MODEL CHECKING
OF CONCURRENT SYSTEMS
- PART II -

Monika Heiner
BTU Cottbus,
Computer Science Institute

BASIC INGREDIENTS

- a language to model the system
  - formal semantics
  - many options, e.g.
    - Petri nets

- a language to specify model properties
  - temporal Logics,
  - several options, e.g.
    - Computational Tree Logic (CTL)

- an analysis approach to check a model against its properties
  - model checking,
  - various approaches (algorithms + data structures), e.g.
    - using reachability graph (RG)
    - labelled state transition system (STS) = Kripke structure
    - Continuous Time Markov Chain (CTMC)
ACM Turing Award = Nobel Prize of computing

goes to Model Checking Pioneers
Edmund M. Clarke, E. Allen Emerson and Joseph Sifakis

-> “For [their roles] in developing Model Checking
into a highly effective verification technology,
widely adopted in the hardware and software industries.”
**Motivation (1)**

- **General Properties (Requirements)**
  - Must be valid for any system, independently of its special functionality.
  - Boundedness
  - Liveness
  - Reversibility
  - . . .

- **Special Properties (Requirements)**
  - Reflect the special functionality.
  - (i) Insights into system behaviour
  - (c) Consistency properties to check the model’s integrity
  - (p) Progress properties - “something good will happen finally”
  - (s) Safety properties - “something bad never happens”
  - . . .

---

**Safety versus Functional Requirements**
**SPECIAL PROPERTIES, EXAMPLES**

- **insights (i)**
  - Is it possible that both robot arms hold a plate simultaneously?
  - How many plates can be concurrently inside the system?

- **consistency (c)**
  - The robot swivel is always positioned at exactly one angle.
  - At any time, a robot arm can be driven in one direction only.

- **progress (p)**
  - Any plate at the input position will finally reach the cell’s output position.

- **safety (s)**
  - To avoid machine collisions, the robot will only rotate with arms retracted.
  - The press will only be closed, when no robot arm is positioned inside.
  - The feed belt may only convey a blank through its light barrier, if the table is in its loading position.

---

**MOTIVATION (2)**

- Some properties can be expressed as un-/reachability of a given system state (marking)
  - Safety property -> unreachable
  - (weak) progress property -> reachable

- But, system states (marking) tend to be lengthy
  - Hard to read
  - Error-prone specification

- Most properties cannot be expressed as un-/reachability of a given system state (marking)

- Wanted: general query language to specify any special property
  - Flexible
  - Readable
  } **temporal logics**
How to specify logical statements with respect to the execution of a system?

**TEMPORAL LOGICS, INTRODUCTION**

```plaintext
int x, y;
read(x, y);

\[ x = y \rightarrow \text{FALSE} \]

\[ y := x; \]

\[ x = y \rightarrow \text{TRUE} \]

\[ y := x + 1; \]

\[ x = y \rightarrow \text{FALSE} \]
```

**TEMPORAL LOGICS, BASICS**

- extension of classical (propositional) logics by temporal operators
- atomic propositions
  - elementary statements, having - in a given state - a well-defined truth value
  - e.g. fork1, for 1-bounded Petri net (Boolean)
  - e.g. buffer = 2, buffer > 2, else (Integer)
- constants
  - TRUE, FALSE
- classical Boolean operators
  - negation \( ! \), conjunction \( * \)
  - disjunction \( + \), implication \( \rightarrow \)
- temporal operators
  - to refer to sequences of states
### CTL Operators, Interleaving Semantics

<table>
<thead>
<tr>
<th>Operator</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>next $f$</td>
<td>AX</td>
</tr>
<tr>
<td>finally $f$</td>
<td>AF</td>
</tr>
<tr>
<td>globally $f$</td>
<td>AG</td>
</tr>
<tr>
<td>$f_1$ until $f_2$</td>
<td>AU</td>
</tr>
</tbody>
</table>

#### Example 3 - Travel Planning

- **next** $v$ _vacation_
- **begin** _preparations_
- **accommodation** _open_
- **transportation** _means_ _open_
- **luggage** _open_
- **camping**
- **hotel**
- **ask** _for rooms_
- **phone**
- **check**
- **accommodation**
- **transportation** _means_
- **luggage**
- **end** _preparations_
- **ready**
- **repeat**
- **pack**
- **luggage**
- **feel**
- **transportation** _means_
- **accommodation** _open_
- **transportation** _means_ _open_
- **luggage** _open_
- **hotel**
- **check**
- **accommodation**
- **transportation** _means_
- **luggage**
- **end** _preparations_
- **ready**
- **repeat**
EXAMPLE 3 - SOME SIMPLE PROPERTIES (1)

