IA159 Formal Verification Methods

Model Checking: An Overview

Jan Obdržálek
Jan Strejček

Department of Computer Science
Faculty of Informatics
Masaryk University

Focus and sources

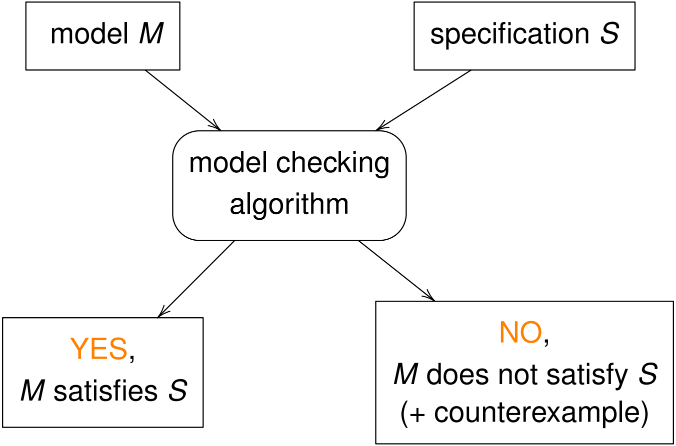
Focus

- model checking in general
- specifications, linear temporal logic (LTL), Büchi automata
- models, Kripke structure, process rewrite systems (PRS)
- model checking problems and decidability
- LTL model checking of finite systems
- state explosion problem

Sources

- Chapters 1, 2, 3 and 9 of *E. M. Clarke, O. Grumberg, and D. A. Peled: Model Checking, MIT, 1999.*
- *R. Mayr: Decidability and Complexity of Model Checking Problems for Infinite-State Systems. PhD thesis, 1998.*

Model checking schema



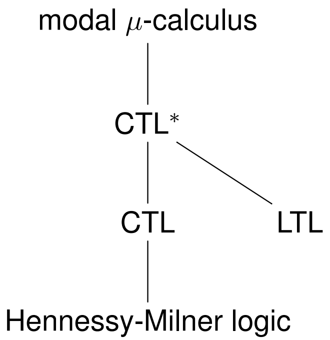
Model checking

Specification

Specification

- a finite formal description of some property that should be satisfied by all behaviours of the system
- usually does not fully specify the system
- typically given by a formula of some temporal logic
  - Linear Temporal Logic (LTL) (linear time)
  - Computational Tree Logic (CTL) (branching time)
  - CTL*, Hennessy–Milner logic, μ calculus, ...
- can be given also by a Büchi automaton, etc.

The hierarchy of basic temporal logics.



The hierarchy of selected temporal logics according to their expressive power.

State-based vs. action-based logics

state-based These logics talk about properties of states of a system. Properties of a single state are reflected by validity of atomic propositions in the state. State-based logic are interpreted over behaviours of the system represented by sequences (or trees) of sets of valid atomic propositions.

action-based Every transition of a system is labelled with an action. Action-based logic are interpreted over behaviours of the system represented only by sequences (or trees) of actions.

We provide definition of both state-based and action-based LTL.

Syntax of state-based LTL

State-based Linear Temporal Logic (LTL) is defined by

$$\varphi ::= \top \mid a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2$$

where $\top$ stands for true and a ranges over a countable set AP of atomic propositions.

Abbreviations $\quad \bot \equiv \neg\top \qquad \mathsf{F}\varphi \equiv \top\,\mathsf{U}\,\varphi \qquad \mathsf{G}\varphi \equiv \neg\mathsf{F}\neg\varphi$

Terminology and intuitive meaning

$\mathsf{X}a$	next	$\bullet\ a\ \bullet\ \bullet\ \bullet\ \dots$
$a\,\mathsf{U}\,b$	until	$a\ a\ \dots\ a\ b\ \bullet\ \bullet\ \bullet\ \dots$
$\mathsf{F}a$	eventually	$\bullet\ \bullet\ \dots\ \bullet\ a\ \bullet\ \bullet\ \bullet\ \dots$
$\mathsf{G}a$	always	$a\ a\ a\ a\ \dots$

