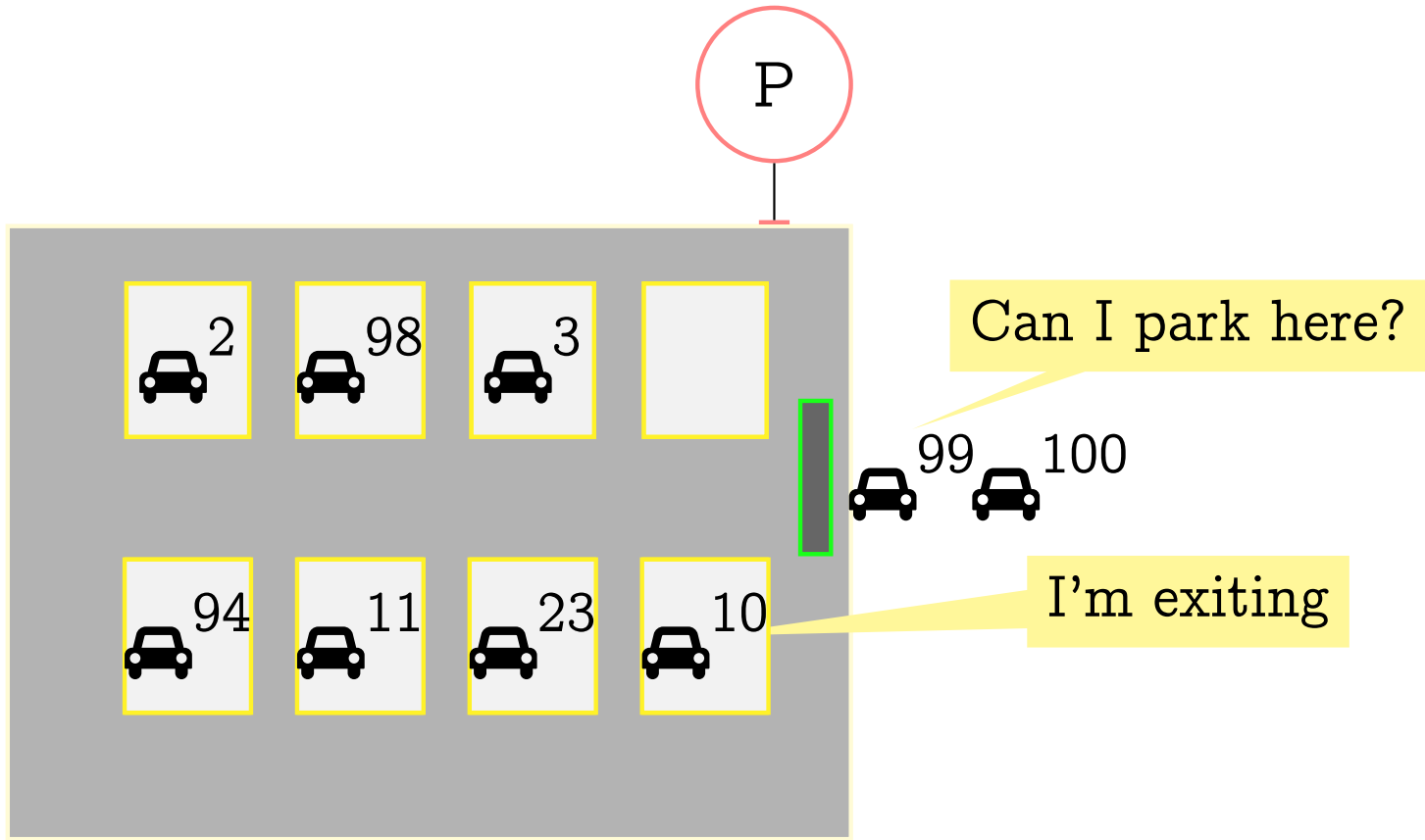


Running example

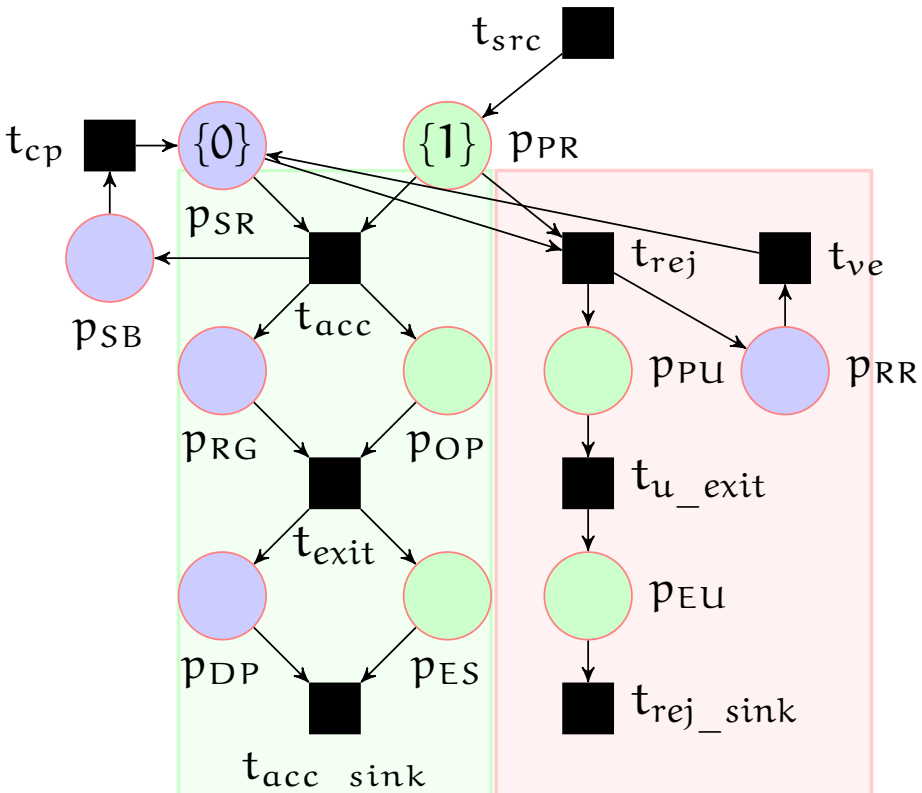


Running example

- single server-multiple client system
- unbounded distinguishable clients of the same type



Modelling an Unbounded Client-Server (UCS)



Finite sets of atomic client propositions P_c and server propositions P_s:

$$P_c = \{\text{parking\_requested}(PR),\ \text{occupy\_parking\_lot}(OP),\ \text{parking\_unavailable}(PU),\ \text{exit\_successfully}(ES),\ \text{exited\_unsuccessfully}(EU)\}$$

$$P_s = \{\text{server\_ready}(SR),\ \text{server\_busy}(SB),\ \text{request\_granted}(RG),\ \text{request\_rejected}(RR),\ \text{deallocate\_parking\_lot}(DP)\}$$

Question: What properties of the system are we interested in?

Properties

1. When a vehicle requests a parking space, it is always the case that for every vehicle, it eventually exits the system, either successfully after being granted a parking space, or unsuccessfully, when its request is denied.

Properties

1. When a vehicle requests a parking space, it is always the case that for every vehicle, it eventually exits the system, either successfully after being granted a parking space, or unsuccessfully, when its request is denied.

$$\mathrm{G}_s(\forall x)\Big(\mathit{parking\_requested}(x) \Rightarrow \mathrm{F}_c\ (\mathit{exit\_successfully}(x)\ \vee\ \mathit{exit\_unsuccessfully}(x))\Big)$$

Properties

1. When a vehicle requests a parking space, it is always the case that for every vehicle, it eventually exits the system, either successfully after being granted a parking space, or unsuccessfully, when its request is denied.

$$G_s(\forall x)\Big(parking\_requested(x) \Rightarrow F_c\ (exit\_successfully(x)\ \vee\ exit\_unsuccessfully(x))\Big)$$

2. It is always the case that if the client occupies a parking lot, it will eventually exit the parking lot.

Properties

1. When a vehicle requests a parking space, it is always the case that for every vehicle, it eventually exits the system, either successfully after being granted a parking space, or unsuccessfully, when its request is denied.

$$G_s(\forall x)\Big(parking\_requested(x) \Rightarrow F_c\ (exit\_successfully(x)\ \vee\ exit\_unsuccessfully(x))\Big)$$

2. It is always the case that if the client occupies a parking lot, it will eventually exit the parking lot.

$$G_s(\forall x)\Big(occupy\_parking\_lot(x) \Rightarrow F_c(exit\_successfully(x))\Big)$$

Properties

3. There may be clients whose requests are rejected.

Properties

3. There may be clients whose requests are rejected.

$$\mathrm{G}_s(\exists x)\big(\mathit{parking\_requested}(x) \wedge \mathrm{F}_c(\mathit{exit\_unsuccessfully}(x))\big)$$

Properties

3. There may be clients whose requests are rejected.

$$G_s(\exists x)\big(parking\_requested(x) \wedge F_c(exit\_unsuccessfully(x))\big)$$

4. There may be clients who have requested for parking and who wait in the parking unavailable state until they are able to exit the system.

Properties

3. There may be clients whose requests are rejected.

$$G_s(\exists x)\big(parking\_requested(x) \wedge F_c(exit\_unsuccessfully(x))\big)$$

4. There may be clients who have requested for parking and who wait in the parking unavailable state until they are able to exit the system.

$$G_s(\exists x)\Big(parking\_requested(x) \wedge F_c\big(parking\_unavailable(x)\ U_c\ exit\_unsuccessfully(x)\big)\Big)$$

The proposed logic: The Monodic* Logic $\mathcal{L}^1_{\mathrm{UCS}}$

The set of client formulae Δ:

$$\Delta ::= p(x) \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \wedge \beta \mid X_c\alpha \mid F_c\alpha \mid G_c\alpha \mid \alpha\ U_c\ \beta$$

where $\alpha, \beta \in \Delta$ and $p \in P_c$, the set of atomic client propositions.

The proposed logic: The Monodic* Logic $\mathcal{L}^1_{UCS}$

The set of client formulae Δ:

$$\Delta ::= p(x) \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \wedge \beta \mid X_c\alpha \mid F_c\alpha \mid G_c\alpha \mid \alpha\ U_c\ \beta$$

where $\alpha, \beta \in \Delta$ and $p \in P_c$, the set of atomic client propositions.
The set of server formulae Ψ:

$$\Psi ::= q \mid \neg\psi \mid (\exists x)\alpha \mid (\forall x)\alpha \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid X_s\psi \mid F_s\psi \mid G_s\psi \mid \psi_1\ U_s\ \psi_2$$

where $\psi, \psi_1, \psi_2 \in \Psi$ and $q \in P_s$, the set of atomic server propositions, $\alpha \in \Delta$.

