Department of Mathematics, Computer Science and Physics, University of Udine

The Safety Fragment of Temporal Logics on Infinite Sequences

Lesson 9

Luca Geatti
luca.geatti@uniud.it
Angelo Montanari
angelo.montanari@uniud.it

May 2nd, 2024

Outline

1. Background
   1. Regular and ω-regular languages
   2. The First- and Second-order Theory of One Successor
   3. Automata over finite and infinite words
   4. Linear Temporal Logic
2. The safety fragment of LTL and its theoretical features
   1. Definition of Safety and Cosafety
   2. Characterizations and Normal Forms
   3. Kupferman and Vardi's Classification

Outline

SYMBOLIC MODEL CHECKING

Tackling the state-space explosion problem

Symbolic Model Checking

- The previous algorithms belongs to the class of explicit-state model checking algorithms:
  - the Kripke Structure M is represented as a set of memory locations, pointers ecc...
- MC suffers from the state-space explosion problem: the number of states of

$$M = M_1 \times M_2 \times \cdots \times M_n$$

is exponential in n;
- the size of system that could be verified by explicit model checkers was restricted to $\approx 10^6$ states.
- Solution: Symbolic Model Checking

Citation for the 2007 Turing Award

Although the 1981 paper demonstrated that the model checking was possible in principle, its application to practical systems was severely limited. The most pressing limitation was the number of states to search. Early model checkers required explicitly computing every possible configuration of values the program might assume. For example, if a program counts the millimeters of rain at a weather station each day of the week, it will need 7 storage locations. Each location will have to be big enough to hold the largest rain level expected in a single day. If the highest rain level in a day is 1 meter, this simple program will have 10^{21} possible states, slightly less than the number of stars in the observable universe. Early model checkers would have to verify that the required property was true for every one of those states.

`https://amturing.acm.org/award_winners/clarke_1167964.cfm`

Symbolic Transition Systems

Consider a (explicit) Kripke structure $\mathcal{M} = (S, I, T, L)$. We can give a Boolean encoding of it:

Symbolic Transition Systems

Consider a (explicit) Kripke structure $\mathcal{M} = (S, I, T, L)$. We can give a Boolean encoding of it:

- let $\bar{s} := \{s_0, \ldots, s_n\}$ be a set of state (Boolean) variables;

Consider a (explicit) Kripke structure $\mathcal{M} = (S, I, T, L)$. We can give a Boolean encoding of it:

- let $\bar{s} := \{s_0, \ldots, s_n\}$ be a set of state (Boolean) variables;
- $S = \{0,1\}^n$, *i.e.*, a state is an assignment to all the state variables;

with n variables we represent 2^n states

Symbolic Transition Systems

Consider a (explicit) Kripke structure $\mathcal{M} = (S, I, T, L)$. We can give a Boolean encoding of it:

- let $\bar{s} := \{s_0, \ldots, s_n\}$ be a set of state (Boolean) variables;
- $S = \{0, 1\}^n$, *i.e.*, a state is an assignment to all the state variables;

  with n variables we represent 2^n states
- $m \models f_I(\bar{s})$ is true iff $m \in I$

Symbolic Transition Systems

Consider a (explicit) Kripke structure $\mathcal{M} = (S, I, T, L)$. We can give a Boolean encoding of it:

- let $\bar{s} \coloneqq \{s_0, \ldots, s_n\}$ be a set of state (Boolean) variables;
- $S = \{0, 1\}^n$, *i.e.*, a state is an assignment to all the state variables;

  with n variables we represent 2^n states
- $m \models f_I(\bar{s})$ is true iff $m \in I$
- $m, m' \models f_T(\bar{s}, \bar{s}')$ is true iff $(m, m') \in T$

Symbolic Transition Systems

Consider a (explicit) Kripke structure $\mathcal{M} = (S, I, T, L)$. We can give a Boolean encoding of it:

- let $\bar{s} := \{s_0, \ldots, s_n\}$ be a set of state (Boolean) variables;
- $S = \{0, 1\}^n$, *i.e.*, a state is an assignment to all the state variables;

  with n variables we represent 2^n states
- $m \models f_I(\bar{s})$ is true iff $m \in I$
- $m, m' \models f_T(\bar{s}, \bar{s}')$ is true iff $(m, m') \in T$
- $m \models f_p(\bar{s})$ is true iff $p \in L(m)$, for all labels $p \in RANGE(L)$

Symbolic Transition Systems

Consider a (explicit) Kripke structure $\mathcal{M} = (S, I, T, L)$. We can give a Boolean encoding of it:

- let $\bar{s} := \{s_0, \ldots, s_n\}$ be a set of state (Boolean) variables;
- $S = \{0, 1\}^n$, *i.e.*, a state is an assignment to all the state variables;

  with n variables we represent 2^n states
- $m \models f_I(\bar{s})$ is true iff $m \in I$
- $m, m' \models f_T(\bar{s}, \bar{s}')$ is true iff $(m, m') \in T$
- $m \models f_p(\bar{s})$ is true iff $p \in L(m)$, for all labels $p \in RANGE(L)$

The corresponding symbolic Kripke structure is the tuple $(\bar{s}, f_I, f_T, \{f_{p_1}, \ldots, f_{p_k}\})$.

Symbolic Transition Systems

- we will write simply $\mathcal{M} = (S, I, T, L)$, meaning a symbolic transition system
- a path (or trace) $\pi = m_0, m_1, \ldots$ is an infinite sequence of assignment to the state variables such that:
  - $m_0 \models I(\bar{s})$;
  - $m_i, m'_{i+1} \models T(\bar{s}, \bar{s}')$ holds, for all $i \geq 0$.

  where $\bar{s}' \coloneqq \{s'_0, \ldots, s'_n\}$.

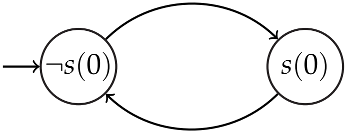
Symbolic Transition Systems

Three main techniques have been proposed:

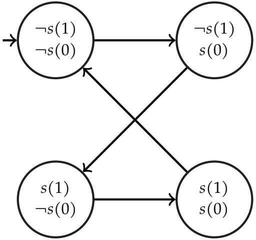
- partial order reduction
- BDD-based symbolic model checking
    - kind of *compressed truth tables*
- SAT-based symbolic model checking, aka *Bounded Model Checking*.

They allowed for the verification of systems with $> 10^{120}$ states.

- substantially larger than the number of atoms in the observable universe (around 10^{80})




```
MODULE main
VAR
  s0 : boolean;
INIT
  !s0;
TRANS
  s0 <-> next(!s0);
```




```
MODULE main
VAR
   s0 : boolean;
   s1 : boolean;
INIT
   !s0 & !s1;
TRANS
   (next(s0) <-> !s0)
     &
   (next(s1) <-> ((s0 & !s1) | (!s0
    & s1)));
```

Example 3 - SMV

```
while true do
  if x < 200 then
    x := x + 1
od
```

```
while true do
  if x > 0 then
    x := x-1
od
```

```
while true do
  if x = 200 then
    x := 0
od
```

```
MODULE main
VAR
  x    : 0 .. 200;
INIT
  x = 199;
TRANS
  (x<200 & next(x)=x+1) |
  (x>0 & next(x)=x+(-1)) |
  (x=200 & next(x)=0);
```

BOUNDED MODEL CHECKING

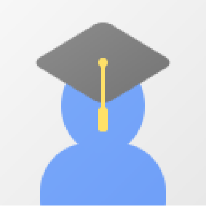
Reference:

Armin Biere et al. (1999). "Symbolic model checking without BDDs". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, pp. 193–207

(a) A. Biere

(b) A. Cimatti

(c) E. Clarke



(d) Y. Zhu

The SAT problem

- given a Boolean formula f, establish if f is satisfiable;

The SAT problem

- given a Boolean formula f, establish if f is satisfiable;
- f is normally given in CNF:

$$f := (L_{1,1} \vee \cdots \vee L_{1,k}) \wedge \cdots \wedge (L_{n,1} \vee \cdots \vee L_{n,m})$$

where each literal $L_{i,j}$ is either a variable or a negation of a variable.

