Two Dimensional Bounded Model Checking: A Novel Verification Strategy

Tephilla Prince
IIT Dharwad

An Ongoing work
with Prof. Ramchandra Phawade and Prof. S. Sheerazuddin

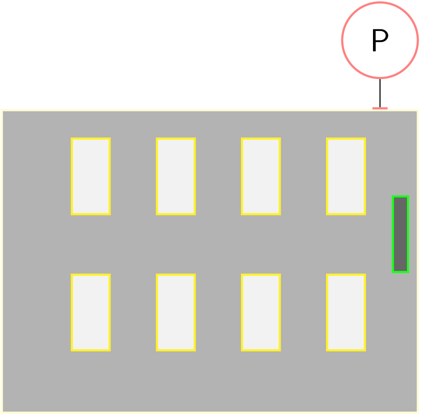
05/07/2022

Contributions

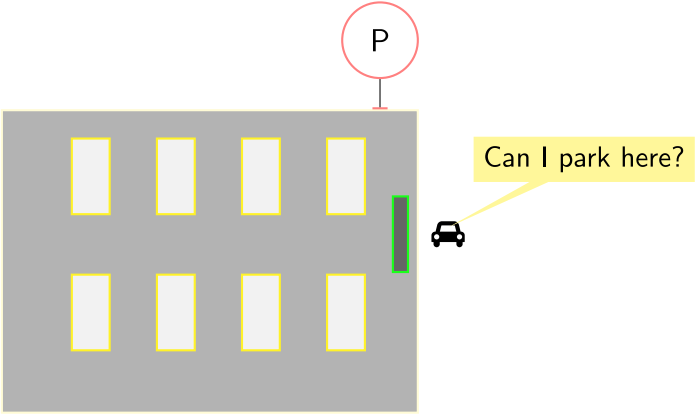
- A novel technique $2D$-BMC to verify the properties of the system.
- A counting logic language $\mathcal{L}_C$ for describing the counting and temporal properties of the system.
- Our tool DCModelChecker that uses $2D$-BMC and $\mathcal{L}_C$.