The Monodic Logic $\mathcal{L}^1_{\mathrm{UCS}}$ (contd)

$$\psi = (\exists x)(\exists y)\Big(\big(\mathrm{PR}(x) \wedge \mathrm{PR}(y)\big) \wedge \mathbf{F}_c\big(\mathrm{ES}(x) \wedge \mathrm{ES}(y)\big)\Big)$$

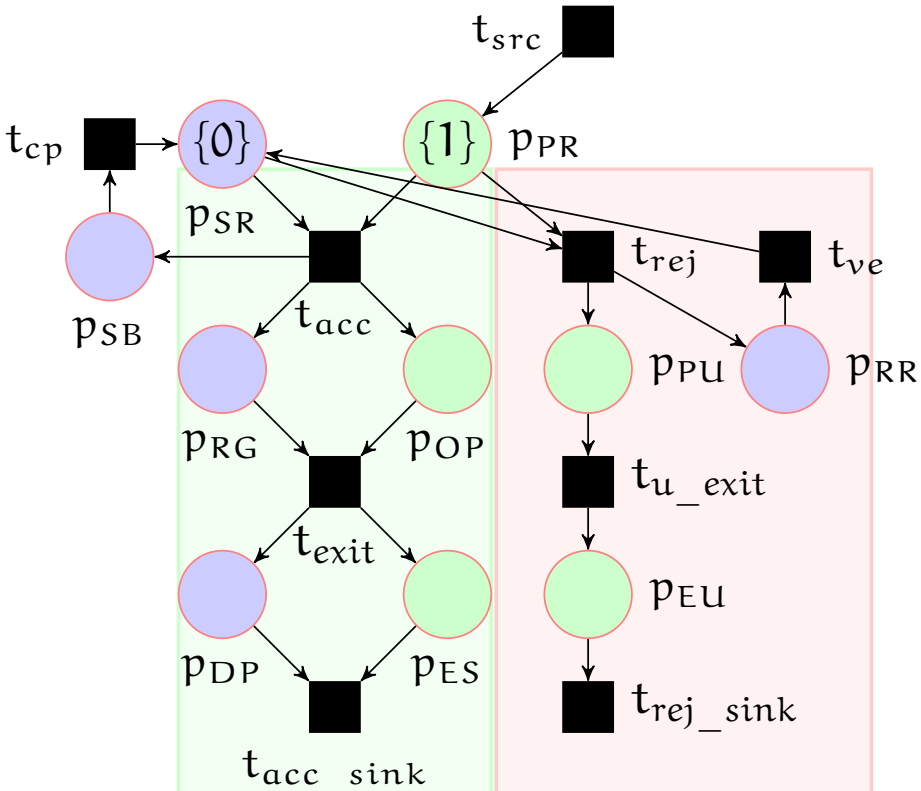
- Is $\psi \in \mathcal{L}^1_{\mathrm{UCS}}$?

The Monodic Logic $\mathcal{L}^1_{\mathrm{UCS}}$ (contd)

$$\psi = (\exists x)(\exists y)\Big(\big(\mathrm{PR}(x) \wedge \mathrm{PR}(y)\big) \wedge \mathbf{F}_c\big(\mathrm{ES}(x) \wedge \mathrm{ES}(y)\big)\Big)$$

- Is $\psi \in \mathcal{L}^1_{\mathrm{UCS}}$?
- Answer: No. **not a well formed formula in** $\mathcal{L}^1_{\mathrm{UCS}}$

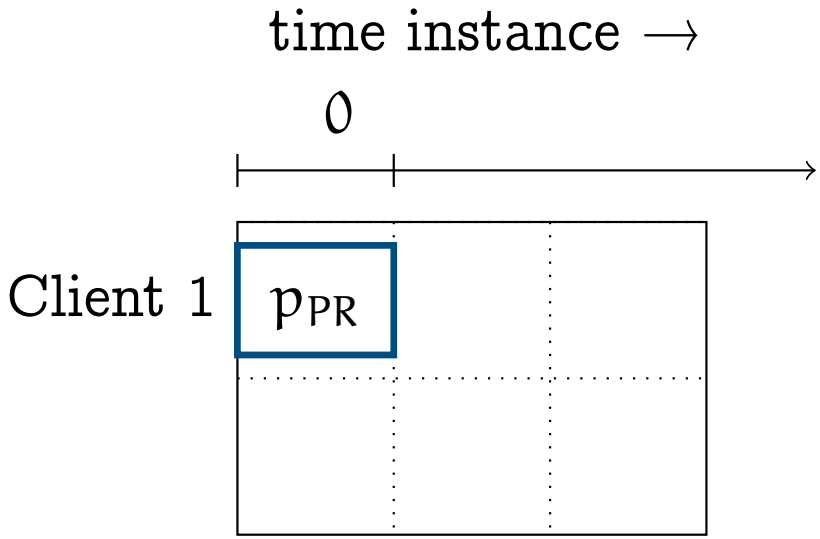
Recall: our model



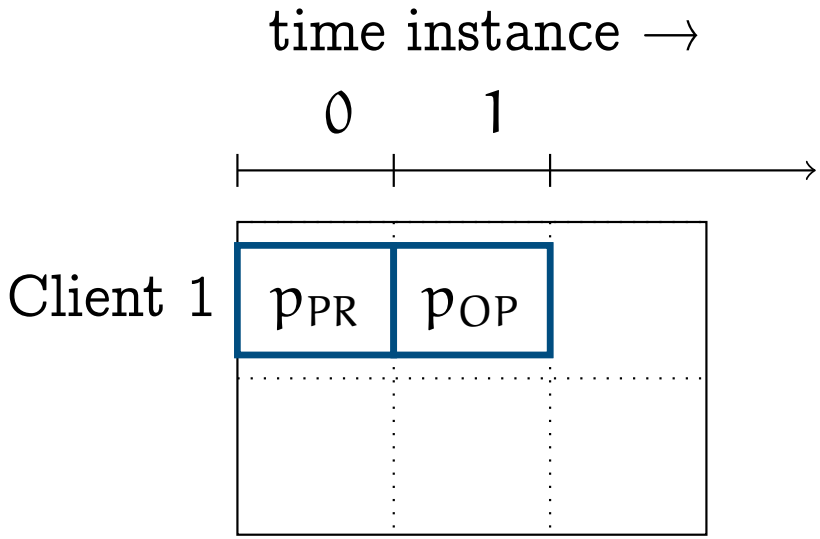
Atomic client propositions P_c and server propositions P_s:

$$P_c = \{\text{parking\_requested}(PR),$$
$$\text{occupy\_parking\_lot}(OP),$$
$$\text{parking\_unavailable}(PU),$$
$$\text{exit\_successfully}(ES),$$
$$\text{exited\_unsuccessfully}(EU)\}$$
$$P_s = \{\text{server\_ready}(SR),$$
$$\text{server\_busy}(SB),$$
$$\text{request\_granted}(RG),$$
$$\text{request\_rejected}(RR),$$
$$\text{deallocate\_parking\_lot}(DP)\}$$

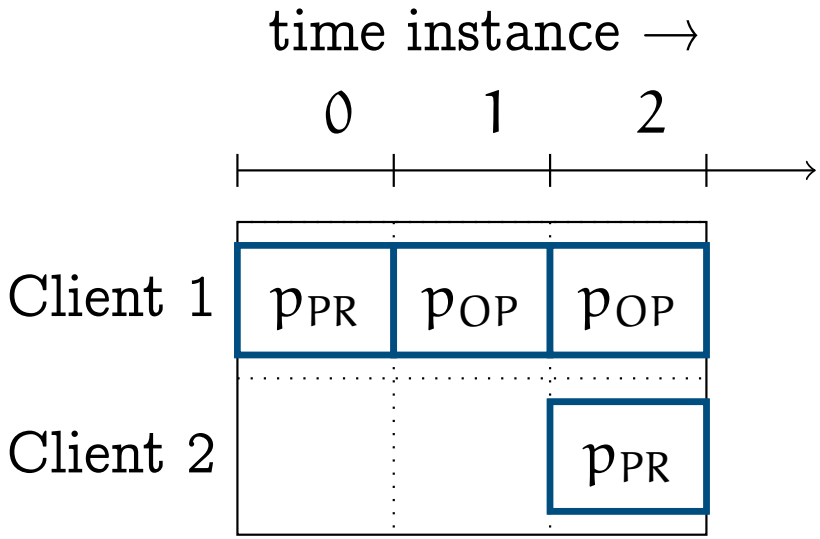
Snapshots of the system + live windows



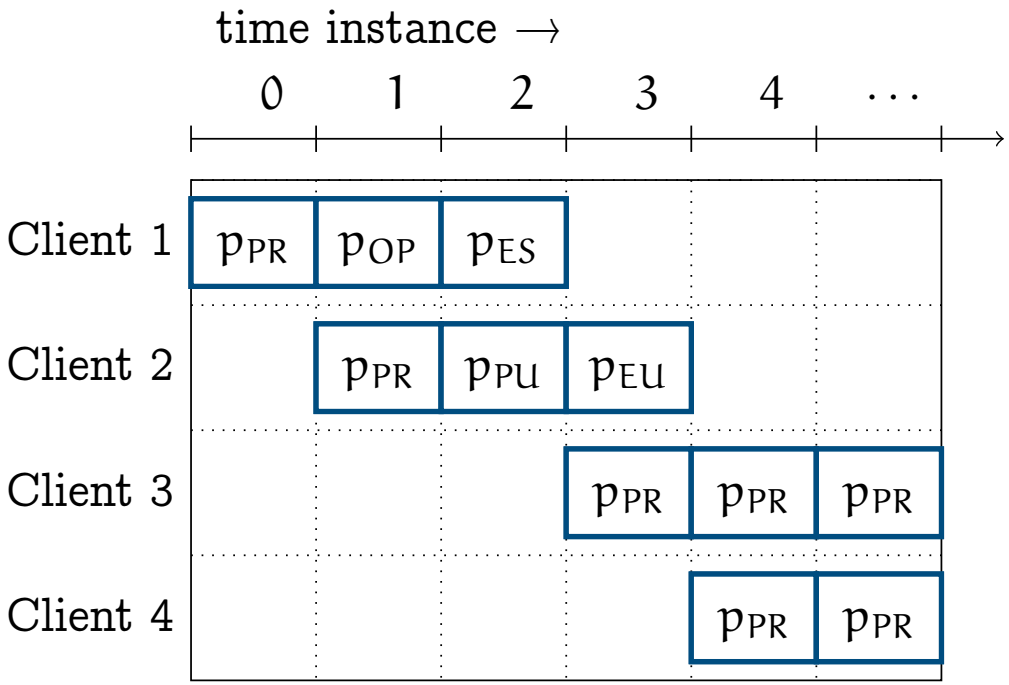
Snapshots of the system + live windows



Snapshots of the system + live windows



Snapshots of the system + live windows (contd)



- 4 distinguishable clients, with overlapping *live windows*, with the bound 5.
- client 1 is at p_{PR} at instance 0. i.e, $PR(x)$ is satisfied in client 1 at instance 0.
- For client 1, the *left boundary* is at instance 0 and its *right boundary* is at instance 2.