The SAT problem

- given a Boolean formula f, establish if f is satisfiable;
- f is normally given in CNF:

$$f := (L_{1,1} \vee \cdots \vee L_{1,k}) \wedge \cdots \wedge (L_{n,1} \vee \cdots \vee L_{n,m})$$

  where each literal $L_{i,j}$ is either a variable or a negation of a variable.
- why not in DNF?

$$f := (L_{1,1} \wedge \cdots \wedge L_{1,k}) \vee \cdots \vee (L_{n,1} \wedge \cdots \wedge L_{n,m})$$

- first NP-complete problem, but ...

The SAT problem

- first NP-complete problem, but ...
- there are several efficient algorithms for solving SAT (*e.g.*, DPLL, CDCL...) along with many heuristics (*e.g.*, 2 watching literals, glue clauses...)

The SAT problem

- first NP-complete problem, but ...
- there are several efficient algorithms for solving SAT (*e.g.*, DPLL, CDCL...) along with many heuristics (*e.g.*, 2 watching literals, glue clauses...)
- some numbers:
  - $>$ 100˙000 variables;
  - $>$ 1˙000˙000 clauses;

- recall that we can reduce $\mathcal{M}, s \models \psi$ to checking the emptiness of $\mathcal{M} \times \mathcal{A}_{\neg\psi}$;

- recall that we can reduce $\mathcal{M}, s \models \psi$ to checking the emptiness of $\mathcal{M} \times \mathcal{A}_{\neg\psi}$;
  - the universal problem $\mathcal{M}, s \models A\psi$ is reduced to the existential problem $\mathcal{M}, s \models E\phi$, where $\phi := \neg\psi$;

- recall that we can reduce $\mathcal{M}, s \models \psi$ to checking the emptiness of $\mathcal{M} \times \mathcal{A}_{\neg\psi}$;
  - the universal problem $\mathcal{M}, s \models A\psi$ is reduced to the existential problem $\mathcal{M}, s \models E\phi$, where $\phi := \neg\psi$;
- Bounded Model Checking (BMC) solves the problem $\mathcal{M}, s \models E\phi$ by proceeding incrementally:
  - we start with $k = 0$;
  - check if there exists and execution π of $\mathcal{M}$ of length k that satisfies ϕ; encode this problem into a SAT instance and call a SAT-solver;
  - if so, we have found a counterexample to ψ; if not, $k{+}{+}$.

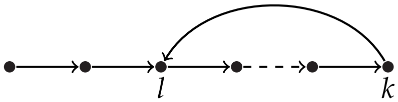
Loop-backs

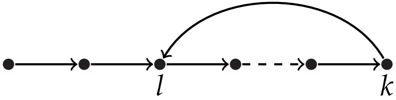
- BMC checks only bounded/finite traces of the system;
- ...but LTL formulas are defined over infinite state sequences;

- BMC checks only bounded/finite traces of the system;
- ...but LTL formulas are defined over infinite state sequences;

Crucial observation:

- a finite trace can still represent an infinite state sequence, if it contains a loop-back.





Definition (k-loop)

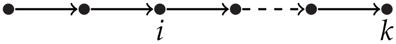
A path π is a (k, l)-loop, with $l \leq k$, if $T(\pi(k), \pi(l))$ holds and $\pi = u \cdot v^\omega$, where:

- $u = \pi(1) \dots \pi(l-1)$;
- $v = \pi(l) \dots \pi(k)$.

We call π a k-loop if there exists $l \leq k$ for which π is a (k, l)-loop.

Given a finite trace π of the system $\mathcal{M}$, BMC distinguishes between two cases:

- either π contains a loop-back (π is lasso-shaped):
  - $\Rightarrow$ apply standard LTL semantics to check if $\pi \models \phi$;
- or π is loop-free:
  - $\Rightarrow$ apply bounded semantics
  - $\Rightarrow$ if a path is a model of ϕ under bounded semantics then any extension of the path is a model of ϕ under standard semantics (conservative semantics)



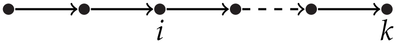
If π is not a k-loop, we introduce bounded semantics for LTL.