- this is a possible combination of choices
  \( EF (hotel \ast car); \)  
  -> TRUE

- this is the only possible combination of choices
  \( AF (hotel \ast car); \)  
  -> FALSE

- this is an impossible combination of choices
  \( l \ EF (train \ast car); \)  
  -> TRUE

- any travel planning will be finished finally
  \( AG (next_vacation -> AF (ready)); \)  
  -> TRUE

- atomic use of the phone only,
  the phone is never reserved over several steps
  \( AG (phone); \)  
  -> TRUE

- liveness of transition end_preparations
  \( AG (EF (accommodation \ast transport\_means \ast luggage)); \)  
  -> TRUE

EXAMPLE 3 - SOME SIMPLE PROPERTIES (2)

- livelock freedom/progress of transition end_preparations
  \( AG (AF (accommodation \ast transport\_means \ast luggage)); \)  
  -> TRUE

- undone luggage will always be done in one step
  \( AG (luggage\_open -> AX (luggage)); \)  
  -> FALSE

- it’s possible that undone luggage will be done in one step
  \( AG (luggage\_open -> EX (luggage)); \)  
  -> TRUE

- undone luggage will be done finally
  \( AG (luggage\_open -> AF (luggage)); \)  
  -> TRUE

- luggage will remain undone until it is done
  \( A (luggage\_open U luggage); \)  
  -> FALSE

- if the luggage is undone, it remains undone until it is done
  \( AG (luggage\_open -> A (luggage\_open U luggage)); \)  
  -> TRUE
**Temporal Logics, Typical Questions**

- **reachability-related**  
  \( \text{EF} ( \varphi ) \)  
  - There exists at least one computational path to reach eventually a state, where \( \varphi \) will be true.

- **safety-related**  
  \( \text{AG} ( ! \varphi ) \)  
  - Equivalent to \( ! \text{EF} ( \varphi ) \)  
  - For every computational path, \( \varphi \) will never be true.

- **invariant-related**  
  \( \text{AG} ( \varphi ) \)  
  - Equivalent to \( ! \text{EF} ( ! \varphi ) \)  
  - For every computational path, \( \varphi \) will be true for ever.

- **liveness-related**  
  \( \text{AG} ( \text{EF} ( \varphi ) ) \)  
  - What ever happens, there exists the chance (at least one computational path) that \( \varphi \) will be true again.

- **progress-related**  
  \( \text{AG} ( \text{AF} ( \varphi ) ) \)  
  - For every computational path, \( \varphi \) will eventually be true.

**Model Checking, More Examples**

- **insights / reachability**  
  - Is it possible, that both robot arms carry a plate at the same time?  
    \( \text{EF} ( \text{arm1}\_\text{mag}\_\text{on} \ast \text{arm2}\_\text{mag}\_\text{on} ) \)

- **consistency property**  
  - The swivel is always either stopped or moves in exactly one direction.  
    \( \text{G} ( \text{robot}\_\text{stop} \text{xor} \text{robot}\_\text{left} \text{xor} \text{robot}\_\text{right} ) \)

- **safety property**  
  - If a robot arm is loaded, its magnet is not deactivated until the robot is in its unloading position.  
    \( \text{AG} ( \varphi \rightarrow \text{A} ( !\text{arm1}\_\text{mag}\_\text{off} \text{U} \psi ) ), \text{with} \)  
    \( \varphi = \text{arm1}\_\text{mag}\_\text{on} \ast \text{arm1}\_\text{pickup}\_\text{angle} \ast \text{arm1}\_\text{pickup}\_\text{ext} \)  
    \( \psi = \text{arm1}\_\text{release}\_\text{angle} \ast \text{arm1}\_\text{release}\_\text{ext} \)  
  - The feed belt may only convey a blank through its light barrier, if the table is in its loading position.  
    \( \text{G} ( \text{belt1}\_\text{light}\_\text{barrier}\_\text{true} \rightarrow (\text{table}\_\text{load}\_\text{angle} \ast \text{table}\_\text{bottom}\_\text{pos} ) ) \)
Bottleneck:
state explosion
(no primitive recursive function ...)