Semantics of state-based LTL

Let $\Sigma = 2^{AP'}$, where $AP' \subseteq AP$ is a finite subset. We interpret LTL on infinite words $w = w(0)w(1)\ldots \in \Sigma^\omega$. By w_i we denote the suffix of w of the form $w(i)w(i+1)w(i+2)\ldots$.
The validity of an LTL formula φ for $w \in \Sigma^\omega$, written $w \models \varphi$, is defined as

$$
\begin{array}{lll}
w \models \top & & \\
w \models a & \text{iff} & a \in w(0) \\
w \models \neg\varphi & \text{iff} & w \not\models \varphi \\
w \models \varphi_1 \wedge \varphi_2 & \text{iff} & w \models \varphi_1 \wedge w \models \varphi_2 \\
w \models \mathrm{X}\varphi & \text{iff} & w_1 \models \varphi \\
w \models \varphi_1 \,\mathrm{U}\, \varphi_2 & \text{iff} & \exists i \in \mathbb{N}_0 : w_i \models \varphi_2 \wedge \forall\, 0 \le j < i : w_j \models \varphi_1
\end{array}
$$

Given an alphabet Σ, an LTL formula φ defines the language

$$L^\Sigma(\varphi) = \{w \in \Sigma^\omega \mid w \models \varphi\}.$$

Action-based LTL

Differences between action-based and state-based LTL

- In the syntax, a ranges over countable set of actions Act.
- Formulae of action-based LTL are then interpreted over infinite sequences w of actions from a finite subset $Act' \subseteq Act$.
- Semantics of formula a is defined as follows:

$$w \models a \qquad \text{iff} \quad a = w(0)$$

Examples of LTL formulae

- $G\neg \mathit{error}$ - safety property
- $G(p \implies Fq)$ - response property
- GFp - liveness property

Büchi automata

A Büchi automaton (BA) is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, where

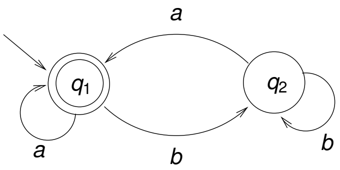
- Σ is a finite alphabet,
- Q is a finite set of states,
- $\delta : Q \times \Sigma \to 2^Q$ is a transition function,
- $q_0 \in Q$ is an initial states,
- $F \subseteq Q$ is a set of accepting states.

A run of $\mathcal{A}$ on infinite word $w = w(0)w(1)... \in \Sigma^\omega$ is an infinite sequence of states $\sigma = \sigma(0)\sigma(1)...$, where $\sigma(0) = q_0$ and $\sigma(i+1) \in \delta(\sigma(i), w(i))$ holds for all i.

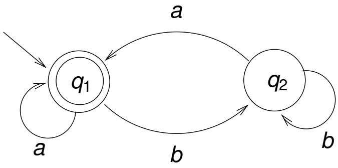
A run σ is accepting if $Inf(\sigma) \cap F \neq \emptyset$, where $Inf(\sigma)$ is the set of the states appearing in σ infinitely often. An automaton $\mathcal{A}$ accepts a word w if there is an accepting run of $\mathcal{A}$ on w. We set

$$L(\mathcal{A}) = \{w \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } w\}.$$

Example of a Büchi automaton



Example of a Büchi automaton



Accepts the words with infinitely many occurrences of *a*.

Model checking

Model

Model

- a finite formal description of all possible behaviours of the system to be verified
- behaviour is a sequence (or a tree) of states/actions
- state is an image of the system in a certain moment (current values of variables, program counter, etc.)
- a state is characterized by validity of atomic propositions (e.g. $PC == start$, $x > 5$)
- many possible formalisms
  - standard languages C, Java, VHDL, . . .
  - dedicated languages, e.g. ProMeLa (Process or Protocol Meta Language)
  - process rewrite systems (infinite-state systems) BPA, BPP, PA, pushdown processes, Petri nets, . . .
  - low-level formalisms: Kripke structure (for state-based approach) and labelled transition systems (for action-based approach)

Example: mutual exclusion in ProMeLa

```
byte cnt = 0;  // number of processes in critical sections
byte turn = 0; // token for entering a critical section

init {
   run(P0); run(P1);  // parallel execution of P0 a P1
}
```

```
proctype P0()
{
  // s0
  do
  // NC0 (noncritical section)
  :: do
     :: (turn == 0) -> break;
     :: else;
     od;
     // CS0 (critical section)
     cnt = cnt + 1;
     cnt = cnt - 1;
     turn = 1;
  od;
}
```

```
proctype P1()
{
  //s1
  do
  // NC1 (noncritical section)
  :: do
     :: (turn == 1) -> break;
     :: else;
     od;
     // CS1 (critical section)
     cnt = cnt + 1;
     cnt = cnt - 1;
     turn = 0;
  od;
}
```