Formally speaking...

- Let CN be a countable set of client names.

Formally speaking...

- Let CN be a countable set of client names.
- Let $CS = (Q, \Sigma, \delta, I, F)$ be a finite state machine describing the client behaviour.

Formally speaking...

- Let CN be a countable set of client names.
- Let $CS = (Q, \Sigma, \delta, I, F)$ be a finite state machine describing the client behaviour.
- Let $L : Q \to 2^{P_c}$ be the labelling function of each client state $q \in Q$ in terms of a subset of propositions P_c true in that state.

Formally speaking...

- Let CN be a countable set of client names.
- Let $CS = (Q, \Sigma, \delta, I, F)$ be a finite state machine describing the client behaviour.
- Let $L : Q \to 2^{P_c}$ be the labelling function of each client state $q \in Q$ in terms of a subset of propositions P_c true in that state.
- Let $\mathfrak{Z} : CN \times \mathbb{N}_0 \to Q$ be a partial mapping describing the local state of each client $a \in CN$ at an instance $i \in \mathbb{N}_0$.

Formally speaking...

- Let CN be a countable set of client names.
- Let $CS = (Q, \Sigma, \delta, I, F)$ be a finite state machine describing the client behaviour.
- Let $L : Q \to 2^{P_c}$ be the labelling function of each client state $q \in Q$ in terms of a subset of propositions P_c true in that state.
- Let $\mathfrak{Z} : CN \times \mathbb{N}_0 \to Q$ be a partial mapping describing the local state of each client $a \in CN$ at an instance $i \in \mathbb{N}_0$.
- For instance, $\mathfrak{Z}(a, i) \in q$, means that the local state of each client a at instance i is state q, where $q \in Q$.

Formally speaking...

A model is a triple $M = (\nu, V, \xi)$ where

- ν gives the local behaviour of the server as follows: $\nu = \nu_0\nu_1\nu_2\ldots$, where for all $0 \leq i$, $\nu_i \subseteq P_s$

Formally speaking...

A model is a triple $M = (\nu, V, \xi)$ where

- ν gives the local behaviour of the server as follows: $\nu = \nu_0\nu_1\nu_2\ldots$, where for all $0 \leq i$, $\nu_i \subseteq P_s$
- V gives the set of live agents (clients) at each instance. $V = V_0V_1V_2\ldots$, where for all $0 \leq i$, V_i is a finite subset of CN, gives the set of live agents at the ith instance. For every $0 \leq i$, V_i and V_{i+1} satisfy the following properties:
  1. if $V_i \subseteq V_{i+1}$ then for every $a \in V_{i+1} - V_i$ such that $\mathfrak{Z}(a, i+1) \in I$.
  2. if $V_{i+1} \subseteq V_i$ then for every $a \in V_i - V_{i+1}$ such that $\mathfrak{Z}(a, i) \in F$.

Formally speaking...

A model is a triple $M = (\nu, V, \xi)$ where

- ν gives the local behaviour of the server as follows: $\nu = \nu_0\nu_1\nu_2\ldots$, where for all $0 \leq i$, $\nu_i \subseteq P_s$
- V gives the set of live agents (clients) at each instance. $V = V_0V_1V_2\ldots$, where for all $0 \leq i$, V_i is a finite subset of CN, gives the set of live agents at the ith instance. For every $0 \leq i$, V_i and V_{i+1} satisfy the following properties:
  1. if $V_i \subseteq V_{i+1}$ then for every $a \in V_{i+1} - V_i$ such that $\mathfrak{Z}(a, i+1) \in I$.
  2. if $V_{i+1} \subseteq V_i$ then for every $a \in V_i - V_{i+1}$ such that $\mathfrak{Z}(a, i) \in F$.
- $\xi = \xi_0\xi_1\xi_2\cdots$, where for all $0 \leq i$, $\xi_i : V_i \to 2^{P_c}$ gives the properties satisfied by each live agent at ith instance.

  Also, $L(\mathfrak{Z}(a, i)) = \xi_i(a)$.

Semantics

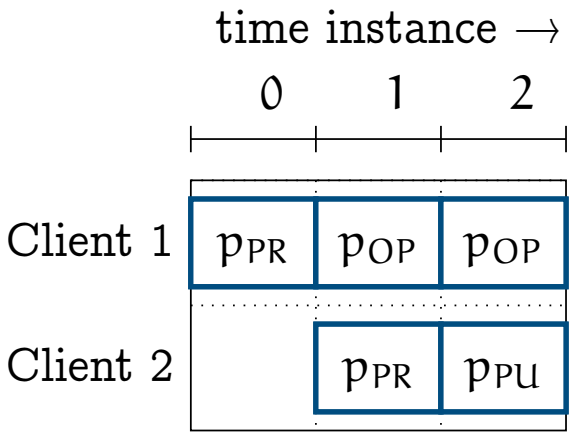
$M, i \models (\exists x)\alpha$ iff $\exists a \in CN,\ a \in V_i$ and $M, [x \mapsto a], i \models_x \alpha$.

Semantics

$M, i \models (\exists x)\alpha$ iff $\exists a \in CN$, $a \in V_i$ and $M, [x \mapsto a], i \models_x \alpha$.
$M, i \models^k (\exists x)\alpha$ iff $\exists a \in CN$, $a \in V_i$ and $M, [x \mapsto a], i \models^k_x \alpha$.

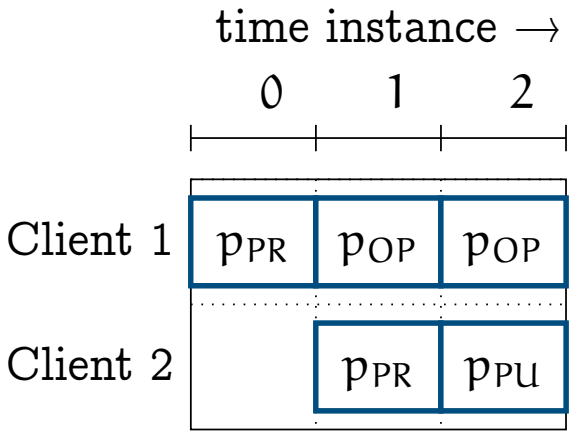
Semantics

$M, i \models (\exists x)\alpha$ iff $\exists a \in CN$, $a \in V_i$ and $M, [x \mapsto a], i \models_x \alpha$.
$M, i \models^k (\exists x)\alpha$ iff $\exists a \in CN$, $a \in V_i$ and $M, [x \mapsto a], i \models^k_x \alpha$.



Semantics

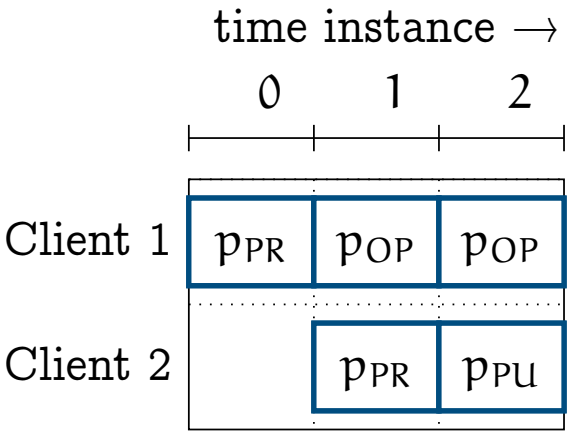
$M, i \models (\exists x)\alpha$ iff $\exists a \in CN$, $a \in V_i$ and $M, [x \mapsto a], i \models_x \alpha$.
$M, i \models^k (\exists x)\alpha$ iff $\exists a \in CN$, $a \in V_i$ and $M, [x \mapsto a], i \models^k_x \alpha$.



Given $\alpha = OP(x) \vee PR(x)$ and
$CN = \{1, 2\}$.

Semantics

$M, i \models (\exists x)\alpha$ iff $\exists a \in CN$, $a \in V_i$ and $M, [x \mapsto a], i \models_x \alpha$.
$M, i \models^k (\exists x)\alpha$ iff $\exists a \in CN$, $a \in V_i$ and $M, [x \mapsto a], i \models^k_x \alpha$.

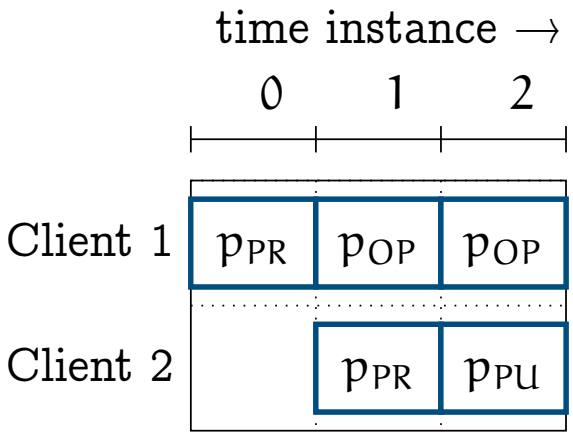


Given $\alpha = OP(x) \vee PR(x)$ and $CN = \{1, 2\}$.

- $M, 0 \models^k (\exists x)(OP(x) \vee PR(x))$.
- $M, 1 \models^k (\exists x)(OP(x) \vee PR(x))$.
- $M, 2 \models^k (\exists x)(OP(x) \vee PR(x))$.

Bounded Semantics

$M, i \models^k (\forall x)\alpha$ iff $\forall a \in CN$, if $a \in V_i$ then $M, [x \mapsto a], i \models^k_x \alpha$



Given $\alpha = ES(x)$ and $CN = \{1, 2\}$.

- $M, 0 \not\models^k (\exists x)(ES(x))$.
- $M, 1 \not\models^k (\exists x)(ES(x))$.
- $M, 2 \not\models^k (\exists x)(ES(x))$.