A motivating example

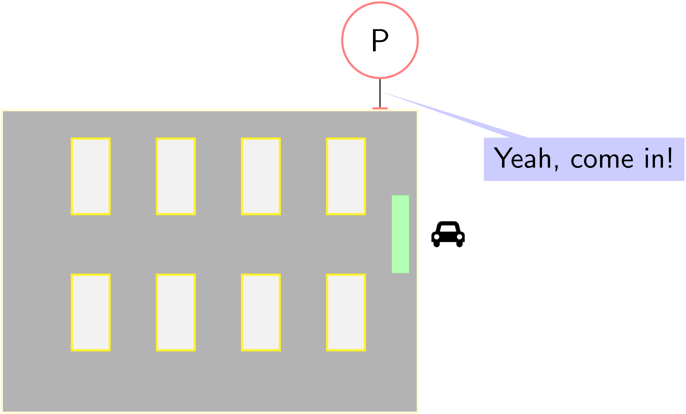
An empty Parking Lot waits for vehicles



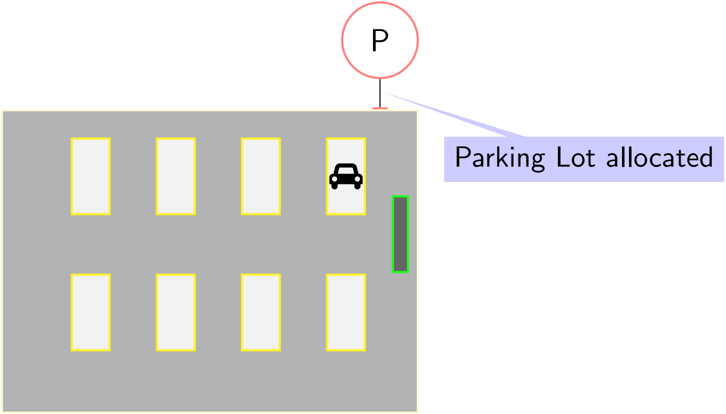
A motivating example



A motivating example

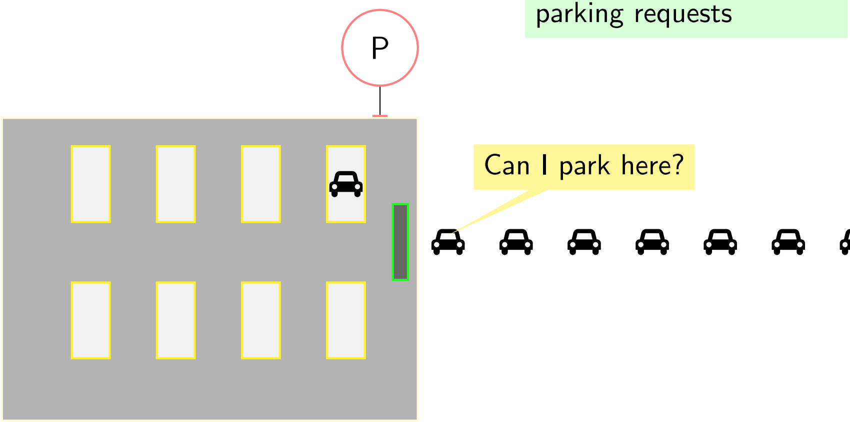


A motivating example



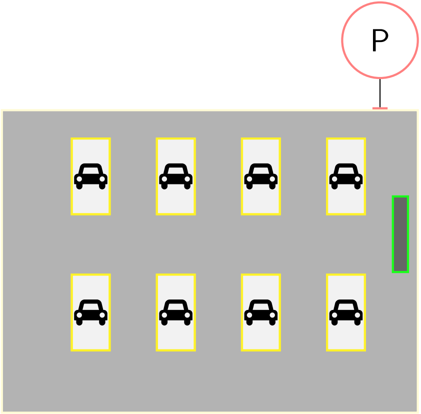
A motivating example

Unbounded number of parking requests

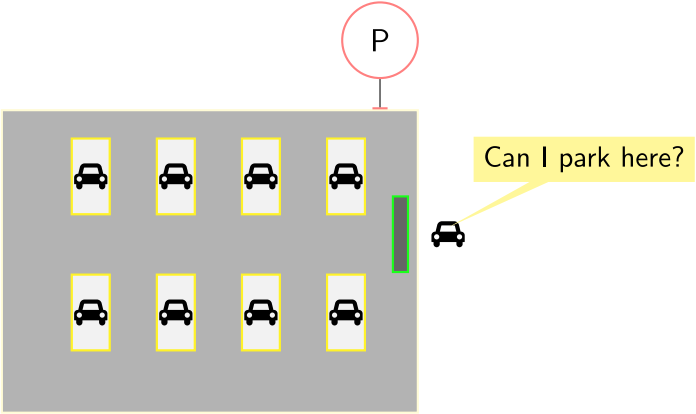


A motivating example

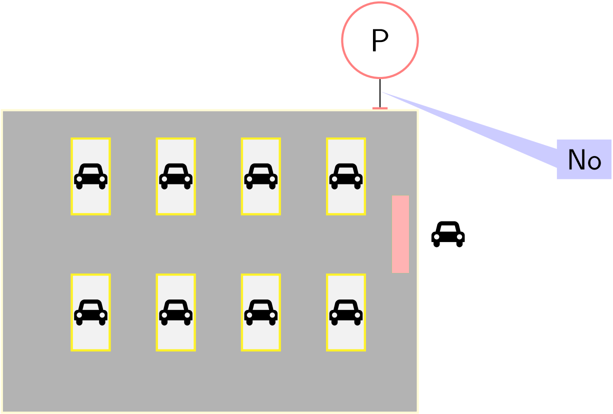
No parking space



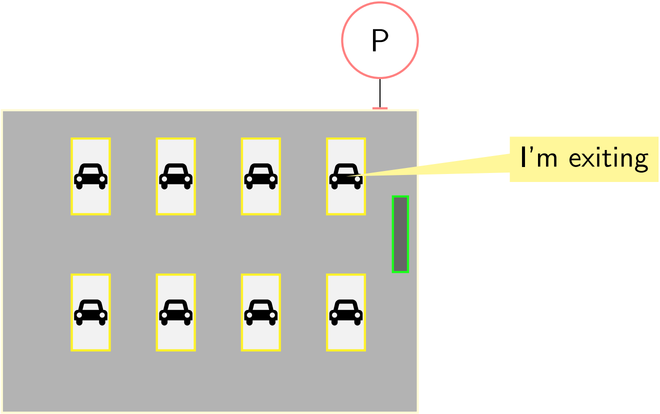
A motivating example



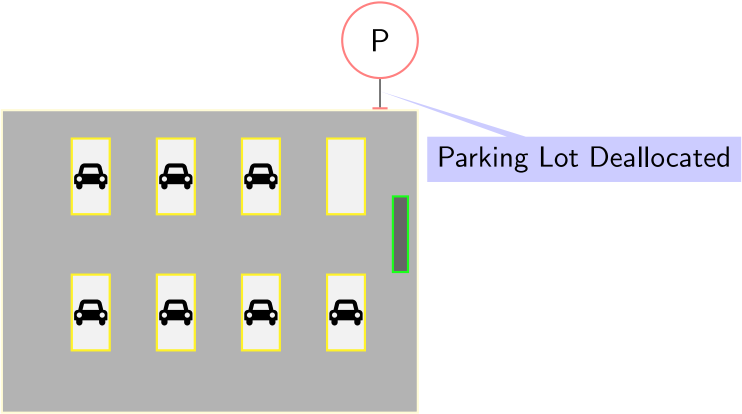
A motivating example



A motivating example

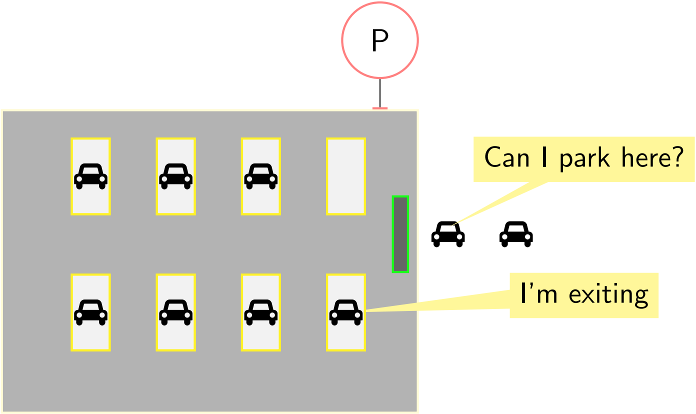


A motivating example

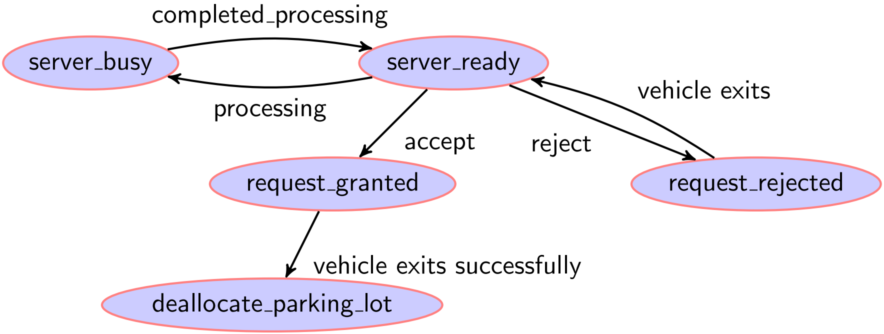


A motivating example

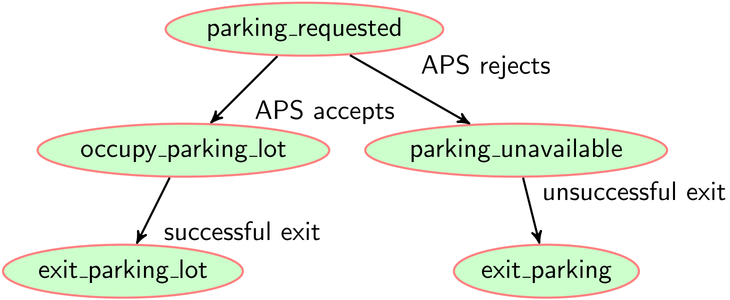
single server-multiple client system (clients of the same type)



State diagram of Autonomous Parking System (APS)