DINING PHILOSOPHERS 1000, BDD-BASED

Number of places/marked places/transitions: 7,000/2,000/5,000

Number of states:

\[
\begin{align*}
1137517608656205162806720354362767684058541876947800011092858232169918 \\
15995958121201133264112069097179071134074139603793701320514129462357710 \\
2442895227384242418853247239522943007188808619270527555972033293948691 \\
3344982712874090358789533181711372863591957907236895570937383074225421 \\
4932997350559348711208726085116502627818524644762991281238722816835426 \\
439043702222227167126998740049615901200930144497021663026892511861696 \\
792192797756430854076755677722422066045029462354355683154921949034887 \\
4138935108726115227535084646719457353408471086965332494805497753382942 \\
1717811011687720510211541690039211766279956422929032376885414750385275 \\
5124881924010536365251190474777411874
\end{align*}
\]

ca. $1.1 \times 10^{667}$

Number of places/marked places/transitions: 7000/2000/5000

Time to compute P-Invariants: 45885.66 sec

Number of P-Invariants: 3000

Time to compute compact coding: 385.59 sec

Number of Clusters: 3000

Number of Variables: 4000

Time: 3285.73 sec, ca. 54.75 min
POSSIBLE ANSWERS TO STATE EXPLOSION

- compositonal methods
  - simple interfaces between modules

- abstraction by ignoring some state information
  - conservative approximation

- different logics -> different algorithms, e.g. LTL

- integer programming

- compressed state space representations
  - BDD, IDD, ...

- lazy state space construction
  - partial order methods

- alternative state spaces
  - partial order representations, e.g. prefix

'BASE CASE' TECHNIQUES

'ALTERNATIVE’ METHODS (PN)

TEMPORAL LOGICS, OVERVIEW

<table>
<thead>
<tr>
<th>semantics</th>
<th>time</th>
<th>interleaving</th>
<th>partial order</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear (LTL)</td>
<td>traces (no conflict, no concurrency)</td>
<td>runs (no conflict, but concurrency)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Manna &amp; Pnueli, Kröger, jsp 2001</td>
<td>Reisig</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BDD/IDD-LTL, ...</td>
<td>tools: ?</td>
<td></td>
</tr>
<tr>
<td>branching (CTL)</td>
<td>reachability graph (conflict &amp; concurrency not distinguishable)</td>
<td>prefix (conflicts &amp; concurrency)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Emmerson, Clarke</td>
<td>McMillan, Esparza, pd 2001</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PROD, Charlie, BDD/IDD-CTL, ...</td>
<td>PEP</td>
<td></td>
</tr>
</tbody>
</table>
### Techniques & Tools, Overview

<table>
<thead>
<tr>
<th>Technique</th>
<th>CTL</th>
<th>LTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability graph</td>
<td>INA, Charlie</td>
<td>PROD, MARIA</td>
</tr>
<tr>
<td>Stubborn set reduced reachability graph</td>
<td>LoLA</td>
<td>PROD (LTL\X)</td>
</tr>
<tr>
<td>Symmetrically reduced reachability graph</td>
<td>LoLA (symmetric formulas)</td>
<td>?</td>
</tr>
<tr>
<td>BDD, NDD, ..., xDD</td>
<td>BDD-CTL, SMART, IDD-CTL</td>
<td>BDD-LTL</td>
</tr>
<tr>
<td>Kronecker algebra</td>
<td>[Kemper]</td>
<td>?</td>
</tr>
<tr>
<td>Prefix</td>
<td>PEP (CTL\0)</td>
<td>QQ (LTL\X)</td>
</tr>
<tr>
<td>Process automata</td>
<td>[pd]</td>
<td>?</td>
</tr>
</tbody>
</table>