Definition (Bounded semantics for LTL)

Let $k \geq 0$ and π a path that is not a k-loop. An LTL formula ϕ is valid along π with bound k, written $\pi \models_k^0 \phi$, iff:

- $\pi \models_k^i p$ iff $p \in L(\pi(i))$
- $\pi \models_k^i \neg p$ iff $p \notin L(\pi(i))$

Bounded Semantics for LTL



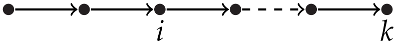
If π is not a k-loop, we introduce bounded semantics for LTL.

Definition (Bounded semantics for LTL)

Let $k \geq 0$ and π a path that is not a k-loop. An LTL formula ϕ is valid along π with bound k, written $\pi \models_k^0 \phi$, iff:

- $\pi \models_k^i \phi_1 \vee \phi_2$ iff $\pi \models_k^i \phi_1$ or $\pi \models_k^i \phi_2$
- $\pi \models_k^i \phi_1 \wedge \phi_2$ iff $\pi \models_k^i \phi_1$ and $\pi \models_k^i \phi_2$

Bounded Semantics for LTL



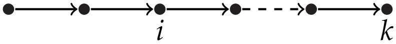
If π is not a k-loop, we introduce bounded semantics for LTL.

Definition (Bounded semantics for LTL)

Let $k \geq 0$ and π a path that is not a k-loop. An LTL formula ϕ is valid along π with bound k, written $\pi \models_k^0 \phi$, iff:

- $\pi \models_k^i \mathsf{X}\phi_1$ iff $i < k$ and $\pi \models_k^{i+1} \phi_1$
- $\pi \models_k^i \phi_1 \mathsf{U} \phi_2$ iff $\exists i \leq j \leq k$ such that $\pi \models_k^j \phi_2$ and $\forall i \leq n < j$ it holds that $\pi \models_k^n \phi_1$

Bounded Semantics for LTL

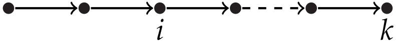


If π is not a k-loop, we introduce bounded semantics for LTL.

Definition (Bounded semantics for LTL)

Let $k \geq 0$ and π a path that is not a k-loop. An LTL formula ϕ is valid along π with bound k, written $\pi \models_k^0 \phi$, iff:

- $\pi \models_k^i \mathsf{G}\phi_1$ iff ???
- $\pi \models_k^i \mathsf{F}\phi_1$ iff ???



If π is not a k-loop, we introduce bounded semantics for LTL.

Definition (Bounded semantics for LTL)

Let $k \geq 0$ and π a path that is not a k-loop. An LTL formula ϕ is valid along π with bound k, written $\pi \models_k^0 \phi$, iff:

- $\pi \models_k^i \mathsf{G}\phi_1$ is always false
- $\pi \models_k^i \mathsf{F}\phi_1$ iff $\exists i \leq j \leq k$ such that $\pi \models_k^j \phi_1$

Now we see how to reduce BMC to SAT.

- the first thing to do is to define a Boolean formula that encodes all the paths of $\mathcal{M}$ of length k.

Now we see how to reduce BMC to SAT.

- the first thing to do is to define a Boolean formula that encodes all the paths of $\mathcal{M}$ of length k.

Definition (Unfolding of the Transition Relation)

For a Kripke structure $\mathcal{M}$ and $k \geq 0$, we define:

$$[\![\mathcal{M}]\!]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$

N.B.: For each $i \geq 0$, with s_i we represent the i^{th}-stepped version of the set of variables $\bar{s}$. For example, $s_1 := \bar{s}'$.

Encoding of the LTL formula

So far, we have seen how to encode paths of length k of the model $\mathcal{M}$.

Encoding of the LTL formula

So far, we have seen how to encode paths of length k of the model $\mathcal{M}$.

- intuitively, this corresponds to the left-hand side of the automaton $\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\neg\psi}$
- now we see how to encode the right-hand side.