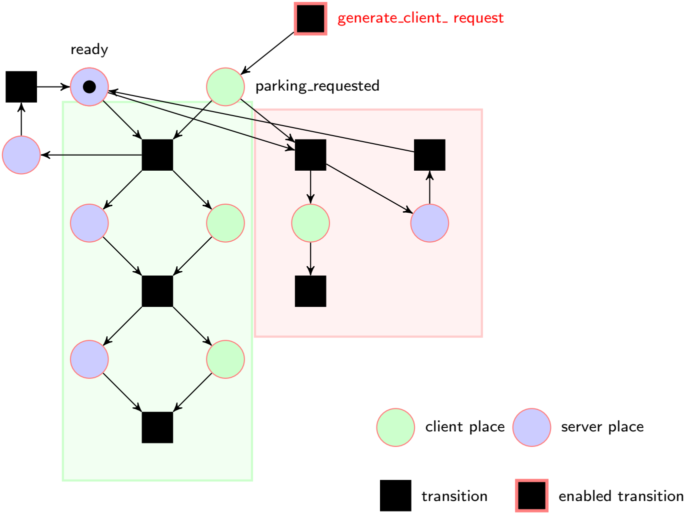


State diagram of vehicle in APS



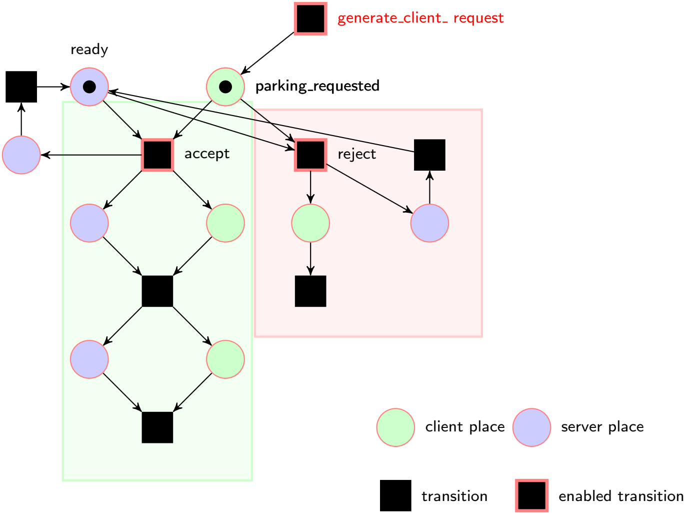
A Petri Net for APS

Initially the server is ready for client requests



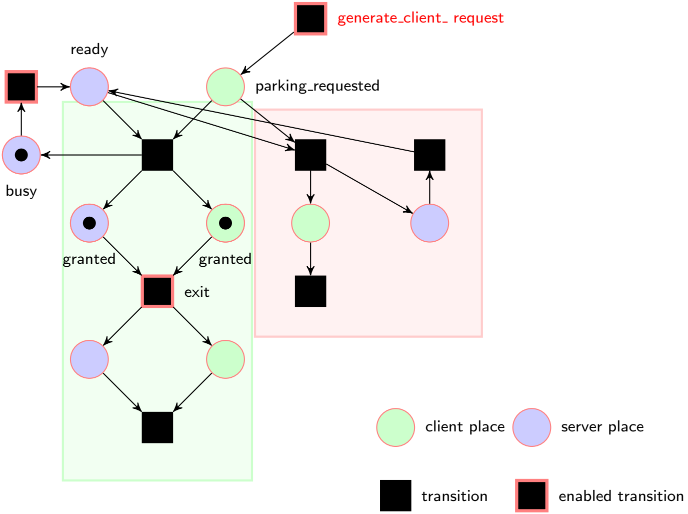
A Petri Net for APS

Client parking request is received



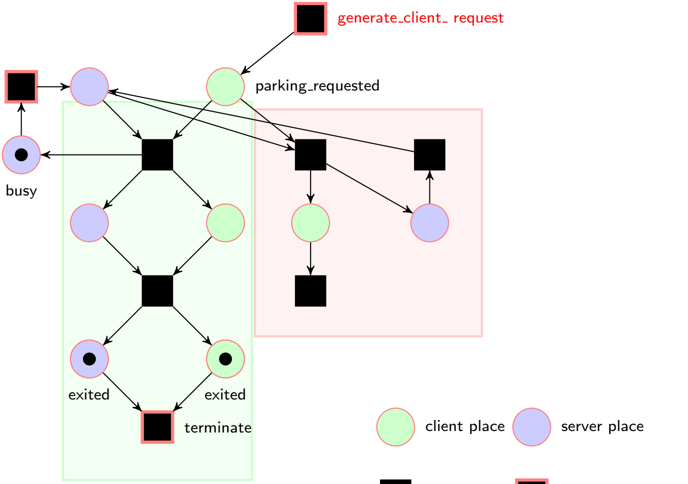
A Petri Net for APS

Server accepts the Client parking request



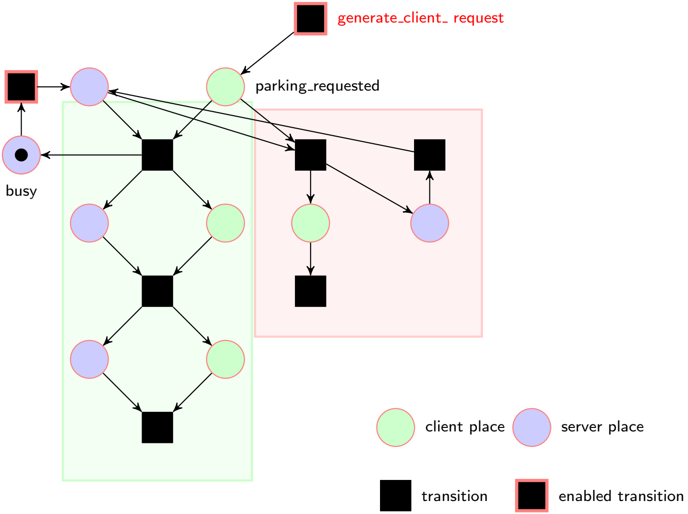
A Petri Net for APS

Client exits the parking space



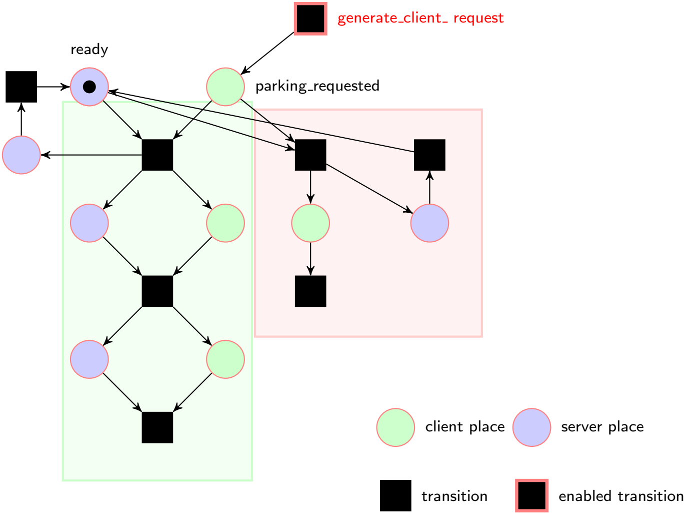
A Petri Net for APS

Client is terminated



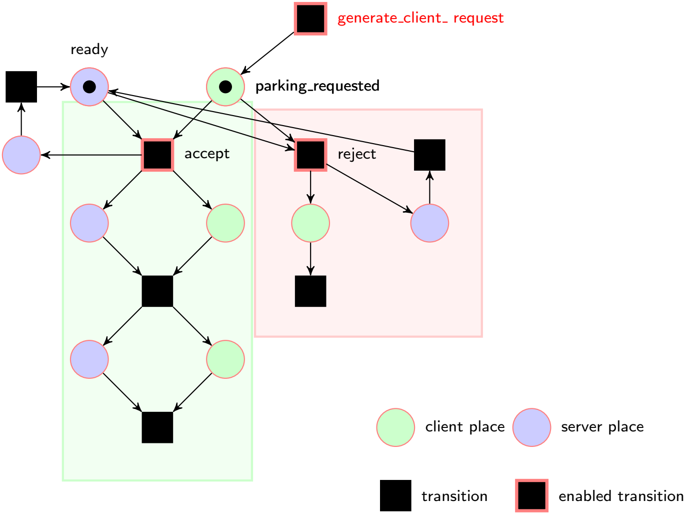
A Petri Net for APS

Server is ready for more requests



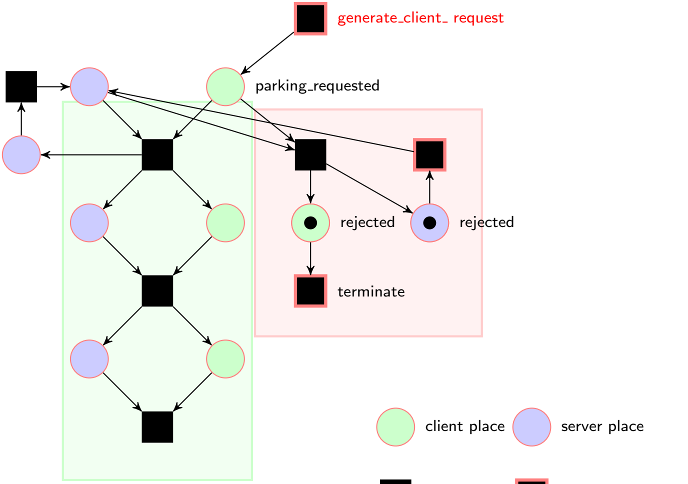
A Petri Net for APS

A new Client parking request is received



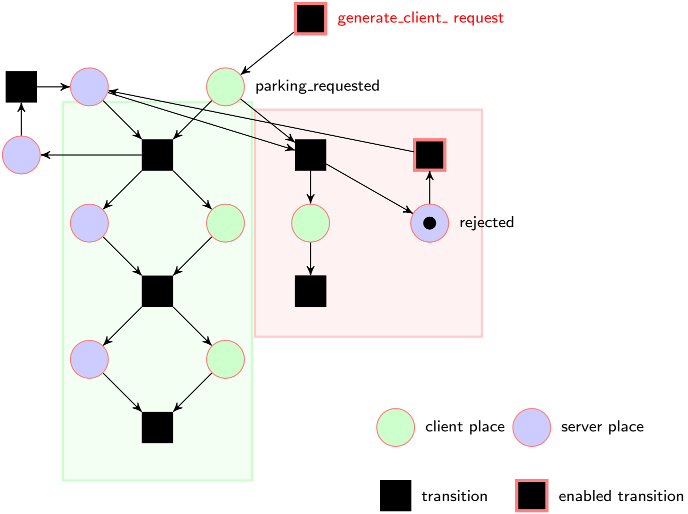
A Petri Net for APS

Server rejects the Client parking request

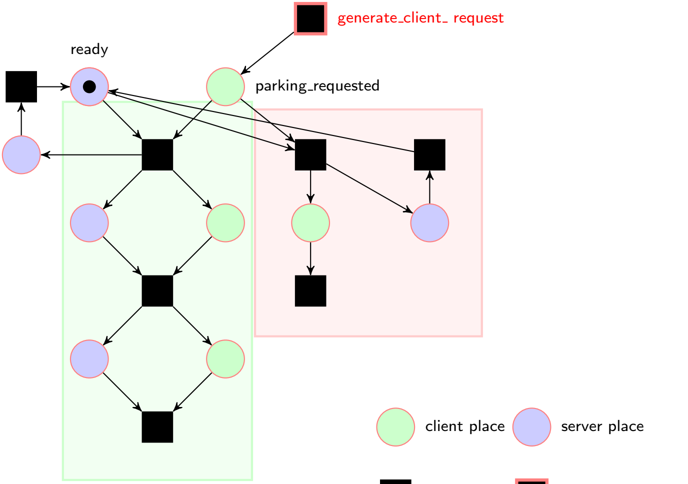


A Petri Net for APS

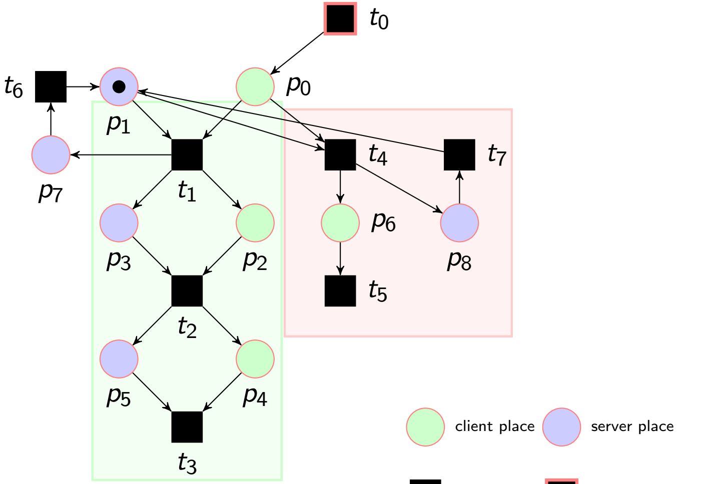
Client is terminated unsuccessfully

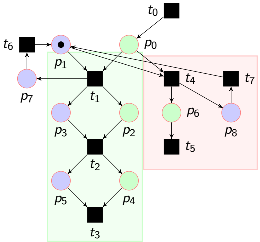


A Petri Net for APS



A Petri Net for APS

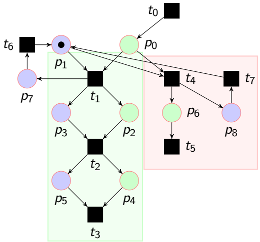




- Is the number of clients occupying parking lots = number of requests granted always?

  $G((\#x)p2(x) <= p3(x) \& \neg((\#x)p3(x) > p2(x)))$

  There is no counterexample for this property for upto bound 50



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

  $G((\#x)p2(x) <= p3(x) \& \neg((\#x)p3(x) > p2(x)))$

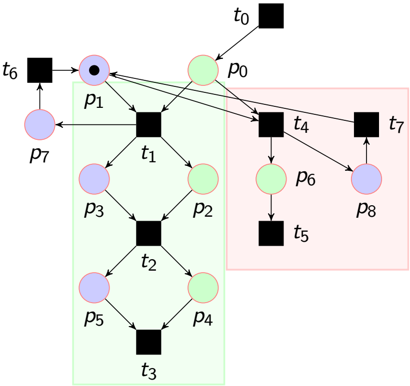
  There is no counterexample for this property for upto bound 50

