

Undecidability of Parameterized Model Checking

Part III: Undecidability of Parameterized Model Checking

Annu Gmeiner Igor Konnov Ulrich Schmid
Helmut Veith Josef Widder



for(synte)
Formal Methods
in Systems Engineering

RiSE
Rigorous Systems Engineering

VTSA 2014, Luxembourg

Classic Model Checking Problems

Consider P^N as interleaving of N processes of type P .

Finite-state MC

Input:

- ▶ a process template P (finite-state),
- ▶ an LTL formula φ ,
- ▶ the number of processes $N \geq 1$.

Problem: check, whether $P^N \models \varphi$.

Classic Model Checking Problems

Consider P^N as interleaving of N processes of type P .

Finite-state MC

Input:

- ▶ a process template P (finite-state),
- ▶ an LTL formula φ ,
- ▶ the number of processes $N \geq 1$.

Problem: check, whether $P^N \models \varphi$.

Decidable!

Classic Model Checking Problems

Consider P^N as interleaving of N processes of type P .

Finite-state MC

Input:

- ▶ a process template P (finite-state),
- ▶ an LTL formula φ ,
- ▶ the number of processes $N \geq 1$.

Problem: check, whether $P^N \models \varphi$.

Decidable!

Parameterized MC

Input:

- ▶ a process template P (finite-state),
- ▶ an (indexed) LTL formula ϕ ,

Problem: check, whether $\forall N \geq 1. P^N \models \phi(N)$.

Decidable?

The First Result: Undecidability of PMC

Apt & Kozen 1986:

One process $P(n)$ with a parameterized loop bound n ,
 $P(n)$ simulates n steps of a Turing machine

```
flag := false
for i := 1 to n do
    simulate one step of T
if T has not halted then
    flag := true
```

As non-halting is undecidable,
so is PMC for n non-communicating processes $P^n(n)$.

The First Result: Undecidability of PMC

Apt & Kozen 1986:

One process $P(n)$ with a parameterized loop bound n ,
 $P(n)$ simulates n steps of a Turing machine

```
flag := false
for i := 1 to n do
    simulate one step of T
if T has not halted then
    flag := true
```

As non-halting is undecidable,
so is PMC for n non-communicating processes $P^n(n)$.

Note: This proof is about sequential and unbounded programs.

PMC for fixed (non-parameterized processes)

Suzuki 1988: PMCP is undecidable for an **apparently** simple case:

one fixed, **finite-state** process P independent of n ,
 P^n is a ring, processes communicate by token passing.

PMC for fixed (non-parameterized processes)

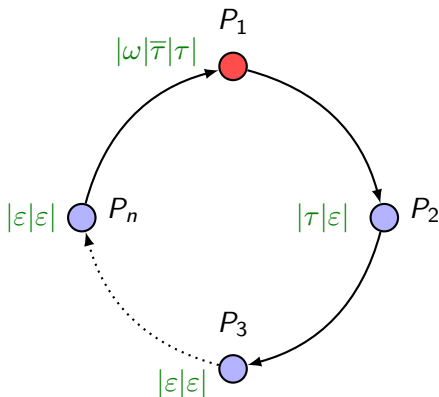
Suzuki 1988: PMCP is undecidable for an **apparently** simple case:

one fixed, **finite-state** process P independent of n ,

P^n is a ring, processes communicate by token passing.

P^n simulates a Turing machine that writes not more than in $2n + 1$ cells.

- ▶ Process P_1 serves as a head and stores **three letters**.
- ▶ Each process P_i , $i > 1$, stores **two letters**.
- ▶ The head moves to the right
 \Rightarrow the tape is cyclically shifted.