Kripke structure

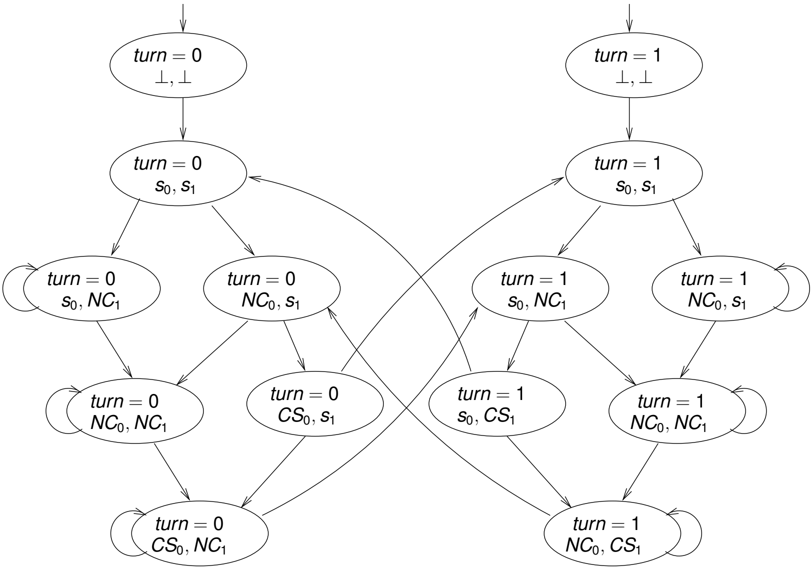
Let AP be a countable set of atomic propositions.

A Kripke structure is a tuple $M = (S, R, S_0, L)$, where

- S is a set of states
- $R \subseteq S \times S$ is transitions relation
- $S_0 \subseteq S$ is a set of initial states
- $L : S \to 2^{AP}$ is a labelling function associating to each state $s \in S$ the set of atomic propositions that are true in s.

A path in M starting in a state s is an infinite sequence $\pi = s_0 s_1 s_2 ...$ of states such that $s_0 = s$ and $(s_i, s_{i+1}) \in R$ holds for every i.

Example: mutual exclusion as a Kripke structure



Process rewrite systems: motivation

- finite-state systems have very limited expressive power
- there are some classes of infinite-state systems with decidable LTL model checking problem
- many standard classes of infinite-state systems are definable uniformly as subclasses of Process Rewrite Systems (PRS)

Process rewrite systems: process terms

Let $Const = \{A, B, C, \ldots\}$ be a countably infinite set of process constants. Process terms are defined by the abstract syntax

$$t ::= \varepsilon \mid A \mid t_1.t_2 \mid t_1 \| t_2,$$

where

- ε is the empty term,
- $A \in Const$ is a process constant (used as an atomic process),
- '$\|$' means a parallel composition, and
- '.' means a sequential composition.

We always work with equivalence classes of terms modulo commutativity and associativity of '$\|$' ($(A\|B)\|C = B\|(A\|C)$) and modulo associativity of '.' ($(A.B).C = A.(B.C)$).

Process rewrite systems: classes of process terms

We distinguish four classes of process terms as:

- "1" terms consisting of a single process constant only (i.e. $\varepsilon \notin 1$), e.g. A.
- "S" sequential terms without parallel composition, e.g. $A.B.C$.
- "P" parallel terms without sequential composition. e.g. $A \| B \| C$.
- "G" general terms with arbitrarily nested sequential and parallel compositions.

Process rewrite systems: syntax

Let $Act = \{a, b, \cdots\}$ be a countably infinite set of atomic actions and $\alpha, \beta \in \{1, S, P, G\}$ such that $\alpha \subseteq \beta$. An (α, β)-PRS (process rewrite system) is a pair $\Delta = (R, t_0)$, where

- $R \subseteq ((\alpha \setminus \{\varepsilon\}) \times Act \times \beta)$ is a finite set of rewrite rules, and
- $t_0 \in \beta$ is an initial term.