- Is there atleast one token in either place p_1 or p_7 or p_8?

  $G((\#x > 0)p1(x)|(\#x > 0)p7(x)$

  $|(\#x > 0)p8(x))$

  There is no counterexample for this property for upto bound 100



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

  $F((\#x)p6(x) > p2(x))$.

  At $k = 4$, $\kappa = 1$ (number of tokens), $\lambda = 3$ (time instance), we get a counterexample.

Introducing Counting Logic $\mathcal{L}_C$

In $\mathcal{L}_C$, there are three types of atomic formulas:

1. describing basic server (system) properties, P_s [1]
2. **counting** sentences like: $(\#x > \mathfrak{c})\alpha$ and $(\#x \leq \mathfrak{c})\alpha$ over client (vehicle) properties [2]

   $P_c = \{\text{parking\_requested}, \text{occupy\_parking\_lot}, \cdots\}$
3. **comparing** sentences like: $(\#x)\alpha \leq \beta$ and $(\#x)\alpha > \beta$.

[1] P_s are propositional constants

[2] $\mathfrak{c}$ denotes the number of vehicles in α

Introducing a Counting Logic $\mathcal{L}_C$

Set of client (vehicle) formulas Δ:

$$\begin{aligned}\alpha, \beta \in \Delta ::= & (\#x > \mathfrak{c})p(x) \mid (\#x \leq \mathfrak{c})p(x) \\ & \mid (\#x)p(x) \leq q(x) \mid (\#x)p(x) > q(x) \\ & \mid \alpha \vee \beta \mid \alpha \wedge \beta\end{aligned}$$

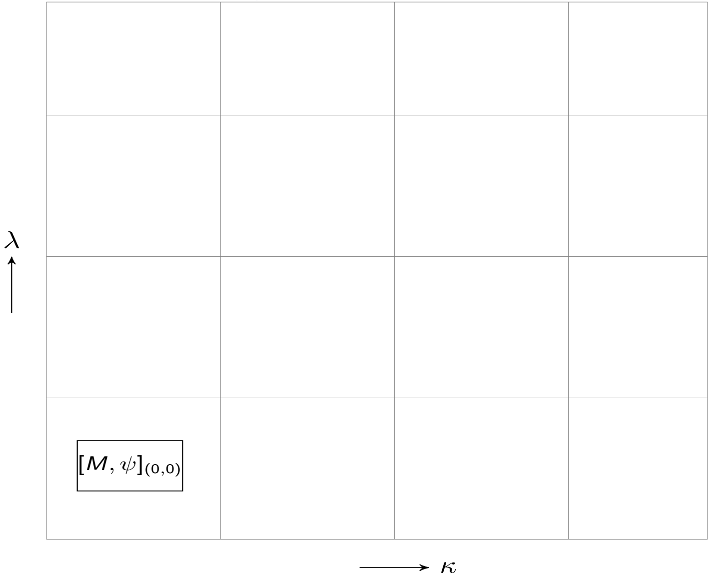
where $p, q \in P_c$ and $\mathfrak{c}$ is a non-negative integer

Introducing a Counting Logic $\mathcal{L}_C$

Set of server (system) formulas Ψ:

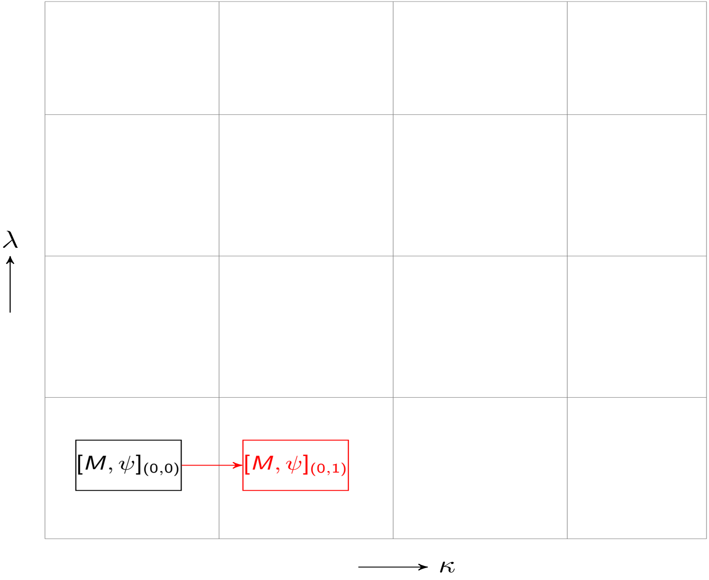
$$\psi \in \Psi ::= q \in P_s \mid \varphi \in \Delta \mid \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi$$
$$\mid X\psi \mid F\psi \mid G\psi \mid \psi_1 U \psi_2$$

