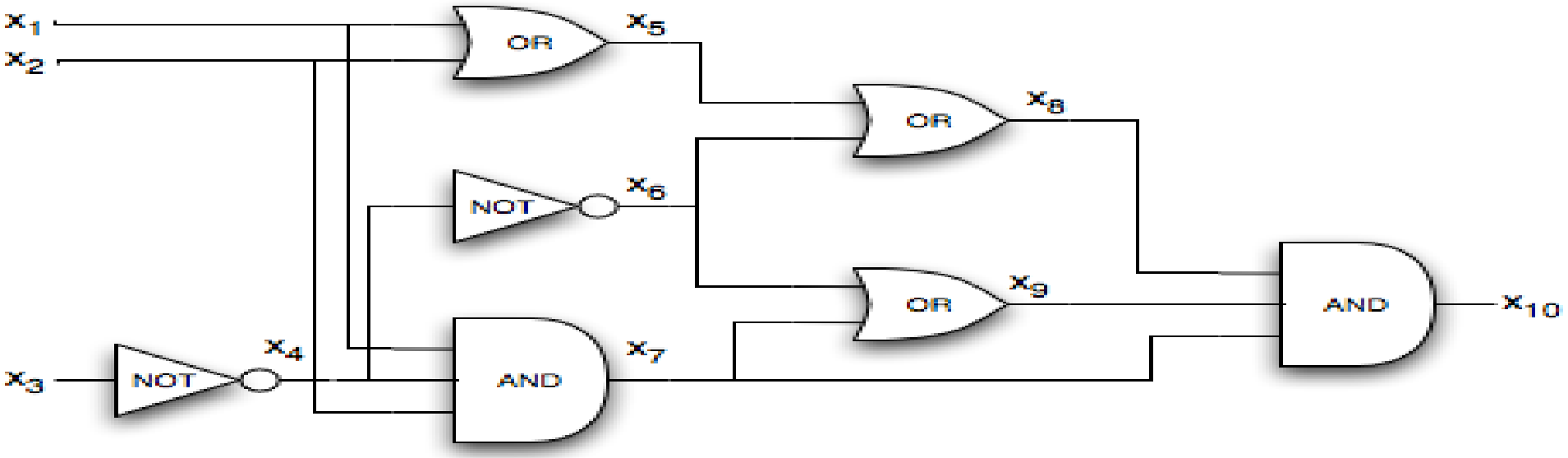
Karlsruhe Institute of Technology

Practical SAT Solving

Lecture 12

Carsten Sinz, Tomáš Balyo | July 18, 2016

INSTITUTE FOR THEORETICAL COMPUTER SCIENCE



Lecture Outline

- Bounded Model Checking
- Satisfiability Modulo Theory (SMT)

Model Checking: Motivation

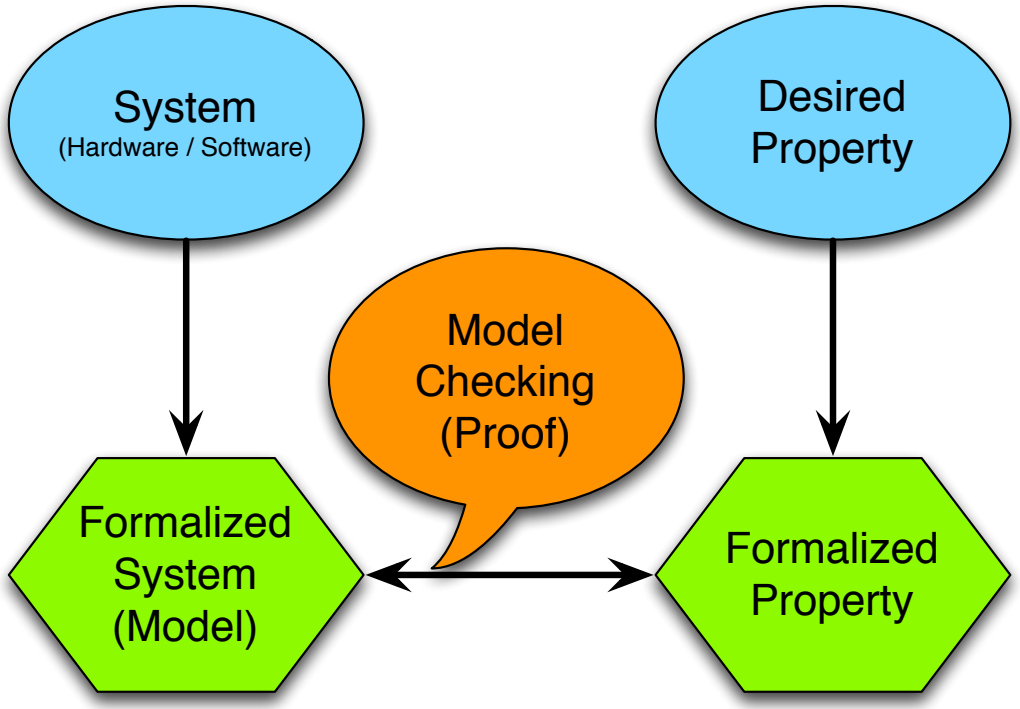
"Buggy smartphone software is the new reality of making complex cellphones in large volumes."