Bounded Semantics

$M, i \models^k F_s\psi$ iff $\exists j : i \leq j \leq k$, $M, j \models^k \psi$.
Given $\psi = (\exists x)ES(x)$.

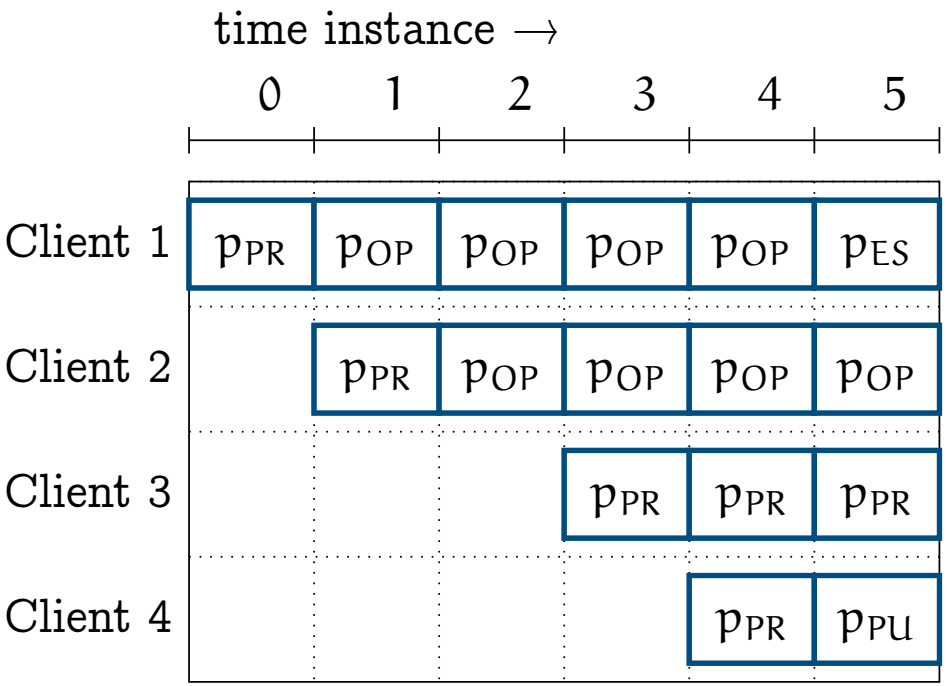


Figure: Snapshot with 4 clients

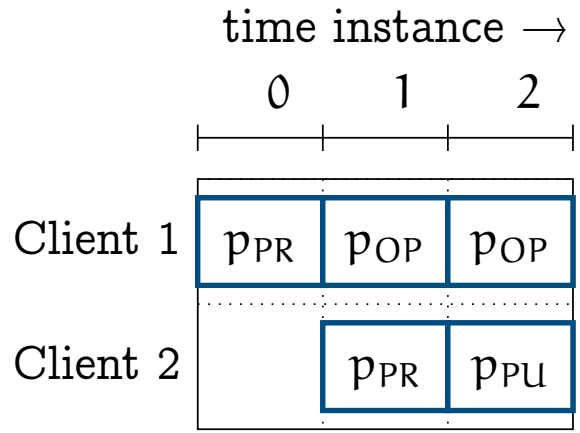


Figure: Snapshot with 2 clients

Bounded Semantics

$M, i \models^k F_s\psi$ iff $\exists j : i \leq j \leq k$, $M, j \models^k \psi$.
Given $\psi = (\exists x)ES(x)$.

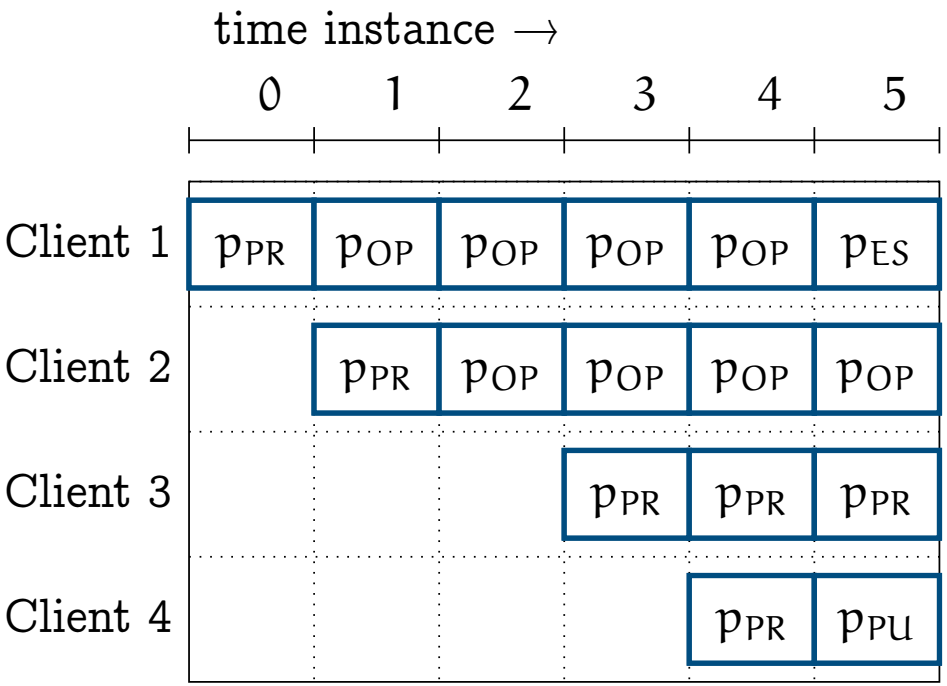


Figure: Snapshot with 4 clients

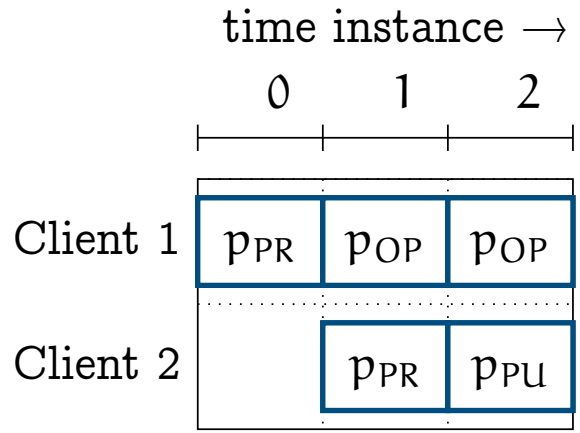


Figure: Snapshot with 2 clients

As shown above $M, 5 \models^k F_s(\exists x)ES(x)$.

Implementation

Verifying $G_s \exists x (F_c(p_0(x)))$

```
 evname: cToken_4_1
 evname: cToken_4_2
 evname: cToken_4_3
 evname: cToken_4_4
 evname: cToken_4_5
 evname: sToken_4_1
LOG: done with initializing empty token variables
LOG: sysObj.MAX_TOKENS_LIMIT before initial marking: 1
LOG: constructed initial Marking
LOG: constructed the complete Transition Function
LOG: nochange constraint added
LOG: UNSAT
LOG: UNSAT
LOG: UNSAT
LOG: UNSAT
LOG: ----------------------------run 5 ----------------------------------
 evname: cToken_5_0
 evname: cToken_5_1
 evname: cToken_5_2
 evname: cToken_5_3
 evname: cToken_5_4
 evname: cToken_5_5
 evname: sToken_5_1
LOG: done with initializing empty token variables
LOG: sysObj.MAX_TOKENS_LIMIT before initial marking: 1
LOG: constructed initial Marking
LOG: constructed the complete Transition Function
LOG: nochange constraint added
LOG: UNSAT
LOG: UNSAT
LOG: UNSAT
LOG: UNSAT
LOG: UNSAT
```

Contributions

- Part 1: Verification of Unbounded Client-Server (UCS) using Petri Nets and Linear Temporal Logic with integer arithmetic
- Part 2: Verification of UCS with distinguishable clients using ν-nets and properties specified in monodic First Order Logic

What if clients had an internal structure?

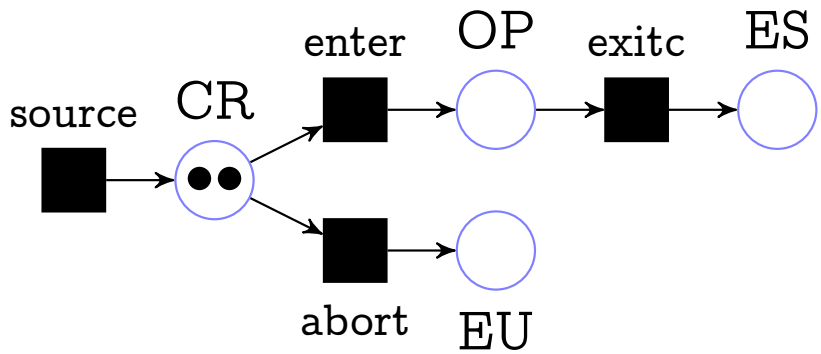


Figure: Petri net N_3 depicting client behaviour

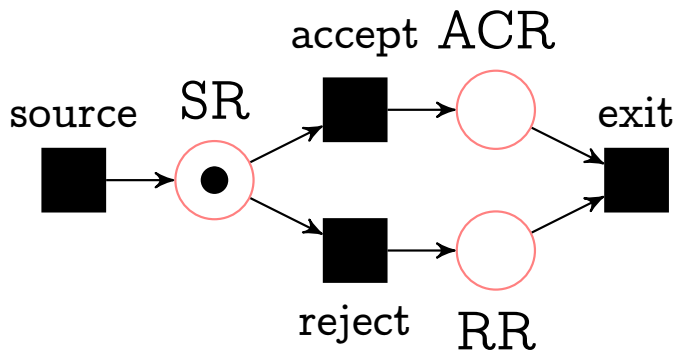


Figure: Petri net N_3 depicting server behaviour

Question: What properties of the system are we interested in?

- **Reachability:** Is $\mu_1 = \langle \mathtt{SR}, \{\!\{\mathtt{client1CR}, \mathtt{client2CR}\}\!\} \rangle$ a reachable marking?
- **Coverability:** Given $\mu_1 = \langle \mathtt{SR}, \{\!\{\mathtt{client1CR}, \mathtt{client2OP}\}\!\} \rangle$ and $\mu_2 = \langle \mathtt{SR}, \{\!\{\mathtt{client1OP}, \mathtt{client2CR}, \mathtt{client3OP}\}\!\} \rangle$.