The $2D$-BMC strategy



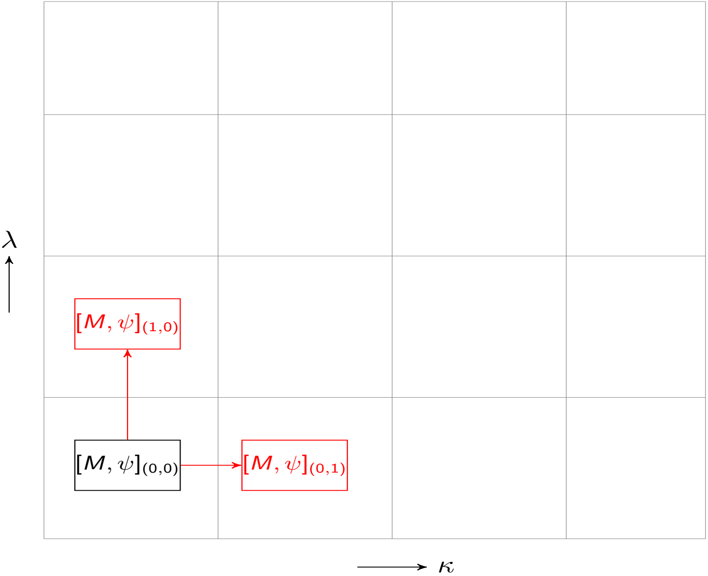
$\lambda + \kappa = k, k \geq 0$, λ : time instance, κ: no. of tokens

The 2D-BMC strategy



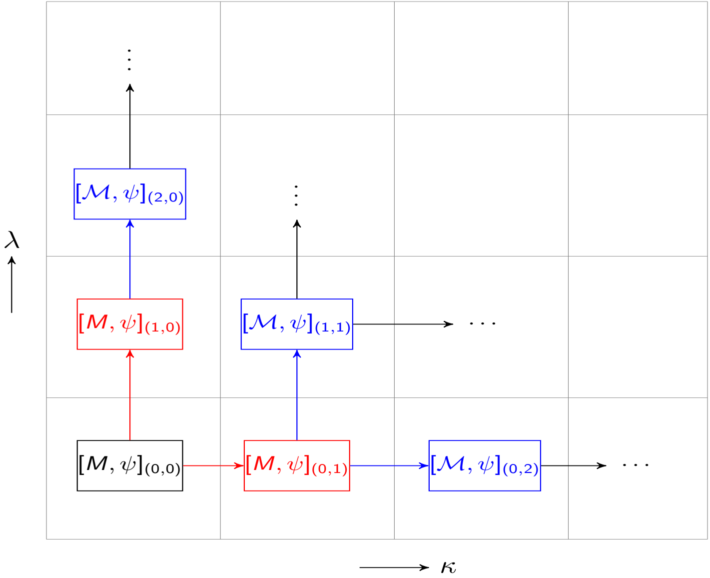
$\lambda + \kappa = k, k \geq 0$, λ : time instance, κ: no. of tokens

The $2D$-BMC strategy

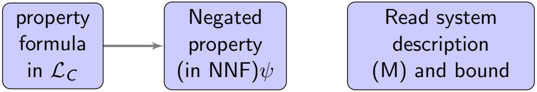


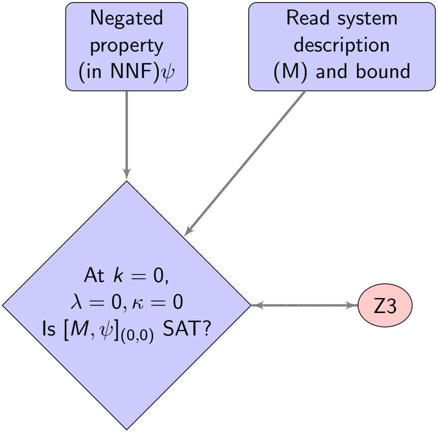
$\lambda + \kappa = k, k \geq 0,\ \lambda$: time instance, κ: no. of tokens

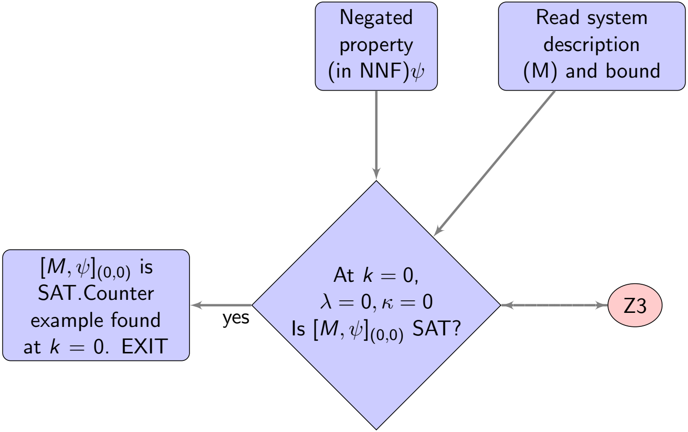
The 2D-BMC strategy



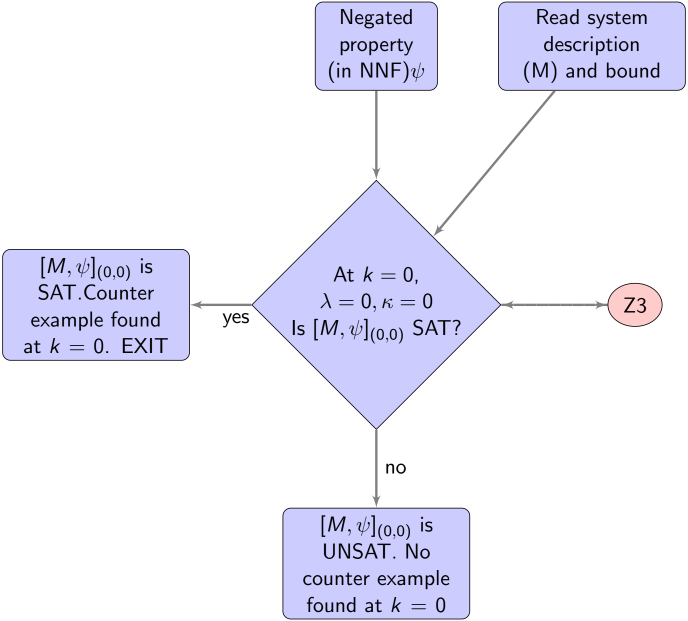
$\lambda + \kappa = k, k \geq 0$, λ : time instance, κ: no. of tokens

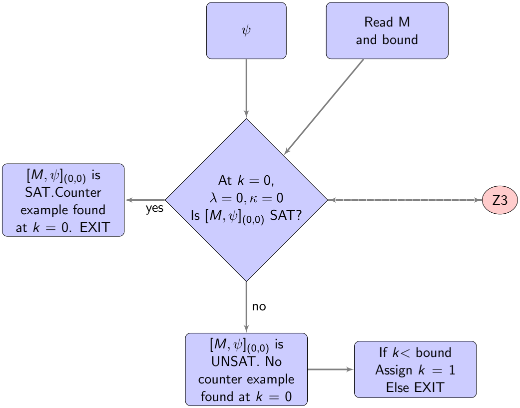

property formula in $\mathcal{L}_C$
Negated property (in NNF) ψ
Read system description (M) and bound