We write $(t_1 \overset{a}{\hookrightarrow} t_2) \in R$ instead of $(t_1, a, t_2) \in R$.

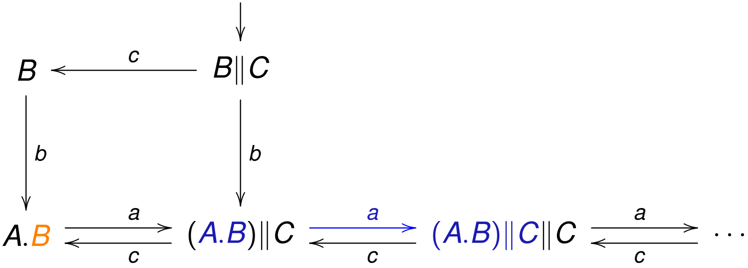
Process rewrite systems: semantics

An (α, β)-PRS $\Delta = (R, t_0)$ defines a labelled transition system where

- states are process terms of β,
- t_0 is the initial state,
- the transition relation $\longrightarrow$ is the least relation satisfying the following inference rules:

$$\frac{(t_1 \stackrel{a}{\hookrightarrow} t_2) \in R}{t_1 \stackrel{a}{\longrightarrow} t_2} \qquad \frac{t_1 \stackrel{a}{\longrightarrow} t_2}{t_1 \| t \stackrel{a}{\longrightarrow} t_2 \| t} \qquad \frac{t_1 \stackrel{a}{\longrightarrow} t_2}{t_1 . t \stackrel{a}{\longrightarrow} t_2 . t}$$

Process rewrite systems: example



(S, G)-PRS $(R, B\|C)$ with rewrite rules

$$R = \{ \quad B \overset{b}{\hookrightarrow} A.B, \quad A.B \overset{a}{\hookrightarrow} (A.B)\|C, \quad C \overset{c}{\hookrightarrow} \varepsilon \quad \}$$

Process rewrite systems: power of rewrite rules

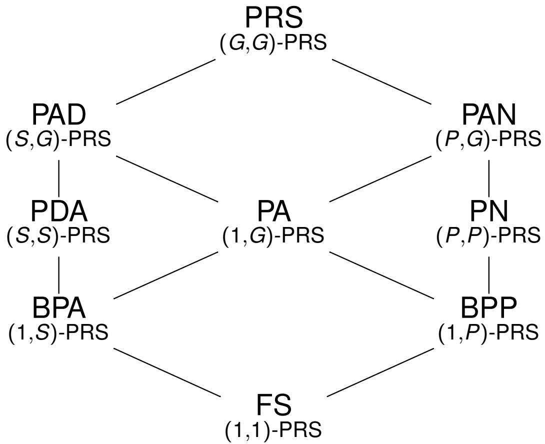
$(1,1)$-PRS	finite-state systems
$m \overset{\texttt{x:=x+1}}{\hookrightarrow} n$	simple sequential programs without procedures
$(1,S)$-PRS	basic process algebra
$m \overset{\texttt{call p}}{\hookrightarrow} p_0.n$	programs with procedure calls no global variables and return values
(S,S)-PRS	pushdown systems
$g.m \overset{\texttt{call p}}{\hookrightarrow} g.p_0.n$	sequential programs with procedures global variables, return values

Process rewrite systems: power of rewrite rules

$(1, P)$-PRS	basic parallel processes
$m \overset{\texttt{create thread f}}{\hookrightarrow} n \| f_0$	programs with simple parallel threads no communication
(P, P)-PRS	Petri nets
$m \| p \overset{\texttt{synchronize}}{\hookrightarrow} n \| q$	programs with parallel threads communication between threads

Process rewrite systems hierarchy (PRS-hierarchy)

The hierarchy compares expressive power of many classes of infinite-state systems including BPA, BPP, PA, Petri nets (PN), and pushdown processes (PDA). FS stands for finite systems.