  Then, is $\mu_1 \preceq \mu_2$?

Modelling UCS via Elementary Object Systems

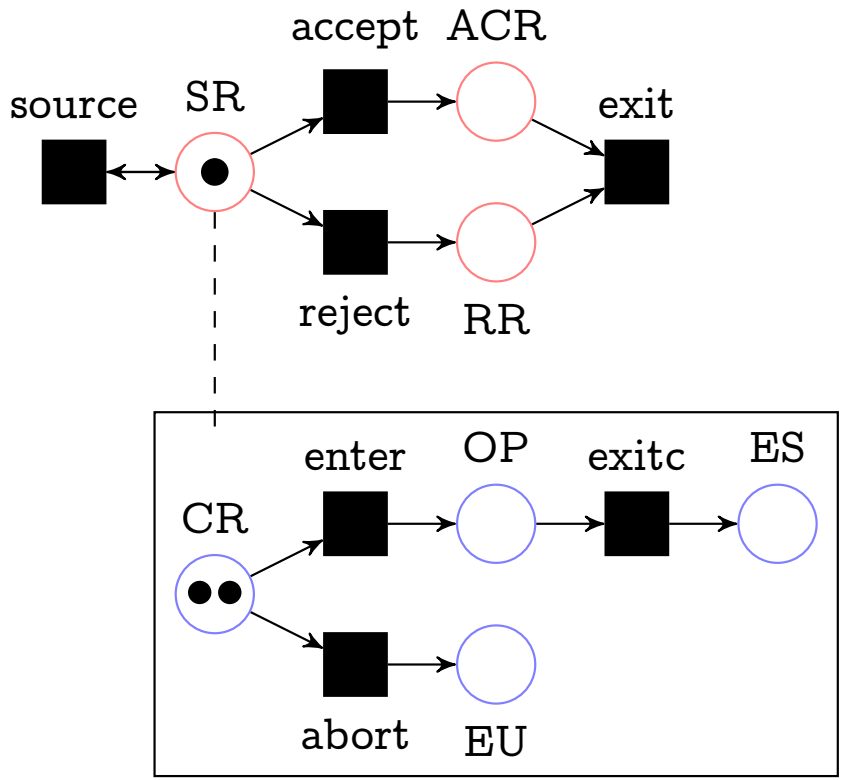
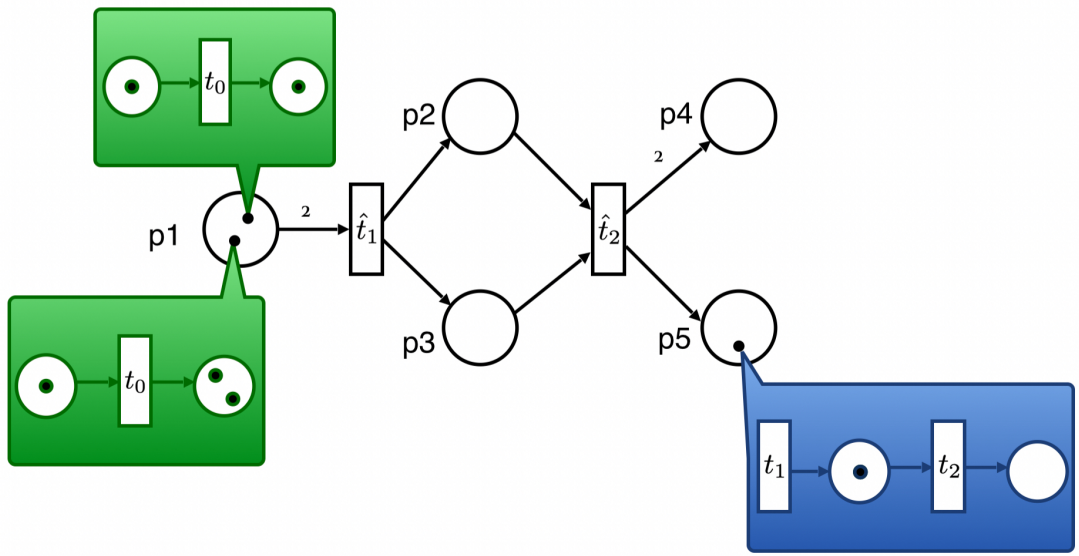


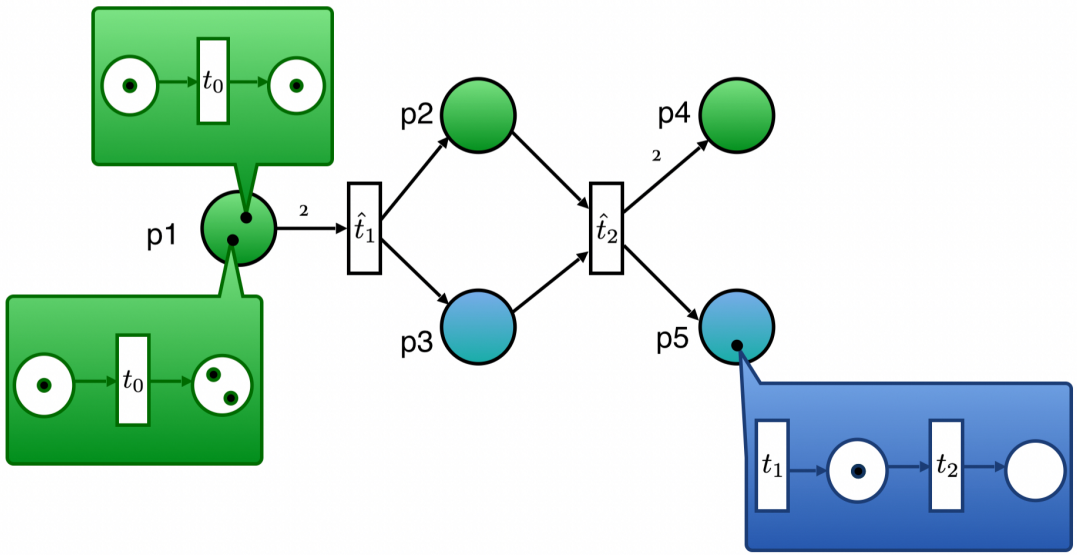
Figure: EOS modeling the single server (unbounded) multiple client system

What are EOSs?