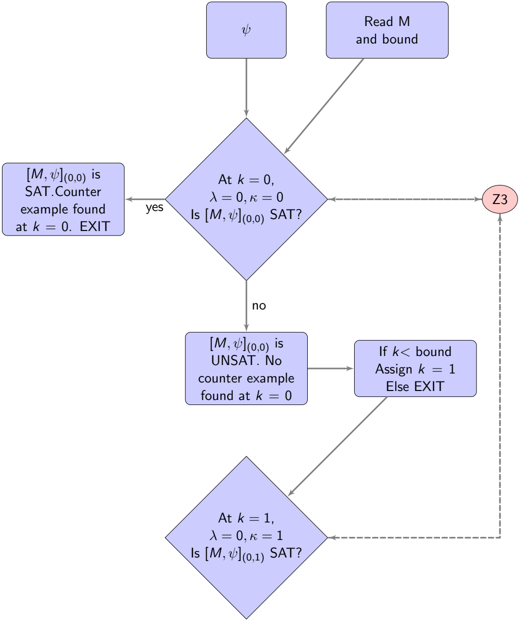

Negated property (in NNF)ψ
Read system description (M) and bound
At $k = 0$, $\lambda = 0, \kappa = 0$ Is $[M, \psi]_{(0,0)}$ SAT?
Z3

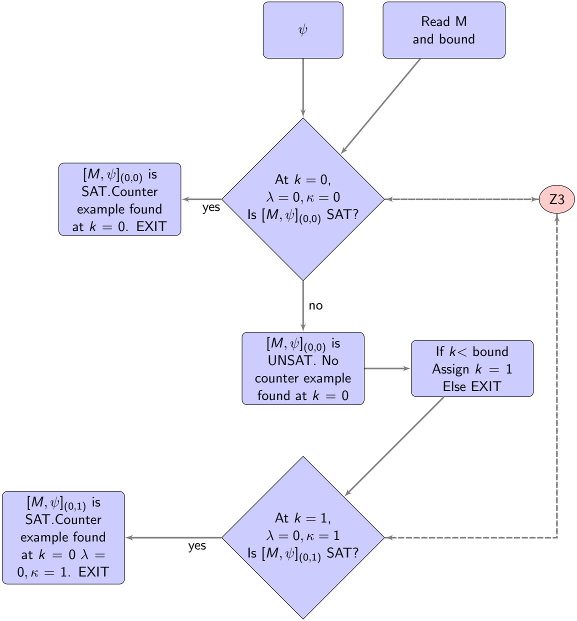


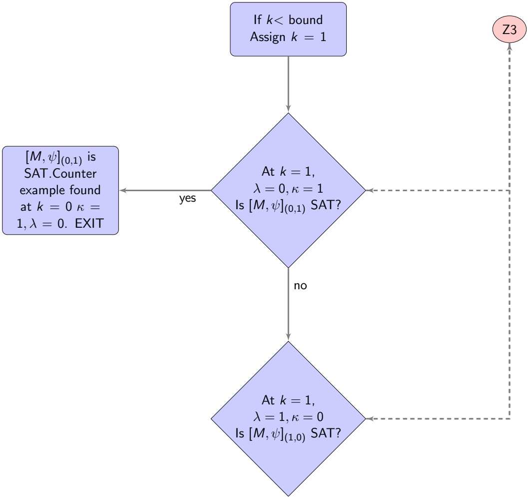
Negated property (in NNF)ψ
Read system description (M) and bound
At $k = 0$, $\lambda = 0, \kappa = 0$ Is $[M, \psi]_{(0,0)}$ SAT?
yes
$[M, \psi]_{(0,0)}$ is SAT.Counter example found at $k = 0$. EXIT
Z3

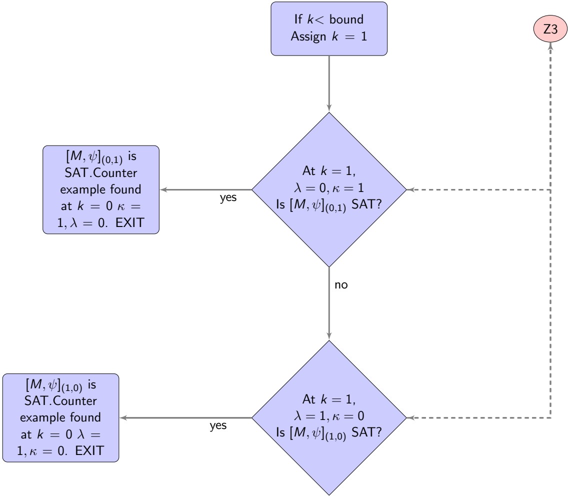

Negated property (in NNF)ψ
Read system description (M) and bound
At $k = 0$, $\lambda = 0, \kappa = 0$ Is $[M, \psi]_{(0,0)}$ SAT?
Z3
yes
$[M, \psi]_{(0,0)}$ is SAT.Counter example found at $k = 0$. EXIT
no
$[M, \psi]_{(0,0)}$ is UNSAT. No counter example found at $k = 0$

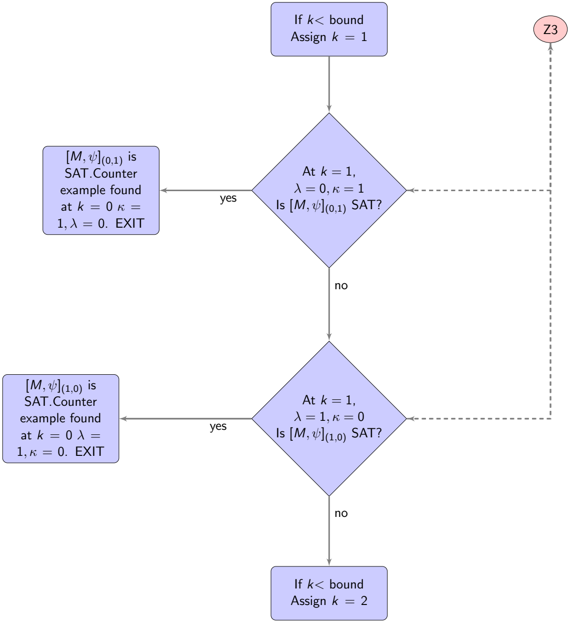

ψ
Read M and bound
At $k = 0$, $\lambda = 0, \kappa = 0$ Is $[M, \psi]_{(0,0)}$ SAT?
yes
$[M, \psi]_{(0,0)}$ is SAT.Counter example found at $k = 0$. EXIT
Z3
no
$[M, \psi]_{(0,0)}$ is UNSAT. No counter example found at $k = 0$
If $k <$ bound Assign $k = 1$ Else EXIT


ψ
Read M and bound
At $k = 0$, $\lambda = 0, \kappa = 0$ Is $[M, \psi]_{(0,0)}$ SAT?
yes
$[M, \psi]_{(0,0)}$ is SAT.Counter example found at $k = 0$. EXIT
Z3
no
$[M, \psi]_{(0,0)}$ is UNSAT. No counter example found at $k = 0$
If $k<$ bound Assign $k = 1$ Else EXIT
At $k = 1$, $\lambda = 0, \kappa = 1$ Is $[M, \psi]_{(0,1)}$ SAT?