Encoding of the LTL formula

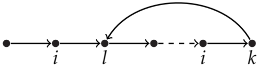
So far, we have seen how to encode paths of length k of the model $\mathcal{M}$.

- intuitively, this corresponds to the left-hand side of the automaton $\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\neg\psi}$
- now we see how to encode the right-hand side.

We have seen that BMC distinguishes between lasso-shaped (k-loop) and loop-free paths:

- we start with the encoding in case of k-loops.

Encoding of a loop

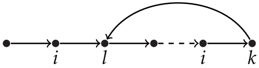


Definition (Loop Encoding)

Let $l \leq k$. We define:

$$ {}_{l}L_k := T(s_k, s_l) \qquad\qquad L_k := \bigvee_{l=0}^{k} {}_{l}L_k $$

Encoding of a loop



Definition (Loop Encoding)

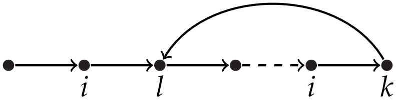
Let $l \leq k$. We define:

$$_{l}L_k := T(s_k, s_l) \qquad L_k := \bigvee_{l=0}^{k} {}_{l}L_k$$

Definition (Successor in a Loop)

Let $l, i \leq k$ and π be a (k, l)-loop. We define the successor $succ(i)$ of i in π as:

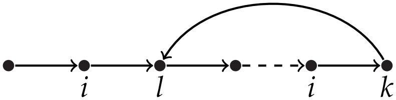
- $succ(i) := i + 1$ if $i < k$;
- $succ(i) := l$ if $i = k$.



Definition (Encoding of an LTL formula for a (k, l)-loop)

Let ϕ be an LTL formula and $l, i, k \geq 0$ such that $l, i \leq k$. We define ${}_l[\![\phi]\!]_k^i$ recursively as follows:

- ${}_l[\![p]\!]_k^i := p(s_i)$
- ${}_l[\![\neg p]\!]_k^i := \neg p(s_i)$

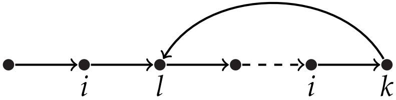


Definition (Encoding of an LTL formula for a (k, l)-loop)

Let ϕ be an LTL formula and $l, i, k \geq 0$ such that $l, i \leq k$. We define ${}_l[\![\phi]\!]_k^i$ recursively as follows:

- ${}_l[\![\phi_1 \vee \phi_2]\!]_k^i := {}_l[\![\phi_1]\!]_k^i \vee {}_l[\![\phi_2]\!]_k^i$
- ${}_l[\![\phi_1 \wedge \phi_2]\!]_k^i := {}_l[\![\phi_1]\!]_k^i \wedge {}_l[\![\phi_2]\!]_k^i$

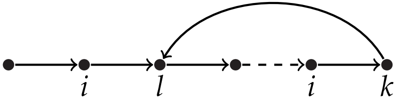
Encoding in case of Loop



Definition (Encoding of an LTL formula for a (k, l)-loop)

Let ϕ be an LTL formula and $l, i, k \geq 0$ such that $l, i \leq k$. We define ${}_l[\![\phi]\!]_k^i$ recursively as follows:

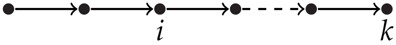
- ${}_l[\![\mathsf{X}\phi_1]\!]_k^i := {}_l[\![\phi_1]\!]_k^{succ(i)}$
- ${}_l[\![\phi_1 \mathsf{U} \phi_2]\!]_k^i := {}_l[\![\phi_2]\!]_k^i \vee ({}_l[\![\phi_1]\!]_k^i \wedge {}_l[\![\phi_1 \mathsf{U} \phi_2]\!]_k^{succ(i)})$



Definition (Encoding of an LTL formula for a (k, l)-loop)

Let ϕ be an LTL formula and $l, i, k \geq 0$ such that $l, i \leq k$. We define ${}_l[\![\phi]\!]_k^i$ recursively as follows:

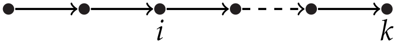
- ${}_l[\![\mathsf{G}\phi_1]\!]_k^i := {}_l[\![\phi_1]\!]_k^i \wedge {}_l[\![\mathsf{G}\phi_1]\!]_k^{succ(i)}$
- ${}_l[\![\mathsf{F}\phi_1]\!]_k^i := {}_l[\![\phi_1]\!]_k^i \vee {}_l[\![\mathsf{F}\phi_1]\!]_k^{succ(i)}$



Definition (Encoding of an LTL formula for a loop-free path)

Let ϕ be an LTL formula and $i, k \geq 0$. We define $[\![\phi]\!]_k^i$ recursively as follows:

- $[\![\phi]\!]_k^{k+1} := \bot$

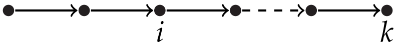


Definition (Encoding of an LTL formula for a loop-free path)

Let ϕ be an LTL formula and $i, k \geq 0$. We define $[\![\phi]\!]_k^i$ recursively as follows:

- $[\![p]\!]_k^i \coloneqq p(s_i)$
- $[\![\neg p]\!]_k^i \coloneqq \neg p(s_i)$

with $i \leq k$

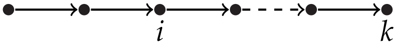


Definition (Encoding of an LTL formula for a loop-free path)

Let ϕ be an LTL formula and $i, k \geq 0$. We define $[\![\phi]\!]_k^i$ recursively as follows:

- $[\![\phi_1 \vee \phi_2]\!]_k^i := [\![\phi_1]\!]_k^i \vee {}_l[\![\phi_2]\!]_k^i$
- $[\![\phi_1 \wedge \phi_2]\!]_k^i := [\![\phi_1]\!]_k^i \wedge {}_l[\![\phi_2]\!]_k^i$

with $i \leq k$



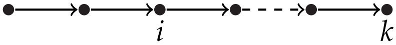
Definition (Encoding of an LTL formula for a loop-free path)

Let ϕ be an LTL formula and $i, k \geq 0$. We define $[\![\phi]\!]_k^i$ recursively as follows:

- $[\![\mathsf{X}\phi_1]\!]_k^i := [\![\phi_1]\!]_k^{i+1}$
- ${}_l[\![\phi_1 \mathsf{U} \phi_2]\!]_k^i := {}_l[\![\phi_2]\!]_k^i \vee ({}_l[\![\phi_1]\!]_k^i \wedge {}_l[\![\phi_1 \mathsf{U} \phi_2]\!]_k^{i+1})$

with $i \leq k$

Encoding in case of NO Loops



Definition (Encoding of an LTL formula for a loop-free path)

Let ϕ be an LTL formula and $i, k \geq 0$. We define $[\![\phi]\!]_k^i$ recursively as follows:

- ${}_l[\![\mathsf{G}\phi_1]\!]_k^i := {}_l[\![\phi_1]\!]_k^i \wedge {}_l[\![\mathsf{G}\phi_1]\!]_k^{i+1}$
- ${}_l[\![\mathsf{F}\phi_1]\!]_k^i := {}_l[\![\phi_1]\!]_k^i \vee {}_l[\![\mathsf{F}\phi_1]\!]_k^{i+1}$

with $i \leq k$

Overall encoding

Definition (Overall encoding)

Let ϕ be an LTL formula, $\mathcal{M}$ be a Kripke structure and $k \geq 0$:

$$[\![M, \phi]\!]_k \coloneqq \underbrace{[\![\mathcal{M}]\!]_k}_{\substack{\text{encoding of}\\ \text{the machine}}} \wedge \left((\underbrace{\neg L_k \wedge [\![\phi]\!]_k^0}_{\substack{\text{loop-free}\\ \text{models}}}) \vee \underbrace{\bigvee_{l=0}^{k} ({}_lL_k \wedge {}_l[\![\phi]\!]_k^0)}_{\substack{\text{lasso-shaped}\\ \text{models}}} \right)$$

Overall encoding

Definition (Overall encoding)