Model checking

Decidability of model checking

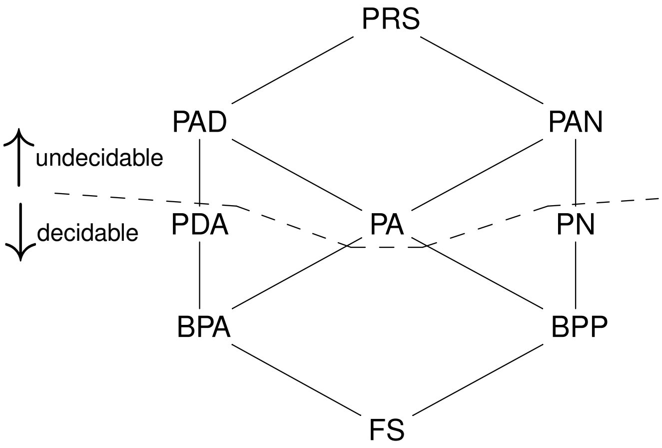
Model checking

Model checking problem is to decide whether all behaviours of a given system satisfy a given specification.

- specific problems for specific input
  - state-based LTL model checking of finite systems
  - action-based CTL model checking of finite systems
  - state-based LTL model checking of pushdown processes
  - action-based LTL model checking of pushdown processes
  - ...
- model checking problem is not decidable for some kinds of input (e.g. action-based LTL model checking of PA processes)
- even small changes of the problem can be important: action-based LTL model checking of PN is decidable, while state-based LTL model checking of PN in undecidable
- all model checking problems are decidable for finite systems

The decidability boundary

The decidability boundary of the action-based LTL model checking in the PRS-hierarchy.



Model checking

Automata-based LTL model checking of finite systems

Automata-based LTL model checking of finite systems

Kripke structure M
with finitely many states

↓

Büchi automaton $\mathcal{A}_M$
accepts paths of M starting in initial states and projected by L to $2^{AP(\varphi)}$

LTL formula φ

↓

Büchi automaton $\mathcal{A}_{\neg\varphi}$
words over $2^{AP(\varphi)}$ violating φ

↓

product Büchi automaton $\mathcal{B}$
$L(\mathcal{B}) = L(\mathcal{A}_M) \cap L(\mathcal{A}_{\neg\varphi})$

↓

$L(\mathcal{B}) \stackrel{?}{=} \emptyset$
e.g. nested DFS algorithm

YES

NO
+ counterexample

Complexity notes

Complexity

Time and space complexity of the LTL model checking algorithm is $\mathcal{O}(|M| \cdot 2^{\mathcal{O}(|\varphi|)})$, where $|M|$ is the number of states and transitions in the Kripke structure M.

- LTL model checking problem is PSPACE-complete.
- state explosion problem - $|M|$ is often exponential in the size of implicit description of the system due to
  - parallelism
  - large data domains
  - dynamically allocated memory
  - ...

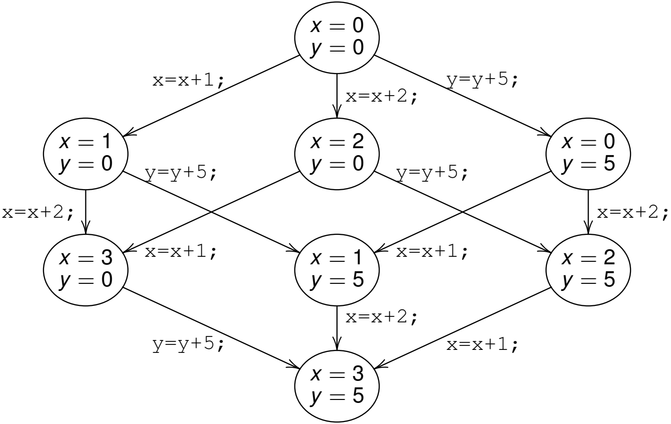
State explosion problem - an example

```
byte x = 0;
byte y = 0;

proctype A() {
   x = x + 1;
}

proctype B() {
   x = x + 2;
}

proctype C() {
   y = y + 5;
}
```



Partial solutions of the state explosion problem

- abstraction
- partial order reduction
- symmetry reduction
- on-the-fly algorithms
- symbolic model checking
- distributed algorithms
- ...

Our topics

- translation LTL$\rightarrow$BA (via alternating 1-weak BA)
- partial order reduction
- state-based LTL model checking of pushdown processes
- abstraction
- counterexample guided abstraction refinement (CEGAR)

Coming next week

LTL model checking of pushdown system

- How can I denote an infinite-state system?
- Can I verify an infinite-state system?
- What are pushdown processes good for?
- Can I do LTL model checking for them?