— RIM-CEO Jim Balsillie about faulty software of Blackbery Storm (Jan. 2009)

"(...) Software verification (...) has been the Holy Grail of computer science for many decades, but now in some very key areas, for example driver verification, were building tools that can do actual proofs about the software and how it works in order to guarantee the reliability."

— Bill Gates, WinHEC 2002

Model Checking



Example: Zune Z2K9

It seems that a random bug is affecting a bunch, if not every, 30GB Zunes. Real early this morning, a bunch of Zune 30s just stopped working.

— TechCrunch, Dec. 31, 2008

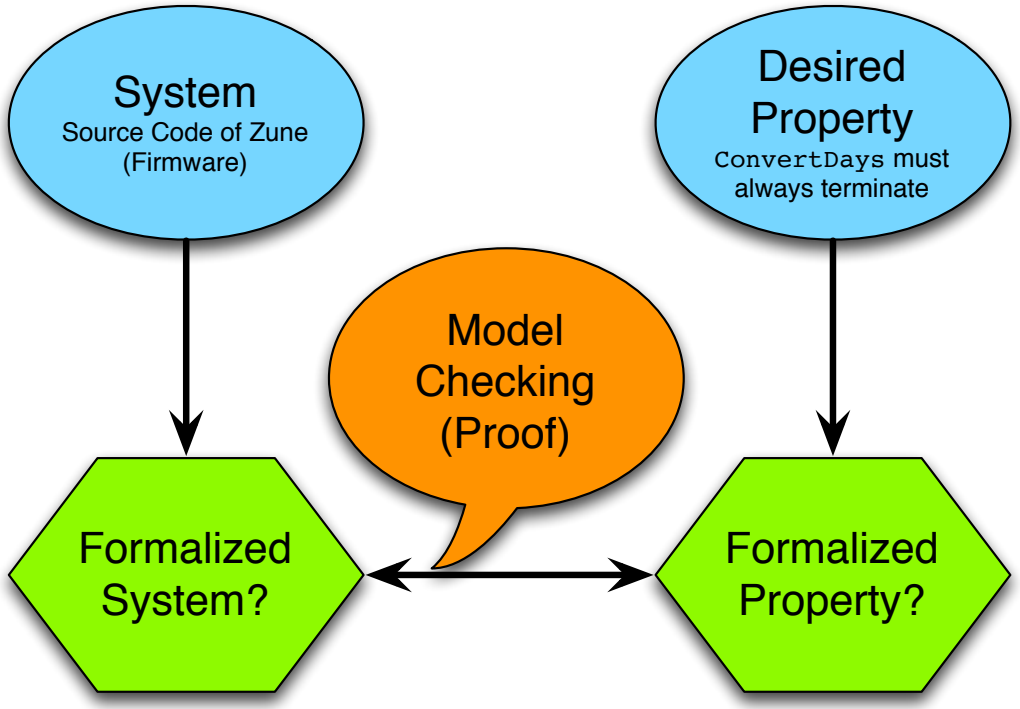
Microsoft responded to the Zune 30GB failure, blaming a leap-year handling bug. And they've provided a fix. Which is to wait til New Years, when the bug will go away by itself. Huh.

