Satisfiability Checking

SMT Applications

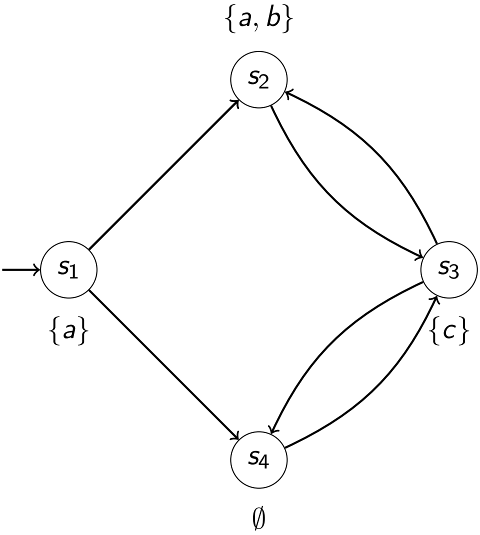
Prof. Dr. Erika Ábrahám

RWTH Aachen University
Informatik 2
LuFG Theory of Hybrid Systems

WS 14/15

Overview

Kripke structures



Kripke structure: Syntax

Definition

Let AP be a finite set of atomic propositions. A Kripke structure is a tuple $M = (S, s_{\text{init}}, T, L)$ with

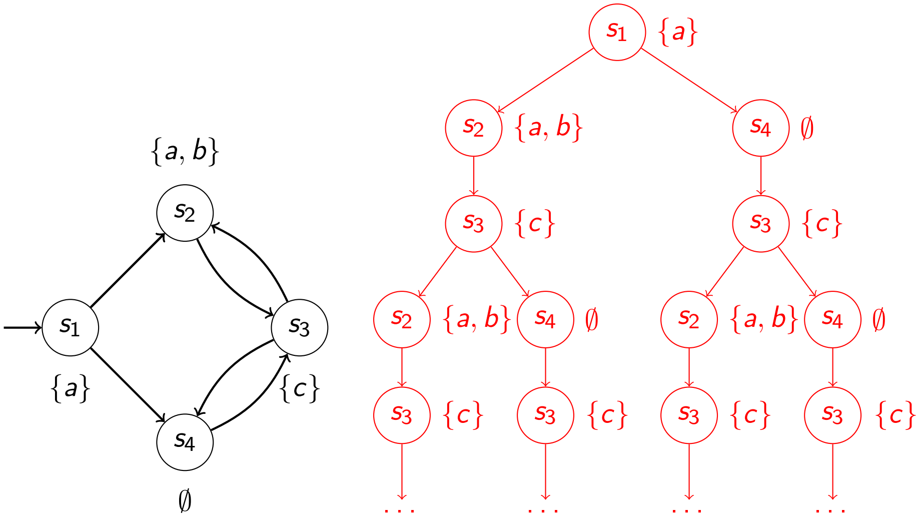
- S a finite set of states,
- $s_{\text{init}} \in S$ an initial state,
- $T \subseteq S \times S$ a transition relation,
- $L : S \to 2^{AP}$ a labeling function
  (2^{AP} denotes the powerset over AP).

The labeling function attaches information to the system: for a state $s \in S$ the set $L(s)$ consists of those atomic propositions that hold in s.

Kripke structure: Semantics

- An (infinite) path $\pi = s_0 s_1 s_2 \ldots$ of a Kripke structure $M = (S, s_{\text{init}}, T, L)$ is a sequence of states such that
  - $s_0 = s_{\text{init}}$ and
  - $(s_i, s_{i+1}) \in T$ for all $i \geq 0$.
- The behaviour of M is given by the set of all of its infinite paths.
- A finite path of M is a finite prefix of an infinite path of M.
- For a finite path $\pi = s_0 \ldots s_k$ we define $|\pi| = k$.
- We write $\pi(j)$ for the jth state (starting with 0) of the path π.
- By π_j we denote the postfix of π starting at $\pi(j)$.