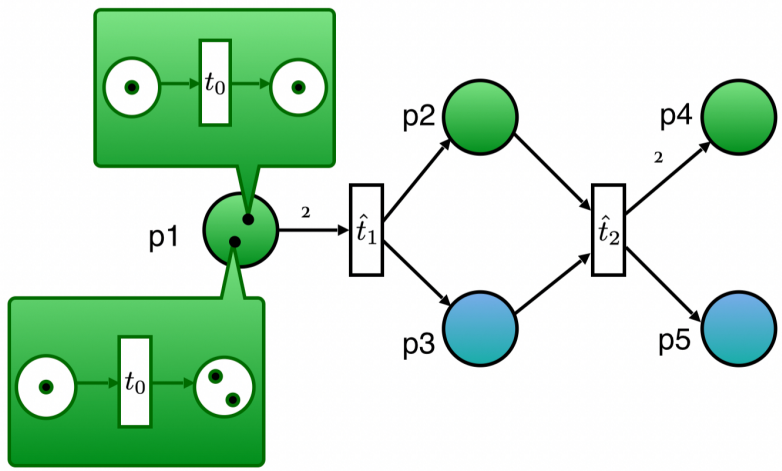
EOS – nested markings



EOS – typed places



EOS – object autonomous events



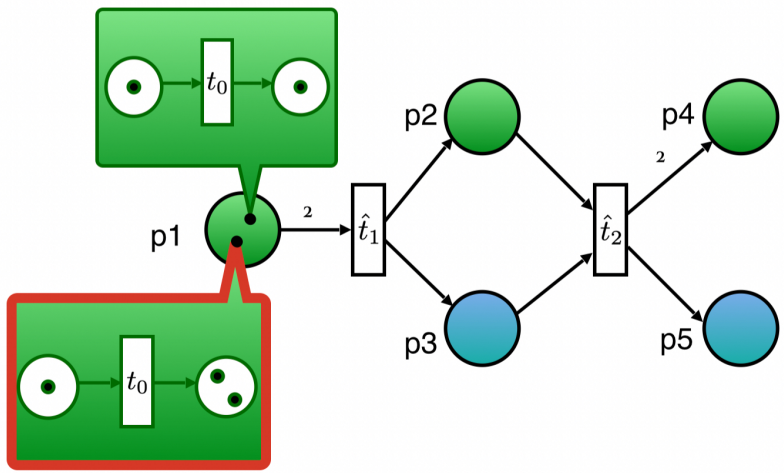
EVENTS

$e_1 = \left(id_{p_1}, t_0\right)$

$e_2 = \left(\hat{t}_1, \varnothing\right)$

$e_3 = \left(\hat{t}_2, t_0 t_1 t_1\right)$

EOS – object autonomous events



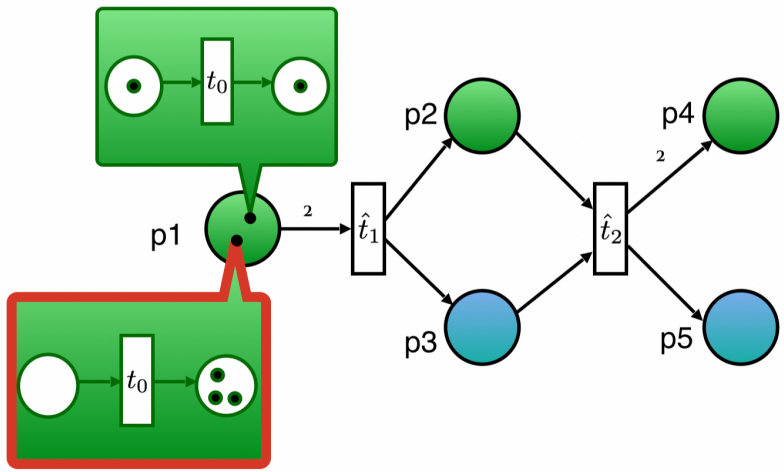
EVENTS

$e_1 = \left(id_{p_1}, t_0\right)$

$e_2 = \left(\hat{t}_1, \varnothing\right)$

$e_3 = \left(\hat{t}_2, t_0 t_1 t_1\right)$

EOS – object autonomous events



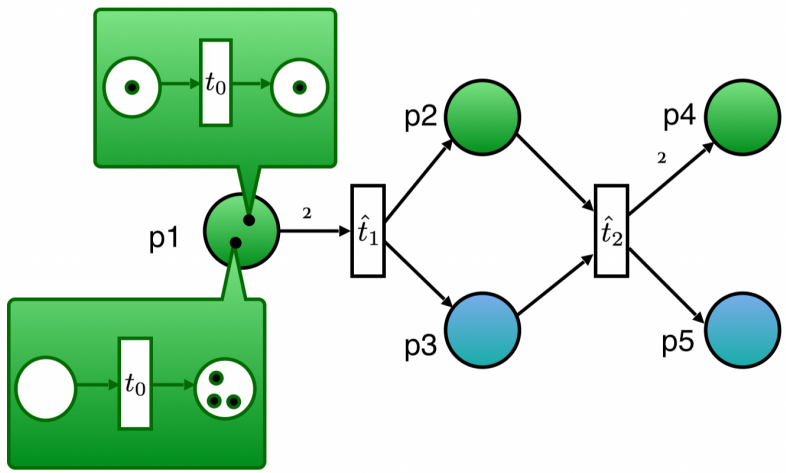
EVENTS

$$e_1 = \left(id_{p_1}, t_0\right)$$

$$e_2 = \left(\hat{t}_1, \varnothing\right)$$

$$e_3 = \left(\hat{t}_2, t_0 t_1 t_1\right)$$

EOS – system autonomous events



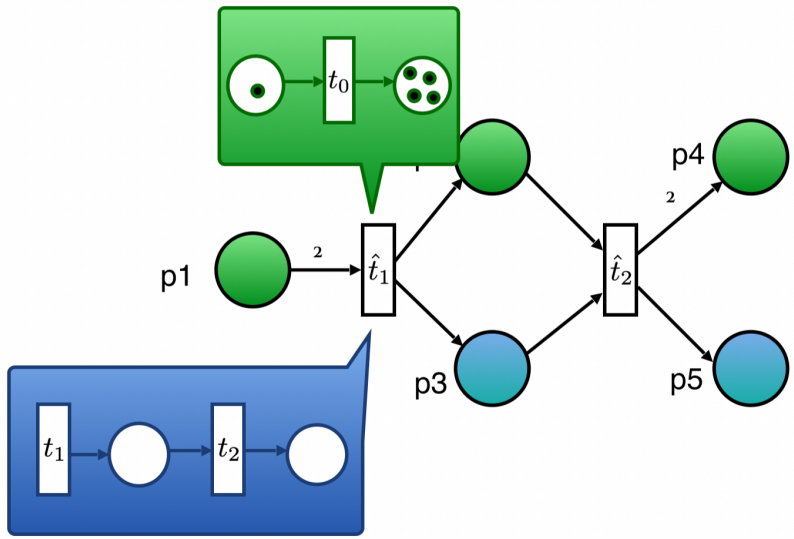
EVENTS

$e_1 = \left(id_{p_1}, t_0 \right)$

$e_2 = \left(\hat{t}_1 , \varnothing \right)$

$e_3 = \left(\hat{t}_2 , t_0 t_1 t_1 \right)$

EOS – system autonomous events



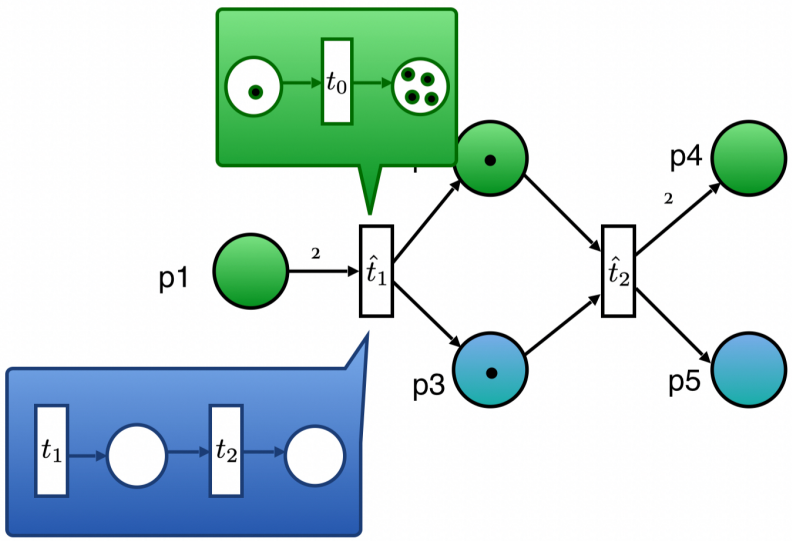
EVENTS

$e_1 = \left(id_{p_1}, t_0\right)$

$e_2 = \left(\hat{t}_1, \varnothing\right)$

$e_3 = \left(\hat{t}_2, t_0 t_1 t_1\right)$

EOS – system autonomous events



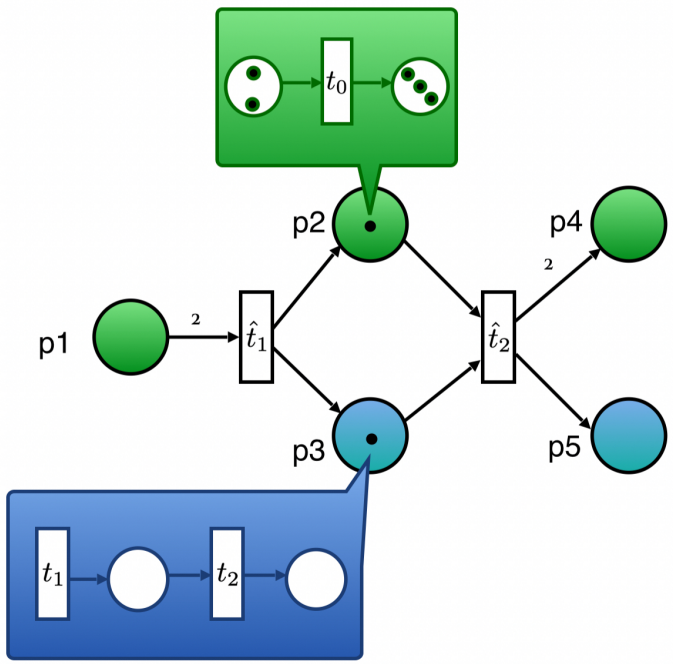
EVENTS

$e_1 = \left(id_{p_1}, t_0\right)$

$e_2 = \left(\hat{t}_1, \varnothing\right)$

$e_3 = \left(\hat{t}_2, t_0 t_1 t_1\right)$

EOS – system autonomous events



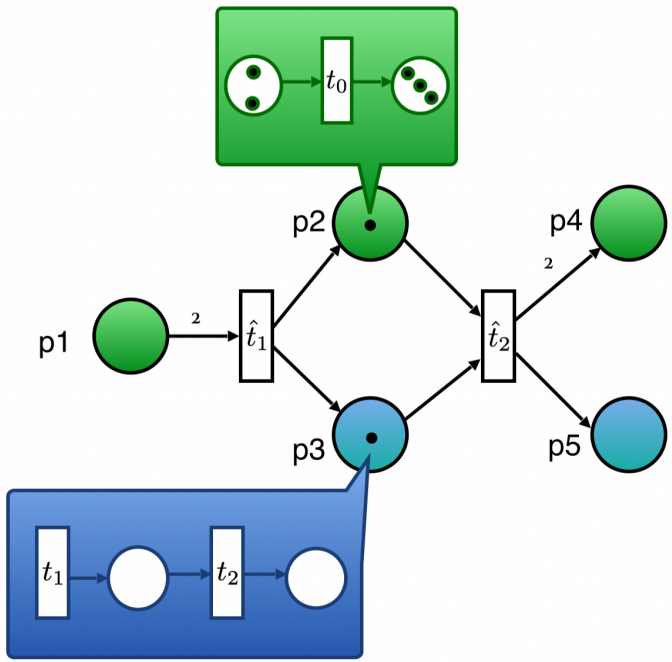
EVENTS

$e_1 = \left(id_{p_1}, t_0\right)$

$e_2 = \left(\hat{t}_1, \varnothing\right)$

$e_3 = \left(\hat{t}_2, t_0 t_1 t_1\right)$

EOS – synchronization events



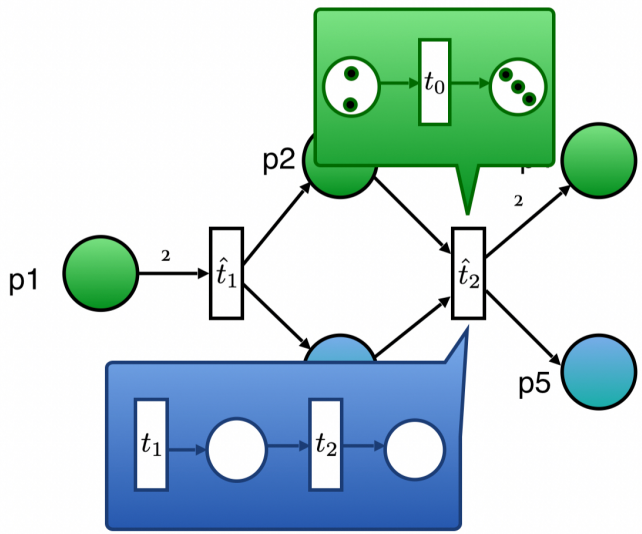
EVENTS

$e_1 = \left(id_{p_1}, t_0\right)$