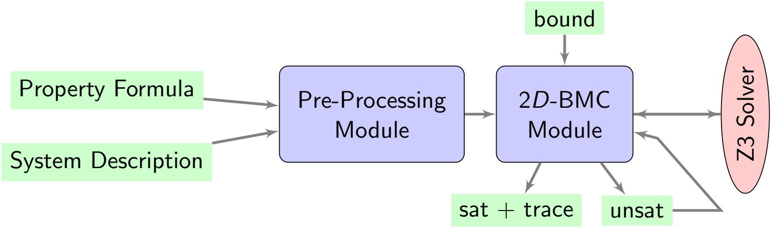

ψ
Read M and bound
At $k = 0$, $\lambda = 0, \kappa = 0$ Is $[M, \psi]_{(0,0)}$ SAT?
yes
$[M, \psi]_{(0,0)}$ is SAT.Counter example found at $k = 0$. EXIT
Z3
no
$[M, \psi]_{(0,0)}$ is UNSAT. No counter example found at $k = 0$
If $k<$ bound Assign $k = 1$ Else EXIT
At $k = 1$, $\lambda = 0, \kappa = 1$ Is $[M, \psi]_{(0,1)}$ SAT?
yes
$[M, \psi]_{(0,1)}$ is SAT.Counter example found at $k = 0$ $\lambda = 0, \kappa = 1$. EXIT


If $k<$ bound
Assign $k = 1$
Z3
$[M,\psi]_{(0,1)}$ is SAT.Counter example found at $k = 0$ $\kappa = 1, \lambda = 0$. EXIT
yes
At $k = 1$, $\lambda = 0, \kappa = 1$ Is $[M,\psi]_{(0,1)}$ SAT?
no
At $k = 1$, $\lambda = 1, \kappa = 0$ Is $[M,\psi]_{(1,0)}$ SAT?


If $k <$ bound
Assign $k = 1$
Z3
At $k = 1$,
$\lambda = 0, \kappa = 1$
Is $[M, \psi]_{(0,1)}$ SAT?
yes
$[M, \psi]_{(0,1)}$ is SAT.Counter example found at $k = 0$ $\kappa = 1, \lambda = 0$. EXIT
no
At $k = 1$,
$\lambda = 1, \kappa = 0$
Is $[M, \psi]_{(1,0)}$ SAT?
yes
$[M, \psi]_{(1,0)}$ is SAT.Counter example found at $k = 0$ $\lambda = 1, \kappa = 0$. EXIT


If $k<$ bound
Assign $k = 1$
Z3
At $k = 1$,
$\lambda = 0, \kappa = 1$
Is $[M, \psi]_{(0,1)}$ SAT?
yes
$[M, \psi]_{(0,1)}$ is SAT.Counter example found at $k = 0$ $\kappa = 1, \lambda = 0$. EXIT
no
At $k = 1$,
$\lambda = 1, \kappa = 0$
Is $[M, \psi]_{(1,0)}$ SAT?
yes
$[M, \psi]_{(1,0)}$ is SAT.Counter example found at $k = 0$ $\lambda = 1, \kappa = 0$. EXIT
no
If $k<$ bound
Assign $k = 2$

Architecture of the Tool DCModelChecker



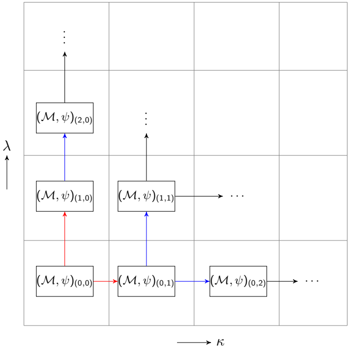
Experiments

Model Name	Property Category	DCModelChecker			ITS-Tools		
		sat	unsat	time(s)	sat	unsat	time(s)
Dekker-PT-010	LTL Cardinality	3	13	11.219	4	12	15.7
	LTL Fireability	2	14	10.353	2	14	18.372
	Reachability Cardinality	0	16	10.497	5	11	3.45
	Reachability Fireability	0	16	11.628	4	12	6.061

- The benchmark was obtained from MCC [3]
- ITS-Tools is a state of the art symbolic model checker [4]

[3] Model Checking Contest https://mcc.lip6.fr/
[4] Yann Thierry Mieg et al.[TACAS 15]

Summary



- In BMC, "We ask whether the system M has any counterexample of length k to (property) ψ. This bounded problem is encoded into SAT." - A.Biere
- In 2D-BMC, we ask whether the system M has any counterexample of length λ and number of clients κ to ψ and encode this bounded problem into SMT.

Summary

- A novel counting logic $\mathcal{L}_C$
- Introduced $2D-$ bounded model checking strategy
- Introduced first-of-its-kind tool DCModelChecker [5] to perform $2D-$ BMC on Petri Nets, using $\mathcal{L}_C$ to specify properties[6]

[5] DCModelChecker tool: https://doi.org/10.6084/m9.figshare.19875226
[6] Technical Report: https://iitdh.ac.in/∼prb/2dbmc_tr_2.pdf.pdf

Future Work

- Optimize our tool by implementing an efficient linear size encoding
- Establish Completeness Criterion for $2D$-BMC and $\mathcal{L}_C$
- Extend $\mathcal{L}_C$ to account for identifiable clients
- Leverage inductive reasoning to have a full decision procedure when satisfiable (similar to the work in QCOVER, ICOVER)

Future Work

- Optimize our tool by implementing an efficient linear size encoding
- Establish Completeness Criterion for $2D$-BMC and $\mathcal{L}_C$
- Extend $\mathcal{L}_C$ to account for identifiable clients
- Leverage inductive reasoning to have a full decision procedure when satisfiable (similar to the work in QCOVER, ICOVER)

Thank you for your attention

Backup

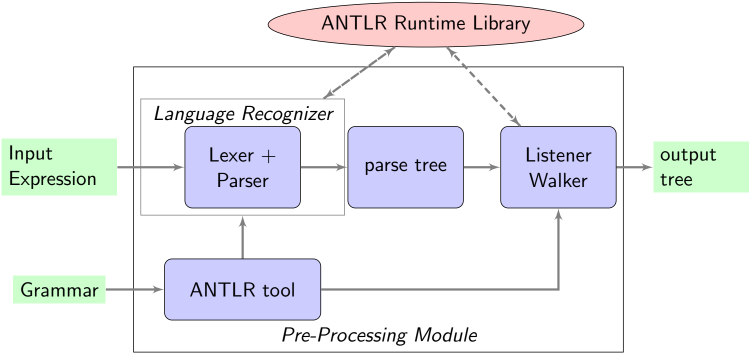


Figure: Pre-Processing Module architecture

$2D-$ Bounded Model Checking Strategy

Petri net model M against the property $\psi = \neg\phi$ is as follows:

$2D-$ Bounded Model Checking Strategy

For any $\kappa \geq 0$, we define $[M]_{\langle 0,\kappa \rangle} = I(s_0) \wedge (\bigwedge_{0 \leq j \leq n_p} p_{j0} \leq \kappa)$.

- k has two parts λ and κ
- λ gives the bound for time instances
- κ gives the bound for number of vehicles in the parking system

$2D-$ Bounded Model Checking Strategy

Inductively, for any $\lambda > 0$,
$[M]_{\langle\lambda,\kappa\rangle} = [M]_{\langle\lambda-1,\kappa\rangle} \wedge (T(s_{\lambda-1}, s_\lambda) \wedge (\bigwedge_{0\leq j\leq n_p} p_{j\lambda} \leq \kappa))$

$2D-$ Bounded Model Checking Strategy

$$[M, \psi]_{\langle \lambda, \kappa \rangle} = [M]_{\langle \lambda, \kappa \rangle} \wedge \Big(\Big)$$

$2D-$ Bounded Model Checking Strategy

$$[M,\psi]_{\langle\lambda,\kappa\rangle} = [M]_{\langle\lambda,\kappa\rangle} \wedge \Big(\big(\neg L_{\langle\lambda,\kappa\rangle} \wedge [\psi]^{0}_{\langle\lambda,\kappa\rangle}\big) \vee () \Big)$$

$2D-$ Bounded Model Checking Strategy

$$[M,\psi]_{\langle\lambda,\kappa\rangle} = [M]_{\langle\lambda,\kappa\rangle} \wedge \Big(\big(\neg L_{\langle\lambda,\kappa\rangle} \wedge [\psi]^0_{\langle\lambda,\kappa\rangle}\big) \vee \bigvee_{l=0}^{k} \big({}_l L_{\langle\lambda,\kappa\rangle} \wedge {}_l[\psi]^0_{\langle\lambda,\kappa\rangle}\big) \Big)$$

Semantics

The logic is interpreted over model sequences. Formally, a model is a sequence $\varrho = m_0, m_1, \ldots$, where for all $i \geq 0$ we have a triple $m_i = (\nu_i, V_i, \xi_i)$ such that:

1. $\nu_i \subset_{fin} P_s$, gives the local properties of the server at instant i.
2. $V_i \subset_{fin} CN$ gives the clients alive at instant i, where CN is a countable set of client names that can be assigned to the vehicles in the system. Further, for all $i \geq 0$, $V_{i+1} \subseteq V_i$ or $V_i \subseteq V_{i+1}$.
3. $\xi_i : V_i \rightarrow 2^{P_c}$ gives the properties satisfied by each live agent at the ith instant.