### Production Cell, On-the-fly LTL Model Checking

<table>
<thead>
<tr>
<th>Requirement formula</th>
<th># states generated</th>
<th>Time effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 a</td>
<td>2259</td>
<td>4.16'</td>
</tr>
<tr>
<td>8 b</td>
<td>1775</td>
<td>3.76'</td>
</tr>
<tr>
<td>9</td>
<td>2305</td>
<td>4.34'</td>
</tr>
<tr>
<td>10</td>
<td>1879</td>
<td>3.10'</td>
</tr>
<tr>
<td>15</td>
<td>1184</td>
<td>2.55'</td>
</tr>
<tr>
<td>29</td>
<td>704</td>
<td>2.16'</td>
</tr>
<tr>
<td>30</td>
<td>703</td>
<td>2.46'</td>
</tr>
<tr>
<td>31 a</td>
<td>27104</td>
<td>23.80'</td>
</tr>
<tr>
<td>31 b</td>
<td>6433</td>
<td>4.02'</td>
</tr>
</tbody>
</table>

-> control model, 5 plates, full state space: 1,657,242  [7,185,779]
<table>
<thead>
<tr>
<th>case study</th>
<th>net size</th>
<th># of states</th>
</tr>
</thead>
<tbody>
<tr>
<td>production cell, 5 plates</td>
<td>231 P / 202 T</td>
<td>1.6 * 10^6</td>
</tr>
<tr>
<td>production cell, 3 plates</td>
<td></td>
<td>7.1 * 10^6</td>
</tr>
<tr>
<td>1000 dining philosophers</td>
<td>7,000 P / 5,000 T</td>
<td>1.1 * 10^667</td>
</tr>
<tr>
<td>solitaire, standard</td>
<td>65 P / 75 T</td>
<td>1.8 * 10^8</td>
</tr>
<tr>
<td>solitaire, non-standard</td>
<td>73 P / 91 T</td>
<td>2.9 * 10^9</td>
</tr>
<tr>
<td>Halobacterium, motor 44</td>
<td>38 P / 60 T</td>
<td>4.8 * 10^7</td>
</tr>
<tr>
<td>MAPK cascade, 80 levels</td>
<td>22 P / 30 T</td>
<td>5.6 * 10^18</td>
</tr>
<tr>
<td>MAPK cascade, 120 levels</td>
<td></td>
<td>1.7 * 10^21</td>
</tr>
<tr>
<td>gene regulation, 6 cells, m01</td>
<td>144 P / 307 T / 2146 A</td>
<td>7.4 * 10^21</td>
</tr>
<tr>
<td>gene regulation, 6 cells, m02</td>
<td></td>
<td>3.3 * 10^25</td>
</tr>
</tbody>
</table>

**SUMMARY MODEL CHECKING (1)**

- model checking can only be as good as the model specification
  - readable <-> unambiguous
  - complete <-> limited size

- correct model checking requires correct requirement specifications
  - take your time, think twice

- model checking proves consistency of
  - model & requirements

- if the answer is NO (-> FALSE)
  - the model can be wrong
  - the requirements can be wrong
  - or both

- if the answer is YES (-> TRUE)
  - model & requirements can still contain (same) logical faults
  - unlikely !?
MODEL CHECKING (2)

- (up to now) restricted to bounded systems
  - numerous ongoing research
  - CTL - undecidable
  - LTL - decidable, but no tools (not yet ?)
  - unboundedness + inhibitor arcs = undecidability

- model checking is extremely time and resource consuming
  - ’external’ quality pressure

- model checking is not manageable without theory supported by tools

- model checking needs knowledgeable professionals
  - study / job specialization
  - profession of “software validator”, “software tester”

- There is no such thing as a fault-free model (system)!
  - sufficient dependability for a given user profile

Model checking is no substitute for thinking!
REFERENCES, METHODS

- Pnueli, A.:

- Emerson, E. A.:

- MacMillan, K. L.:
  Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits; Proc. of the 4th Workshop on Computer Aided Verification, 164-174, 1992.

- Esparza, J.:

- EM Clarke, O Grumberg, and DA Peled:

REFERENCES, TOOLS

- J Spranger:

- Martin Schwarick:

- A Tovchigrechko:

- A Franzke:

- M Heiner, C Rohr and M Schwarick:

- M Heiner, C Rohr, M Schwarick and A Tovchigrechko:
  MARCIE’s secrets of efficient model checking; Trans. on Petri Nets and Other Models of Concurrency XI, LNCS 9930, 286-296, 2016.
REFERENCES, SOME CASE STUDIES


Thanks!

http://www-dssz.informatik.tu-cottbus.de