$e_2 = \left(\hat{t}_1, \varnothing\right)$

$e_3 = \left(\hat{t}_2, t_0 t_1 t_1\right)$

EOS – synchronization events



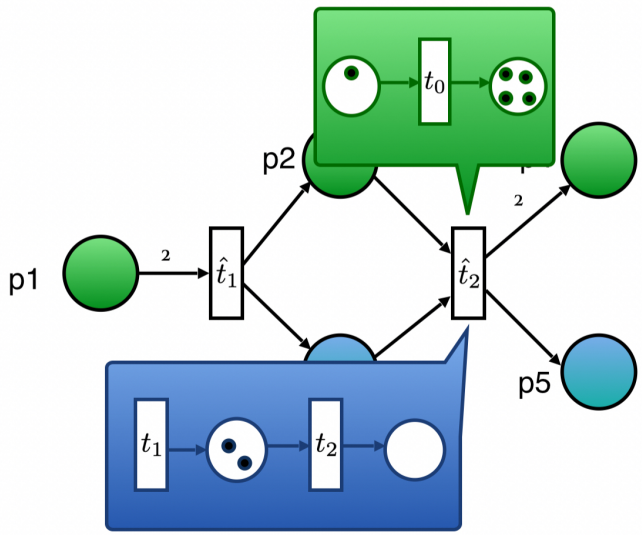
EVENTS

$e_1 = \left(id_{p_1}, t_0\right)$

$e_2 = \left(\hat{t}_1, \varnothing\right)$

$e_3 = \left(\hat{t}_2, t_0 t_1 t_1\right)$

EOS – synchronization events



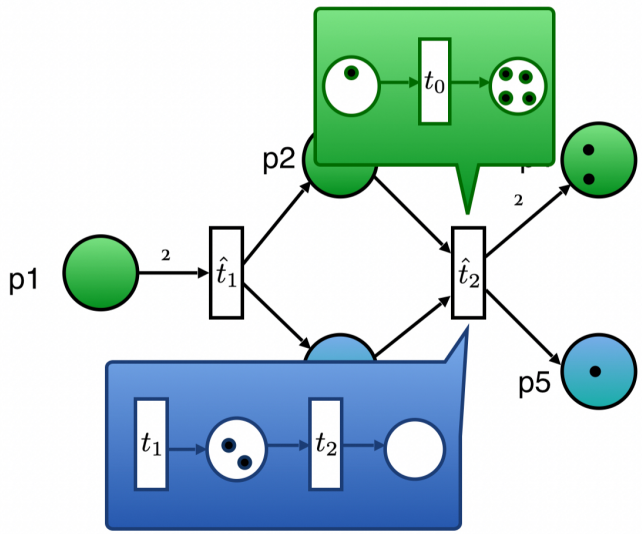
EVENTS

$e_1 = \left(id_{p_1}, t_0\right)$

$e_2 = \left(\hat{t}_1, \varnothing\right)$

$e_3 = \left(\hat{t}_2, t_0 t_1 t_1\right)$

EOS – synchronization events



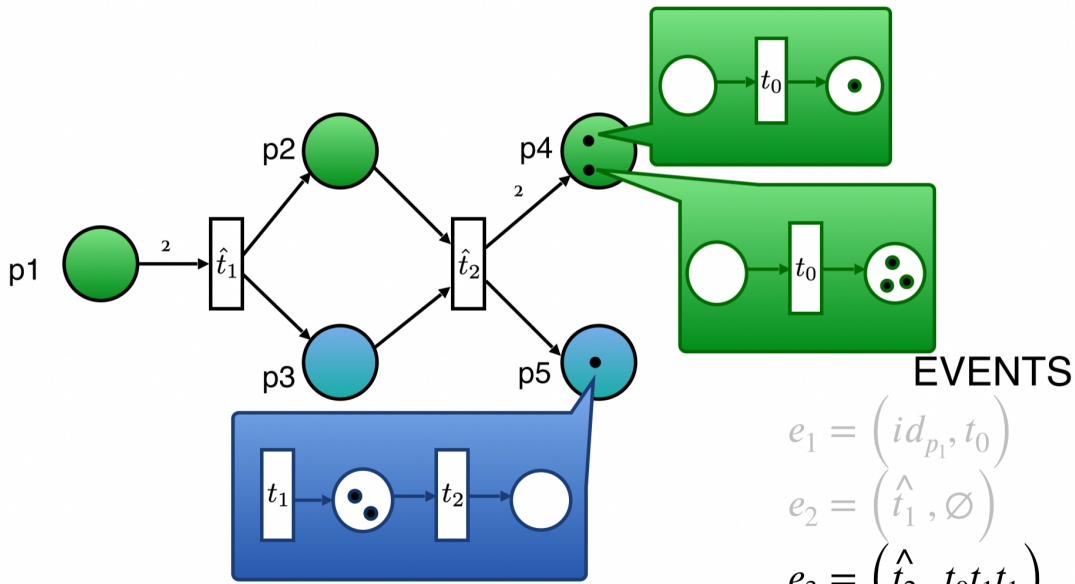
EVENTS

$e_1 = \left(id_{p_1}, t_0\right)$

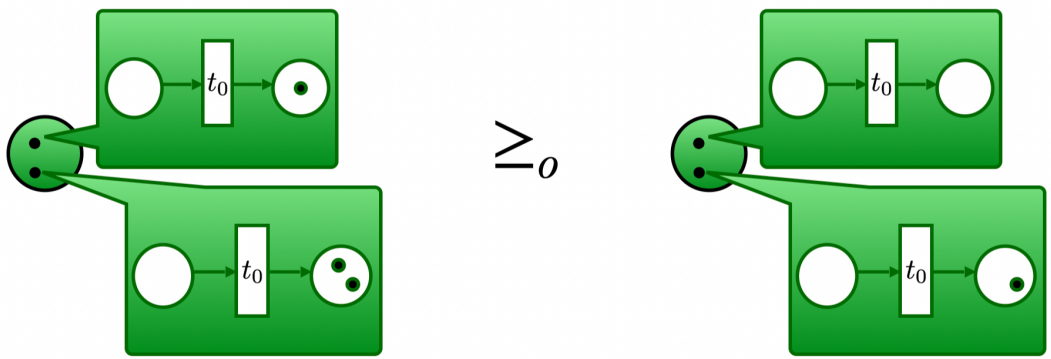
$e_2 = \left(\hat{t}_1, \varnothing\right)$

$e_3 = \left(\hat{t}_2, t_0 t_1 t_1\right)$

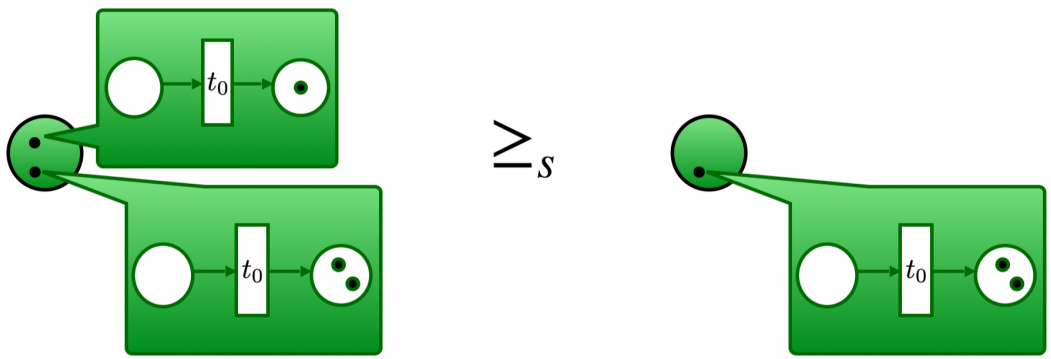
EOS – synchronization events



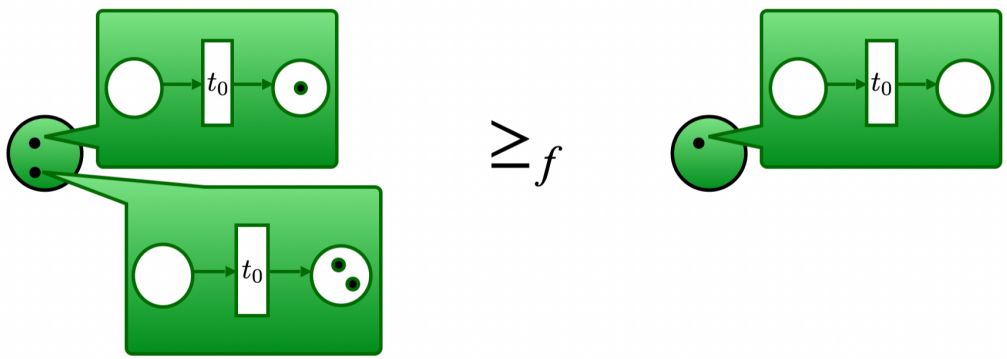
Object lossiness



System lossiness



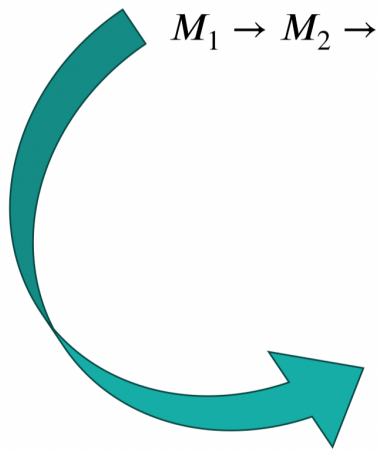
Full lossiness



$(\preccurlyeq, \ell)$-lossy runs

Perfect runs: only standard steps

$M_1 \to M_2 \to M_3 \to \dots$



$(\preccurlyeq, \ell)$-runs: at most $\ell \leq |\mathbb{N}|$ steps of type $\preccurlyeq$

$M_1 \to M_2 \succcurlyeq M'_3 \to M'_4 \succcurlyeq M'_5 \succcurlyeq M'_6 \to M_7 \to \dots$

$(\preccurlyeq, \ell)$-lossy problems

Is the system **robust** up to ℓ occurrences of $\preccurlyeq$?

$(\preccurlyeq, \ell)$-lossy problems

Is the system **robust** up to ℓ occurrences of $\preccurlyeq$?

$(\preccurlyeq, \ell)$-deadlock freeness

Input

An EOS E and an initial marking M.

Output

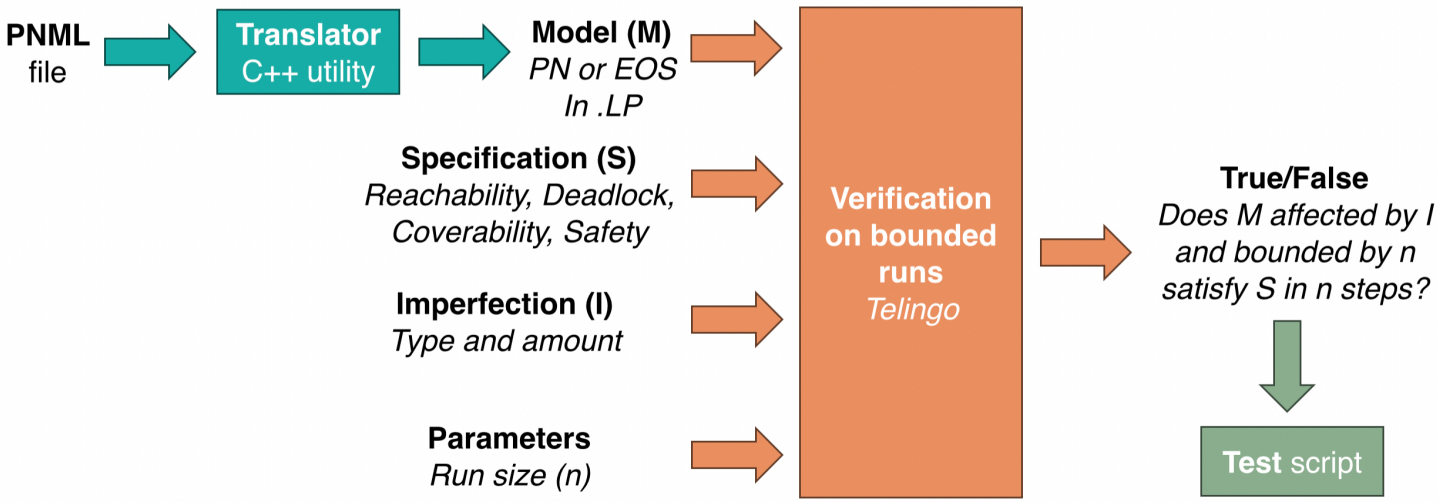
Is there a $(\preccurlyeq, \ell)$-run from M to a marking where no event is enabled?