PMC for fixed (non-parameterized processes)

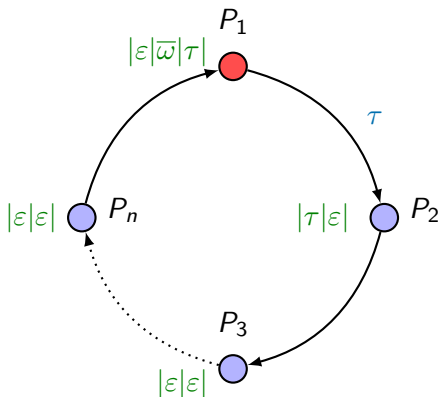
Suzuki 1988: PMCP is undecidable for an **apparently** simple case:

one fixed, **finite-state** process P independent of n ,

P^n is a ring, processes communicate by token passing.

P^n simulates a Turing machine that writes not more than in $2n + 1$ cells.

- ▶ Process P_1 serves as a head and stores **three letters**.
- ▶ Each process P_i , $i > 1$, stores **two letters**.
- ▶ The head moves to the right
 \Rightarrow the tape is cyclically shifted.



PMC for fixed (non-parameterized processes)

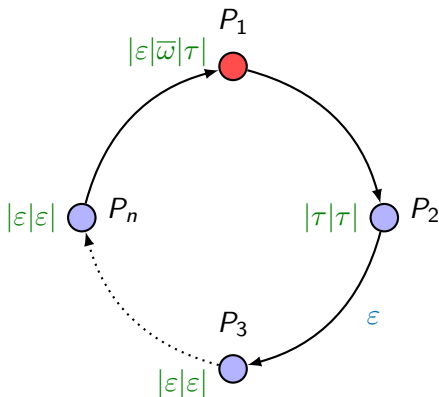
Suzuki 1988: PMCP is undecidable for an **apparently** simple case:

one fixed, **finite-state** process P independent of n ,

P^n is a ring, processes communicate by token passing.

P^n simulates a Turing machine that writes not more than in $2n + 1$ cells.

- ▶ Process P_1 serves as a head and stores **three letters**.
- ▶ Each process P_i , $i > 1$, stores **two letters**.
- ▶ The head moves to the right
 \Rightarrow the tape is cyclically shifted.



PMC for fixed (non-parameterized processes)

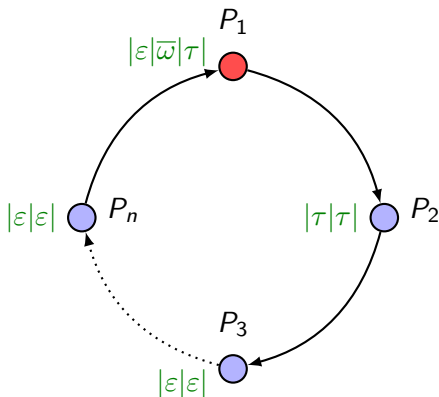
Suzuki 1988: PMCP is undecidable for an **apparently** simple case:

one fixed, **finite-state** process P independent of n ,

P^n is a ring, processes communicate by token passing.

P^n simulates a Turing machine that writes not more than in $2n + 1$ cells.

- ▶ Process P_1 serves as a head and stores **three letters**.
- ▶ Each process P_i , $i > 1$, stores **two letters**.
- ▶ The head moves to the right \Rightarrow the tape is cyclically shifted.
- ▶ The proof is **very technical**.
- ▶ A simpler proof is given by **Emerson & Namjoshi, 1995**.

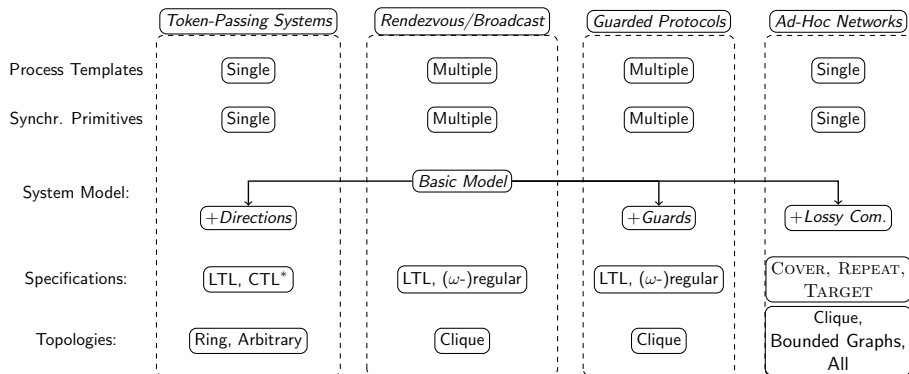


Many more results on Parameterized Model Checking

We have recently finished a book on:

“Decidability of Parameterized Verification”.

To appear in Synthesis Lectures on Distributed Computing Theory.



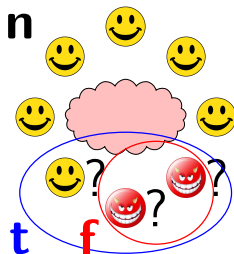
Fault-tolerant distributed algorithms and parameterized model checking

Recall our parameterized problem:

- ▶ given a process of a distributed algorithm and spec. φ
- ▶ show for all n , t , and f satisfying $n > 3t \wedge t \geq f \geq 0$
 $P(n, t, f) \parallel \dots \parallel P(n, t, f) \models \varphi$
- ▶ every $M(n, t, f)$ is a system of $n - f$ correct processes

The problem statement is different!

- ▶ Processes are **parameterized** in n , t , and f .
- ▶ Processes communicate by sending messages to **everybody**.
- ▶ Processes count received messages.
- ▶ LTL specifications over $[\forall i. f(i)]$ and $[\exists i. f(i)]$.



Is our parameterized model
checking problem decidable?

Undecidability of PMC for FTDA

Given a two-counter machine \mathcal{M} ,
we construct a process P and an LTL formula φ .

Then the non-halting of \mathcal{M} is formulated as: $\forall n \geq 2. P^n \models \varphi$.

The idea of the proof:

- ▶ Construct a finite-state process P
- ▶ P has only finitely many locations
- ▶ P does not communicate at all
- ▶ LTL formulas with **G** and **F** over atomic propositions like $[\exists i. pc_i = Z]$.

Counter Machines

A machine is composed of the following commands

$$\begin{aligned} \ell_i &: \mathbf{inc} \ C(\ell_i); \ \mathbf{goto} \ \ell_j \\ \ell_i &: \mathbf{if} \ C(\ell_i) = 0 \ \mathbf{then} \ \mathbf{goto} \ \ell_j \\ &\quad \mathbf{else} \ \mathbf{dec} \ C(\ell_i); \ \mathbf{goto} \ \ell_k \\ \ell_m &: \mathbf{halt} \end{aligned}$$

over a few counters, e.g., $C(\ell_i) \in \{B, C\}$.

The halting (as well as non-halting) of a two-counter machine is undecidable [Minsky1967].

Counter Machines

A machine is composed of the following commands

$$\begin{aligned} \ell_i &: \text{inc } C(\ell_i); \text{ goto } \ell_j \\ \ell_i &: \text{if } C(\ell_i) = 0 \text{ then goto } \ell_j \\ &\quad \text{else dec } C(\ell_i); \text{ goto } \ell_k \\ \ell_m &: \text{halt} \end{aligned}$$

over a few counters, e.g., $C(\ell_i) \in \{B, C\}$.

The halting (as well as non-halting) of a two-counter machine is undecidable [Minsky1967].