Kripke structure: Semantics

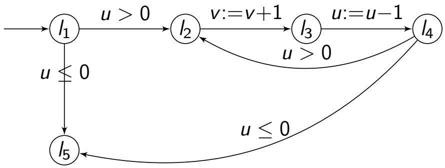


Transition systems

```
     m(int u, int v){
l1:  while (u>0) do
l2:    v:=v+1;
l3:    u:=u-1;
l4:  od
l5:  return v
     }
```

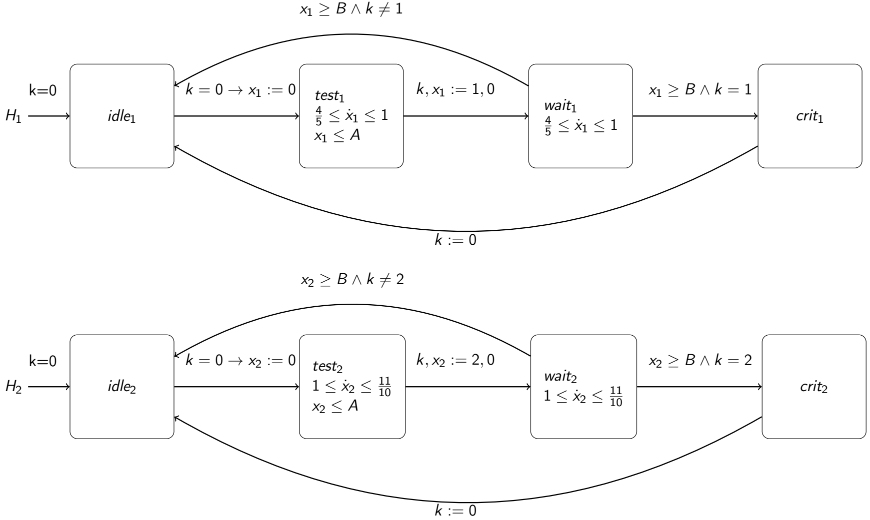
Transition systems

```
      m(int u, int v){
l1:   while (u>0) do
l2:     v:=v+1;
l3:     u:=u-1;
l4:   od
l5:   return v
      }
```



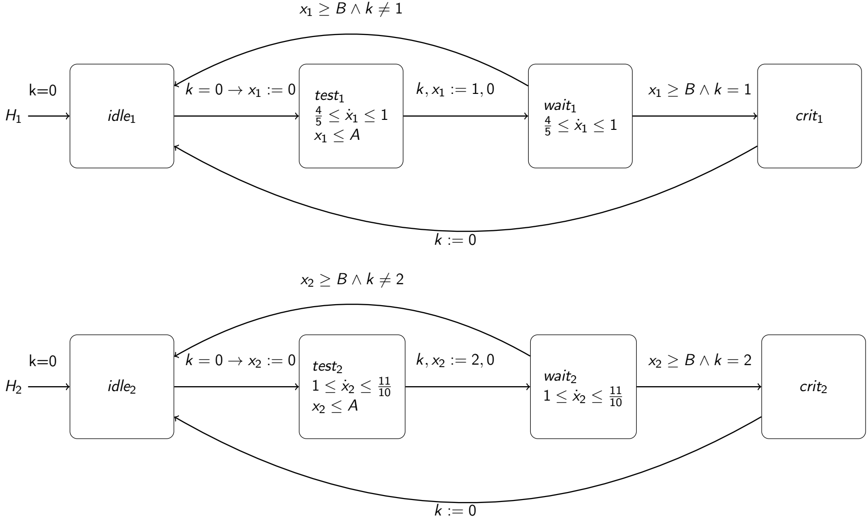
Fischer's mutual exclusion protocol

There are also more complex systems we can deal with similarly...



Fischer's mutual exclusion protocol

There are also more complex systems we can deal with similarly...