Semantics

The truth of a formula at an instant in the model is given by the relations $\models$ and $\models_\Delta$ defined by induction over the structure of ψ and α respectively as follows:

1. $\varrho, i \models q$ iff $q \in \nu_i$. Note that q's denote atomic local server propositions. Therefore, a q holds in the model ϱ at instance i if q is in the set ν_i.
2. $\varrho, i \models \varphi$ iff $\varrho, i \models_\Delta \varphi$. Recall that φ is a sentence from the set of client formulae Δ. In order to define the satisfiability of φ, we need to use the rules defined for the relation $\models_\Delta$.
3. $\varrho, i \models \neg\psi$ iff $\varrho, i \not\models \psi$. This rule is standard.
4. $\varrho, i \models \psi \vee \psi'$ iff $\varrho, i \models \psi$ or $\varrho, i \models \psi'$. This rule is standard.

Semantics

5. $\varrho, i \models \psi \wedge \psi'$ iff $\varrho, i \models \psi$ and $\varrho, i \models \psi'$. This rule is standard.
6. $\varrho, i \models X\psi$ iff $\varrho, i+1 \models \psi$. This rule is standard.
7. $\varrho, i \models F\psi$ iff $\exists j \geq i$, $\varrho, j \models \psi$. This rule is standard.
8. $\varrho, i \models G\psi$ iff $\forall j \geq i$, $\varrho, j \models \psi$. This rule is standard.
9. $\varrho, i \models \psi_1 U \psi_2$ iff $\exists j \geq i$, $\varrho, j \models \psi_2$ and for all $i \leq j' < j : \varrho, j' \models \psi_1$. This rule is standard.

Semantics

10. $\varrho, i \models_\Delta (\#x > \mathfrak{c})p(x)$ iff $|\{a \in V_i \mid p \in \xi_i(a)\}| > \mathfrak{c}$. The client formula $(\#x > \mathfrak{c})p(x)$ holds in model ϱ at instance i if there are strictly more than $\mathfrak{c}$ clients that satisfy the property p at instance i.
11. $\varrho, i \models_\Delta (\#x)p(x) \leq q(x)$ iff $|\{a \in V_i \mid p \in \xi_i(a)\}| \leq |\{b \in V_i \mid q \in \xi_i(b)\}|$. The client formula $(\#x)p(x) \leq q(x)$ holds in the model ϱ at instance i if the number of clients satisfying property p is less than the number of clients satisfying the property q, at the same instance i.
12. $\varrho, i \models_\Delta (\#x \leq \mathfrak{c})p(x)$ iff $|\{a \in V_i \mid p \in \xi_i(a)\}| \leq \mathfrak{c}$. The client formula $(\#x \leq \mathfrak{c})p(x)$ holds in model ϱ at instance i if there are less than $\mathfrak{c}$ clients that satisfy the property p at instance i.

Semantics

13. $\varrho, i \models_\Delta (\#x)p(x) > q(x)$ iff $|\{a \in V_i \mid p \in \xi_i(a)\}| > |\{b \in V_i \mid q \in \xi_i(b)\}|$. The client formula $(\#x)p(x) > q(x)$ holds in the model ϱ at instance i if the number of clients satisfying property p is strictly more than the number of clients satisfying the property q, at the same instance i.
14. $\varrho, i \models_\Delta \alpha \vee \beta$ iff $\varrho, i \models_\Delta \alpha$ or $\varrho, \pi, i \models_\Delta \beta$. This rule is standard.
15. $\varrho, i \models_\Delta \alpha \wedge \beta$ iff $\varrho, i \models_\Delta \alpha$ and $\varrho, \pi, i \models_\Delta \beta$. This rule is standard.

Preliminaries: Petri Nets

A Petri Net structure is a tuple $N = (P, T, F, W)$ where

- P is a finite set of places
- T is a finite set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation
- $W : F \to \aleph_0$ is a weight function, where $\aleph_0$ is the set of non-negative integers.
- For any place (resp. transition) z of the set $P \cup T$, the set $\{x \mid (x, z) \in F\}$ is called *pre-transitions* (resp. *pre-places*) of z
- the set $\{x \mid (z, x) \in F\}$ is called *post-transitions* (resp. *post-places*) of z

Preliminaries: Petri Nets

- A marking M of a Petri Net is a function $M : P \to \aleph_0$.
- A Petri Net (system) is a tuple $\mathcal{M} = (N, M_0)$ where N is a Petri Net structure and M_0 is an initial marking.
- A transition t is *enabled* at marking M, if for each pre-place p of t we have $M(p) \geq W(p, t)$.
- A new marking is obtained when an enabled transition is *fired*, and is obtained by removing $W(p, t)$ tokens from each pre-place p of t, and adding $W(t, p)$ tokens to each post-place p of t, leaving tokens in the remaining places as it is.

Preliminaries: Petri Nets

The propositional encoding of N is defined by the formula $\mathcal{T} = \mathcal{T}_{enabled} \wedge \mathcal{T}_{firability} \wedge \mathcal{T}_{next}$, where:

- the formula $\mathcal{T}_{enabled}$ states that more than one transition can be enabled at a time. i.e, it is an **or** over the preconditions of all enabled transitions and is given by $\mathcal{T}_{enabled} = pre_{t0} \vee pre_{t1} \vee \cdots \vee pre_{t_n}$;
- the formula $\mathcal{T}_{next}$ gives us the next transition that will be fired, and is expressed as an **or** expression over each transition **and** its postcondition **and** all other transitions and postconditions are negated.
  $\mathcal{T}_{next} =$
  $$(t_0 \wedge \neg t_1 \wedge \cdots \wedge \neg t_n \wedge post_{t_0} \wedge \neg post_{t_1} \wedge \cdots \wedge \neg post_{t_n}) \vee$$
  $$(\neg t_0 \wedge t_1 \wedge \cdots \wedge \neg t_n \wedge \neg post_{t_0} \wedge post_{t_1} \wedge \cdots \wedge \neg post_{t_n}) \vee \cdots \vee$$
  $$(\neg t_0 \wedge \neg t_1 \wedge \cdots \wedge t_n \wedge \neg post_{t_0} \wedge \neg post_{t_1} \wedge \cdots \wedge post_{t_n});$$

Preliminaries: Petri Nets

- the formula $\mathcal{T}_{firability}$ relates the pre condition of a transition with its distinct postcondition and is given as

  $\mathcal{T}_{firability} = (post_{t_0} \rightarrow pre_{t_0}) \wedge (post_{t_1} \rightarrow pre_{t_1}) \wedge \cdots \wedge (post_{t_n} \rightarrow pre_{t_n})$;

  where for each t_i, pre_{ti} is a propositional formula defined over place variables encoding that the precondition of firing t_i is satisfied and $post_{ti}$ is a formula defined over place variables encoding the update in tokens after firing t_i is satisfied.

We denote the encoding of Petri Net system $\mathcal{M} = (N, M_0)$ by $[\mathcal{M}] = (\mathcal{T}, M_0)$ where M_0 is the initial assignment of the marking vector M.