Decidability Results (PNSE'24)

	Problems	$\preccurlyeq_f$	$\preccurlyeq_o$	$\preccurlyeq_s$
Conservative EOS	0-reach	**Undec** [Bußı4]	**Undec** [Bußı4]	**Undec** [Bußı4]
	cover	**Dec** [Bußı4]	**Undec** (2CM)	**Undec** (2CM)
	ℓ-reach/cover	**Dec** (Comp.)	**Undec** (Comp.)	**Undec** (Comp.)
	ω-reach/cover	**Dec** (Comp.)	**Undec** (Comp.)	**Undec** (Comp.)
EOS	0-reach	**Undec** [Bußı4]	**Undec** [Bußı4]	**Undec** [Bußı4]
	cover	**Undec** [Bußı4]	**Undec** (cEOS)	**Undec** (cEOS)
	ℓ-reach/cover	**Undec** (Gadget)	**Undec** (cEOS)	**Undec** (Comp.)
	ω-reach/cover	**Dec** (WSTS)	**Undec** (cEOS)	**Undec** (Comp.)

[Bußı4] Köhler-Bußmeier, Michael. 'A Survey of Decidability Results for Elementary Object Systems'. 1 Jan. 2014 : 99 – 123.

Prototype (CILC'24)



Why bounded verification?

Problems on general EOS	$\preccurlyeq_f$	$\preccurlyeq_o$	$\preccurlyeq_s$
0-reach	U	U	U
0-cover	U	U	U
ℓ -reach/cover	U	U	U
ω-reach/cover	D	U	U

F. Di Cosmo, S. Mal, T. Prince, *Deciding Reachability and Coverability in Lossy EOS*, PNSE'24

Why Telingo?

Telingo is **declarative** and supports **temporal constraints**

- E.g., `:- &tel(>? (lossy >(>? lossy)` allows at most one lossy step
- The meaning of lossy is declared orthogonally to EOS specification

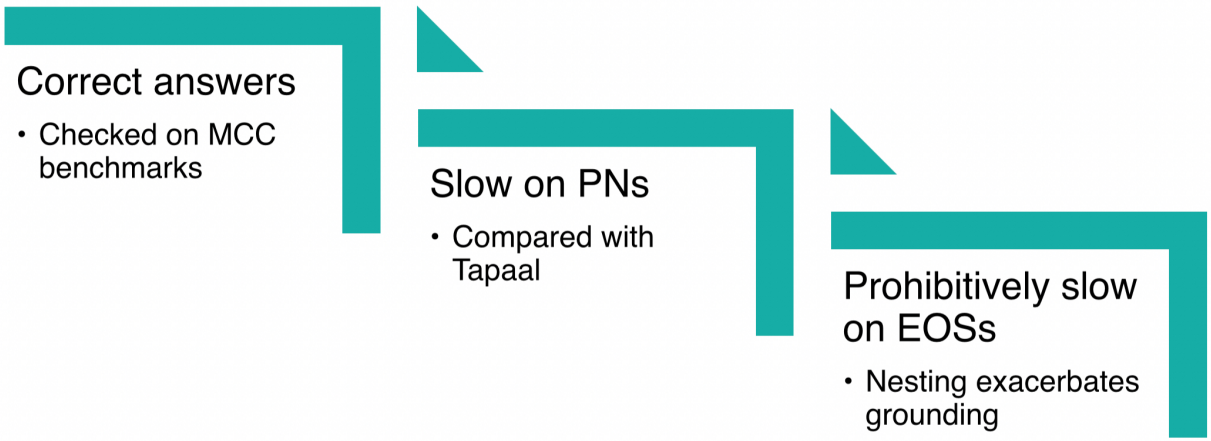
Telingo **returns finite runs**

- Perfectly matches bounded verification

Encoding of PNs and EOSs is **elegant in ASP**

- E.g., when compared to SMT – R. Phawade, T. Prince, S. Sheerazuddin et al., *Bounded Model Checking for Unbounded Client Server Systems,* Arxiv (2022)

Correctness and performances



problem	lossiness	−imax =5 (s)	−imax =10 (s)	−imax =20 (s)	TAPAAL (s)
deadlock	none	UNSAT in 0.052	SAT in 0.622	SAT in 82.754	SAT in $5e-6$
deadlock	any	SAT in 0.009	SAT in 0.010	SAT in 0.009	NA
1-safeness	none	UNSAT in 0.010	UNSAT in 0.014	UNSAT in 0.027	UNSAT in 0
1-safeness	any	UNSAT in 0.013	UNSAT in 0.018	UNSAT in 0.032	NA

Table 1
Comparative Results with TAPAAL for the Eratosthenes-PT-010 PN from the MCC benchmarks [12].

Contributions

- Part 1: Verification of Unbounded Client-Server (UCS) using Petri Nets and Linear Temporal Logic with integer arithmetic
- Part 2: Verification of UCS with distinguishable clients using ν-nets and properties specified in monodic First Order Logic
- Part 3: Verification of UCS where the clients have internal structure using Elementary Object Systems and properties specified in Linear Temporal Logic

List of publications:

1. Ramchandra Phawade, Tephilla Prince, S. Sheerazuddin: Bounded Model Checking for Unbounded Client Server Systems. Arxiv (2022)
2. Tephilla Prince: On Verifying Unbounded Client-Server Systems. Springer EUMAS 2023: 465-471
3. Francesco Di Cosmo, Tephilla Prince: Bounded Verification of Petri Nets and EOSs using Telingo: An Experience Report. CILC 2024
4. Francesco Di Cosmo, Soumodev Mal, Tephilla Prince: Deciding Reachability and Coverability in Lossy EOS. PNSE@Petri Nets 2024: 74-95

List of publications (ongoing):

5 Francesco Di Cosmo, Soumodev Mal, Tephilla Prince: Decidability and Complexity of Reachability and Coverability in Lossy EOS (Under review at TopNOC)

6 Ramchandra Phawade, Tephilla Prince, S. Sheerazuddin: Bounded Model Checking for Unbounded Client Server Systems. (Yet to be submitted to TopNOC)

7 Ramchandra Phawade, Tephilla Prince, S. Sheerazuddin: A Monodic Logic for Verifying Unbounded Client-Server Systems (yet to be submitted)

Appendix

Bounded Semantics

$M, [x \mapsto a], i \models^k_x F_c \alpha$ iff
$\exists j : i \leq j \leq k$, $a \in V_j$ and $M, [x \mapsto a], j \models^k_x \alpha$.

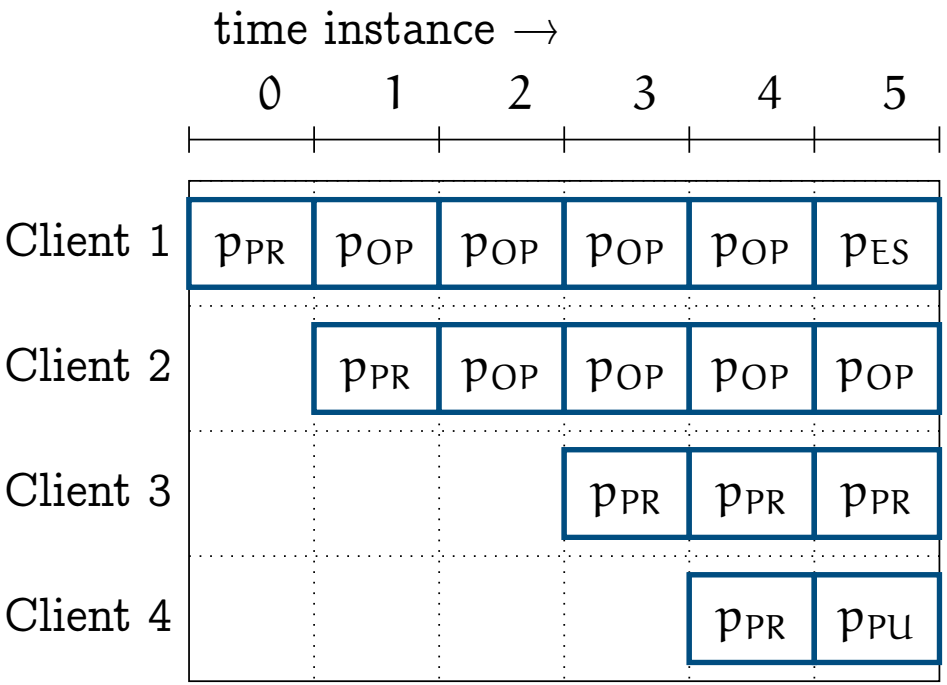


Figure: Snapshot with 2 clients

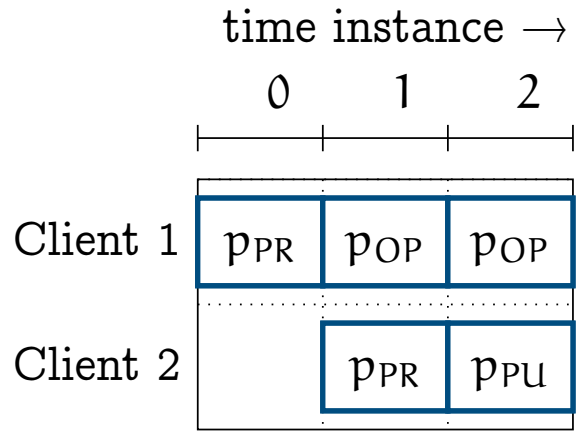


Figure: Snapshot with 4 clients

Given $\alpha = \mathrm{PR}(x)$. The above formula is satisfiable for all clients in both figures.