...but now we focus of Kripke structures and discrete transition systems.

Model checking

Early 1980s: First implementations of model checking as verification technique

- Explicit representations of the transition graphs
- Problem: Due to the state space explosion not applicable for complex industrial settings

1990: Symbolic model checking

- BDDs represent *characteristic functions* of state sets symbolically
- Problem: Building the BDD may be expensive

1999: Bounded model checking *[Biere et al.]*

- Check the existence of finite paths of incremental length by a SAT- or SMT-solver
- Problem: Incomplete (in general)
- Works for different logics, we consider only reachability

Model checking and counterexamples

- Given a model M and a logical description φ of safe states (invariant), a counterexample is a finite path of M reaching a non-φ state.
- If a system is buggy, counterexamples are extremely important for detecting and fixing the error.
- Bounded model checking (BMC) is a technique to search for finite counterexamples (also for more complex models and logics).

Bounded model checking

Algorithm:

1. Set $k = 0$
2. Construct a logical formula BMC_k describing a finite path
   - through the underlying model
   - of length k,
   - and violating a safety property φ after k steps.
3. Check BMC_k for satisfiability using a SAT or SMT solver
4. If SAT, the resulting assignment describes a counterexample $\rightarrow$ terminate
5. If UNSAT, increment k, goto 1.

Bounded model checking

$$BMC_k = I(s_0) \wedge T(s_0, s_1) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge \neg\varphi(s_k)$$

- I and T are (nearly) straightforward for Kripke structures and timeless transition systems
- $I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$ is called the unfolding of the transition relation

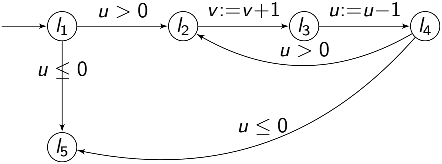
BMC example

```
     m(int u, int v){
l1:  while (u>0) do
l2:    v:=v+1;
l3:    u:=u-1;
l4:  od
l5:  return v
     }
```

Safety property: Return value = sum of the input values

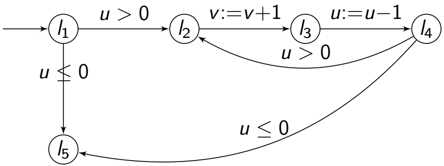
BMC example

```
     m(int u, int v){
l1:  while (u>0) do
l2:    v:=v+1;
l3:    u:=u-1;
l4:  od
l5:  return v
     }
```



Safety property: Return value = sum of the input values

```
     m(int u, int v){
l1:  while (u>0) do
l2:    v:=v+1;
l3:    u:=u-1;
l4:  od
l5:  return v
     }
```

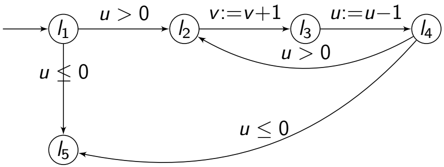


Safety property: Return value = sum of the input values

BMC for counterexamples: (encoding location l_j in step i by $at_i = j$)

BMC example

```
     m(int u, int v){
l1:  while (u>0) do
l2:    v:=v+1;
l3:    u:=u-1;
l4:  od
l5:  return v
     }
```



Safety property: Return value = sum of the input values

BMC for counterexamples: (encoding location l_j in step i by $at_i = j$)

$$
\begin{array}{ll}
I(i): & at_i = 1 \\
T_{1,2}(i, i+1): & at_i = 1 \wedge at_{i+1} = 2 \wedge u_i > 0 \wedge u_{i+1} = u_i \wedge v_{i+1} = v_i \\
T_{1,5}(i, i+1): & at_i = 1 \wedge at_{i+1} = 5 \wedge u_i \leq 0 \wedge u_{i+1} = u_i \wedge v_{i+1} = v_i \\
\ldots & \\
T(i, i+1): & T_{1,2}(i, i+1) \vee T_{1,5}(i, i+1) \vee \ldots \\
\varphi(i): & at_i \neq 5 \vee v_i = v_0 + u_0 \\
BMC_0: & I(0) \wedge \neg\varphi(0) \qquad \mathit{UNSAT} \\
BMC_0: & I(0) \wedge T(0,1) \wedge \neg\varphi(1) \qquad \mathit{SAT}
\end{array}
$$