— Gizmodo, Dec. 31, 2008

Z2K9: Source Code

```
BOOL ConvertDays(UINT32 days, SYSTEMTIME* lpTime)
{
  int dayofweek, month, year;
  ...
  year = ORIGINYEAR;
  while (days > 365) {
    if (IsLeapYear(year)) {
      if (days > 366) {
        days -= 366;
        year += 1;
      }
    } else {
      days -= 365;
      year += 1;
    }
  }
  ...
```

Model Checking: Z2K9 Example



Kripke Structure

Definition: Kripke structure

Let A be a set of propositional variables. A Kripke structure M over A is a tuple $M = (S, I, T, L)$, where

- S: set of states (finite)
- $I \subseteq S$: set of initial states
- $T : S \times S$: (total) transition relation (total: for each state s there is an s' with $T(s, s')$)
- $L : S \to \mathcal{P}(A)$: labelling function, indicating which propositions are true in a state

Kripke structures are used to describe systems.

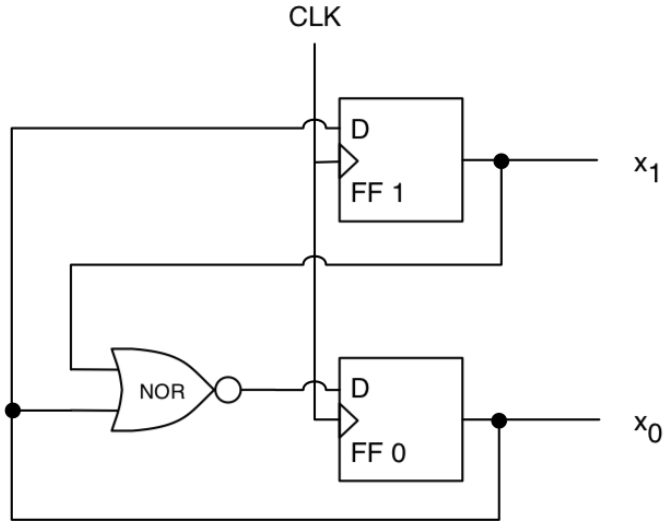
Kripke Structure: Paths

Definition: Path

A path π in a Kripke structure $M = (S, I, T, L)$ is an infinite sequence of states, $\pi = s_0 s_1 s_2 \ldots$, such that $s_0 \in I$ and $T(s_i, s_{i+1})$ for all $i \geq 0$.

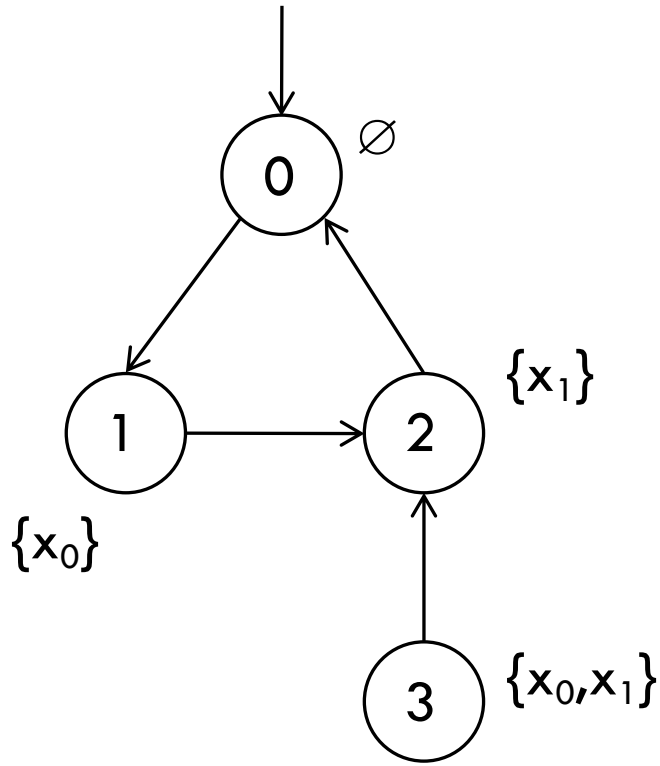
Example: Circuit as Kripke Structure

- Kripke structures can be used to model hardware circuits
- Example (see right): 2-bit counter, counting continuously from 0 to 2.