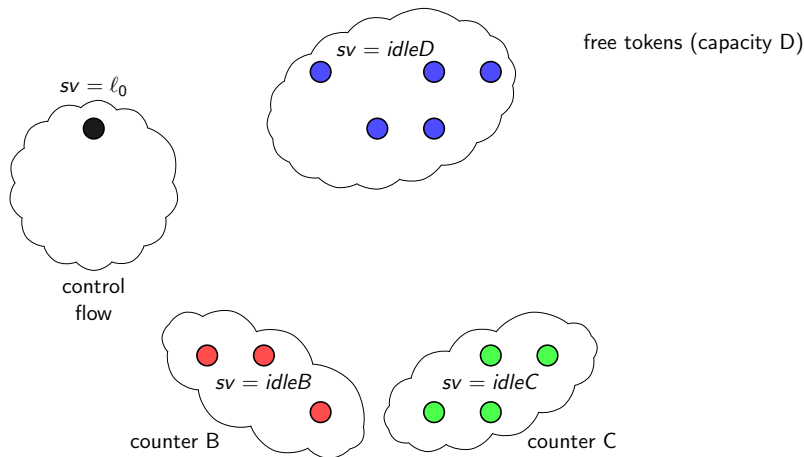
Many undecidability results are proven by reduction to halting of a 2CM.

For instance, [Esparza 1997]:

checking, whether a Petri net satisfies a linear μ -calculus formula simulates halting of a two-counter machine.

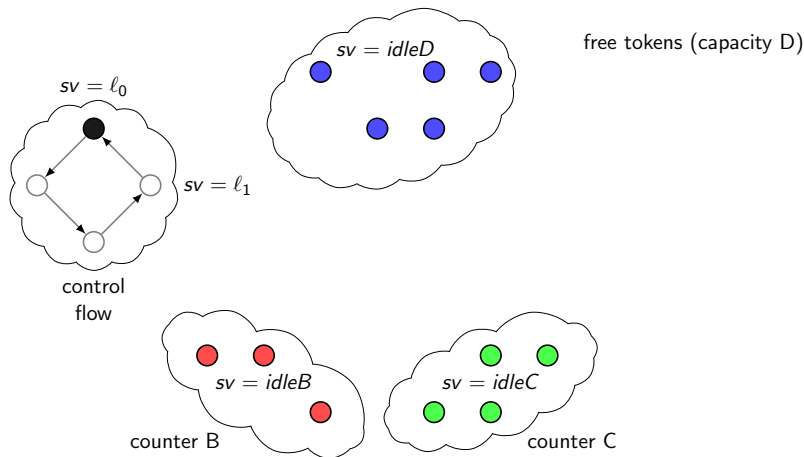
How a configuration looks like?

The process states are partitioned into four classes:



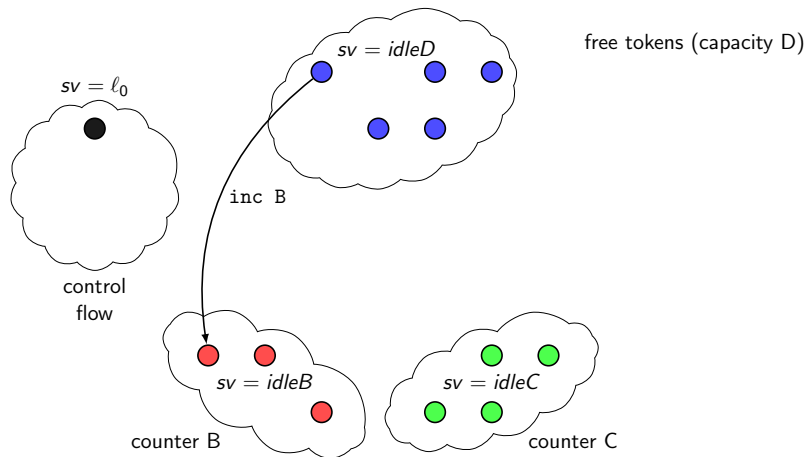
How a configuration looks like?