BMC is not complete

- Termination is guaranteed iff counterexample exists
- If no counterexample exists, procedure does not terminate
- Upper bound for k to ensure property: Completeness threshold
- Another possibility to make BMC complete: k-induction (not content of this lecture)

Completeness threshold

- For each (finite state) system M, property p and given translation scheme there exists a number $\mathcal{CT}$, called completeness threshold.
- For reachability properties, $\mathcal{CT}$ is equal to the reachability diameter, i.e., the minimal distance required to reach all (reachable) states of the system.

Completeness threshold

- For each (finite state) system M, property p and given translation scheme there exists a number $\mathcal{CT}$, called completeness threshold.
- For reachability properties, $\mathcal{CT}$ is equal to the reachability diameter, i.e., the minimal distance required to reach all (reachable) states of the system.

Definition (Reachability Diameter)

$$rd(M) := min\Big\{ i \mid \forall n > i.\ \forall s_0, \ldots, s_n.\ \exists t \leq i.\ \exists s'_0, \ldots, s'_t.$$

$$\left(I(s_0) \wedge \bigwedge_{j=0}^{n-1} T(s_j, s_{j+1}) \right) \rightarrow \left(I(s'_0) \wedge \bigwedge_{j=0}^{t-1} T(s'_j, s'_{j+1}) \wedge s'_t = s_n \right) \Big\}$$

"Every state that is reachable in n steps, is also reachable in i steps."

- This yields maximal shortest paths in the system.

Completeness threshold

- Problem: Show the property for all n
- For V the set of variables defining the states, $n = i + 1, \ldots, 2^{|V|}$!

Completeness threshold

- Problem: Show the property for all n
- For V the set of variables defining the states, $n = i + 1, \ldots, 2^{|V|}$!
- Solution: $n = i + 1$ suffices

Completeness threshold

- Problem: Show the property for all n
- For V the set of variables defining the states, $n = i + 1, \ldots, 2^{|V|}$!
- Solution: $n = i + 1$ suffices

Definition (Reachability Diameter)

$$rd(M) := min\Big\{ i \mid \forall s_0, \ldots, s_{i+1}.\ \exists s'_0, \ldots, s'_i.$$

$$\left(I(s_0) \wedge \bigwedge_{j=0}^{i} T(s_j, s_{j+1}) \right) \rightarrow \left(I(s'_0) \wedge \bigwedge_{j=0}^{i-1} T(s'_j, s'_{j+1}) \wedge \bigvee_{j=0}^{i} s'_j = s_{i+1} \right) \Big\}$$

"Every state that is reachable in $i + 1$ steps, is also reachable in at most i steps."

Completeness threshold

- Problem: Formula contains quantifier alternation

Completeness threshold

- Problem: Formula contains quantifier alternation
- Solution: Over-approximation of $rd(M)$

Completeness threshold

- Problem: Formula contains quantifier alternation
- Solution: Over-approximation of $rd(M)$

Definition (Recurrence Diameter)

$rdr(M) :=$

$$max\left\{ i \mid \exists s_0 \dots s_i : I(s_0) \wedge \bigwedge_{j=0}^{i-1} T(s_j, s_{j+1}) \wedge \bigwedge_{j=0}^{i-1} \bigwedge_{k=j+1}^{i} s_j \neq s_k \right\}$$

"Longest loop-free initial path in M."

- As every shortest path is a loop-free path, this is an over-approximation of $rd(M)$.

Overview

Deductive verification: Example

Overview

Schere, Stein, Papier

Overview

Hardware design