Example: 2-Bit Counter

- Kripke structure $M_{\texttt{counter}}$ over $A = \{x_0, x_1\}$
- Path(s) from state 0: $\pi_0 = 0, 1, 2, 0, 1, 2, \ldots$
- Safety property: state 3 is never reached



Transition Relation as Formula

- Initial state(s): $I(x_0, x_1) = \neg x_0 \wedge \neg x_1$
- Transition relation: $T(x_0, x_1, x_0', x_1') = (x_0' \Leftrightarrow \neg(x_0 \vee x_1)) \wedge (x_1' \Leftrightarrow x_0)$
- Error states: (for which the safety property is violated):
  $E(x_0, x_1) = x_0 \wedge x_1$

Bounded Model Checking

- Ensures that on all paths *up to length k*, the safety property holds.
- Encoding (as validity property):

$$I(s_0) \wedge \bigwedge_{i=0}^{i<k} T(s_i, s_{i+1}) \quad \Rightarrow \quad \bigwedge_{i=0}^{i\leq k} \neg E(s_i)$$

- As SAT problem:

$$I(s_0) \wedge \bigwedge_{i=0}^{i<k} T(s_i, s_{i+1}) \quad \wedge \quad \bigvee_{i=0}^{i\leq k} E(s_i)$$

- Algorithm: Search for increasing k for a state that violates the safety property.

Temporal Logic LTL: Semantics

$$
\begin{aligned}
M, \pi \models p \quad &\Leftrightarrow \quad p \in L(s_0) \\
M, \pi \models \neg f_1 \quad &\Leftrightarrow \quad M, \pi \not\models f_1 \\
M, \pi \models f_1 \vee f_2 \quad &\Leftrightarrow \quad M, \pi \models f_1 \text{ or } M, \pi \models f_2 \\
M, \pi \models f_1 \wedge f_2 \quad &\Leftrightarrow \quad M, \pi \models f_1 \text{ and } M, \pi \models f_2 \\
M, \pi \models \mathbf{X} f_1 \quad &\Leftrightarrow \quad M, \pi^1 \models f_1 \\
M, \pi \models \mathbf{F} f_1 \quad &\Leftrightarrow \quad \text{there is a } k \geq 0, \text{ such that } M, \pi^k \models f_1 \\
M, \pi \models \mathbf{G} f_1 \quad &\Leftrightarrow \quad \text{for all } k \geq 0\text{: } M, \pi^k \models f_1 \\
M, \pi \models f_1 \mathbf{U} f_2 \quad &\Leftrightarrow \quad \text{there is a } k \geq 0, \text{ such that } M, \pi^k \models f_2, \text{ and} \\
&\qquad \text{for all } 0 \leq j < k\text{: } M, \pi^j \models f_1 \\
M, \pi \models f_1 \mathbf{R} f_2 \quad &\Leftrightarrow \quad \text{for all } k \geq 0\text{: if for each } j < k \; M, \pi^j \models f_1, \\
&\qquad \text{then } M, \pi^k \models f_2
\end{aligned}
$$

What is SMT?

- Extend propositional logic by certain theories of first-order logic.
- Theories are mainly about arithmetic (linear integer arithmetic, non-linear real arithmetic, etc.)
- Example: $x + y < 5 \wedge (2x - y > 4 \vee x + y > 7)$
- In most cases quantifier-free fragment of first-order logic (because of decidability).
- Syntax and semantics of SMT standardized in SMT-LIB standard.

SMT-LIB

SMT-LIB is an international initiative aimed at facilitating research and development in Satisfiability Modulo Theories (SMT). Since its inception in 2003, the initiative has pursued these aims by focusing on the following concrete goals.

- Provide standard rigorous descriptions of background theories used in SMT systems.
- Develop and promote common input and output languages for SMT solvers.
- Connect developers, researchers and users of SMT, and develop a community around it.
- Establish and make available to the research community a large library of benchmarks for SMT solvers.
- Collect and promote software tools useful to the SMT community.

This website provides access to the following main artifacts of the initiative.

- Documents describing the SMT-LIB input/output language for SMT solvers and its semantics;
- Specifications of background theories and *logics*;
- A large library of input problems, or benchmarks, written in the SMT-LIB language.
- Links to SMT solvers and related tools and utilities.

Home
About
News
Standard
- Language
- Theories
- Logics
- Examples

Benchmarks
Software
- Solvers
- Utilities

Contact
Related
Credits

SMT-LIB: Logics