The process states are partitioned into four classes:



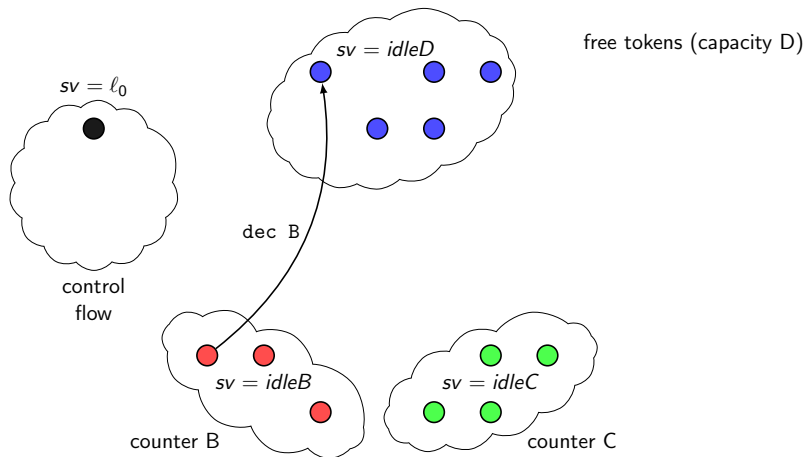
How a configuration looks like?

The process states are partitioned into four classes:



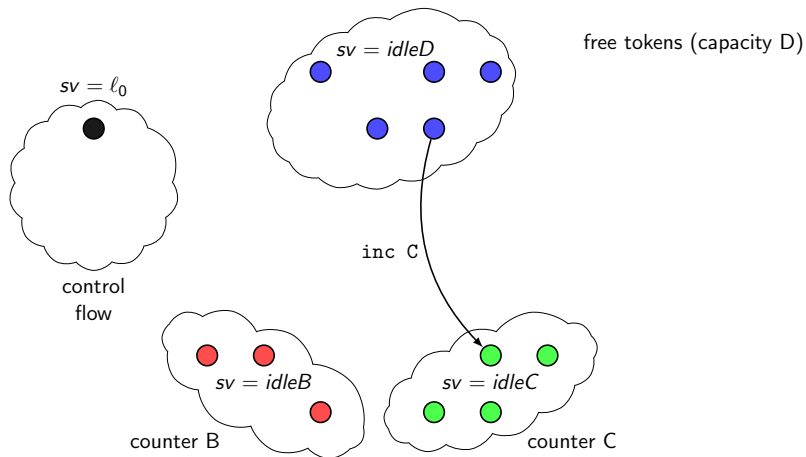
How a configuration looks like?

The process states are partitioned into four classes:



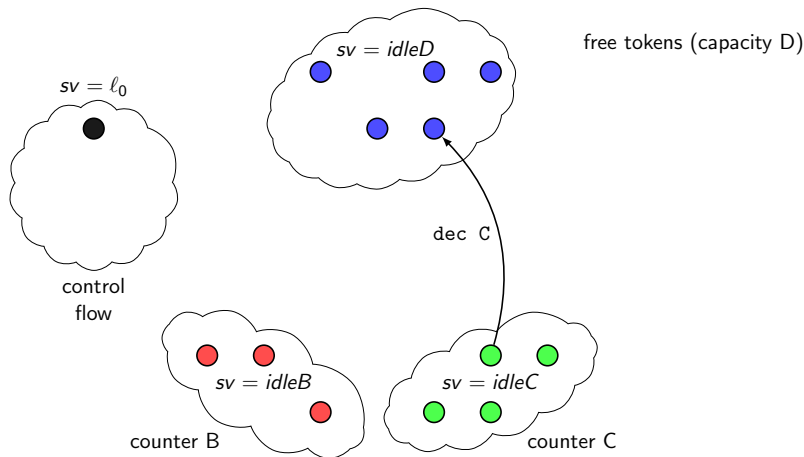
How a configuration looks like?

The process states are partitioned into four classes:



How a configuration looks like?

The process states are partitioned into four classes:



Parameterized model checking is undecidable for our problem.

We need either abstraction techniques, or semi-decision procedures.

In Part IV, we continue with abstraction techniques for parameterized model checking of fault-tolerant distributed algorithms.

Thank you!