Let ϕ be an LTL formula, $\mathcal{M}$ be a Kripke structure and $k \geq 0$:

$$[\![M, \phi]\!]_k := \underbrace{[\![\mathcal{M}]\!]_k}_{\substack{\text{encoding of}\\ \text{the machine}}} \wedge \left(\underbrace{(\neg L_k \wedge [\![\phi]\!]_k^0)}_{\substack{\text{loop-free}\\ \text{models}}} \vee \underbrace{\bigvee_{l=0}^{k} ({}_l L_k \wedge {}_l[\![\phi]\!]_k^0)}_{\substack{\text{lasso-shaped}\\ \text{models}}} \right)$$

Theorem (Soundness)

$[\![\mathcal{M}, \phi]\!]_k$ is satisfiable iff $\mathcal{M} \models_k E\phi$.

Completeness

Algorithm:

- start with $k = 0$
- call a SAT-solver on $[\![\mathcal{M}, \phi]\!]_k$
- if it is SAT, stop; otherwise, $k{+}{+}$.

Completeness

Algorithm:

- start with $k = 0$
- call a SAT-solver on $[\![\mathcal{M}, \phi]\!]_k$
- if it is SAT, stop; otherwise, $k{+}{+}$.

What happens if $\mathcal{M} \not\models \phi$?

Completeness

Algorithm:

- start with $k = 0$
- call a SAT-solver on $[\![\mathcal{M}, \phi]\!]_k$
- if it is SAT, stop; otherwise, $k{+}{+}$.

What happens if $\mathcal{M} \not\models \phi$?

- the procedure does not terminate

Completeness

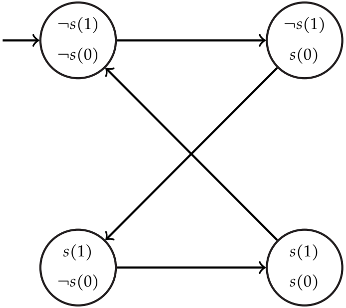
Algorithm:

- start with $k = 0$
- call a SAT-solver on $[\![\mathcal{M}, \phi]\!]_k$
- if it is SAT, stop; otherwise, k++.

What happens if $\mathcal{M} \not\models \phi$?

- the procedure does not terminate
- in order to be complete, BMC needs to compute the recurrence diameter: very costly
- BMC is mainly used as a bug finder, rather than as a prover.

Example



- $\phi_1 \coloneqq \mathsf{GF}(s(0) \wedge s(1))$ ✓
- $\phi_2 \coloneqq \mathsf{FG}(\neg s(0) \wedge \neg s(1))$ ✗

LTL-SAT is the problem of establishing if, given an LTL formula ϕ, there exists an infinite state sequence σ such that $\sigma \models \phi$.

Solving LTL-SAT with BMC

LTL-SAT is the problem of establishing if, given an LTL formula ϕ, there exists an infinite state sequence σ such that $\sigma \models \phi$.

- how can one solve LTL-SAT with BMC?

Solving LTL-SAT with BMC

LTL-SAT is the problem of establishing if, given an LTL formula ϕ, there exists an infinite state sequence σ such that $\sigma \models \phi$.

- how can one solve LTL-SAT with BMC?
- model checking:

$$[\![\mathcal{M}]\!]_k \wedge \left((\neg L_k \wedge [\![\phi]\!]_k^0) \vee \bigvee_{l=0}^{k} ({}_lL_k \wedge {}_l[\![\phi]\!]_k^0) \right)$$

Solving LTL-SAT with BMC

LTL-SAT is the problem of establishing if, given an LTL formula ϕ, there exists an infinite state sequence σ such that $\sigma \models \phi$.

- how can one solve LTL-SAT with BMC?
- model checking:

$$[\![\mathcal{M}]\!]_k \wedge \left((\neg L_k \wedge [\![\phi]\!]_k^0) \vee \bigvee_{l=0}^{k} ({}_l L_k \wedge {}_l[\![\phi]\!]_k^0) \right)$$

- satisfiability checking

$$\top \wedge \left((\neg L_k \wedge [\![\phi]\!]_k^0) \vee \bigvee_{l=0}^{k} ({}_l L_k \wedge {}_l[\![\phi]\!]_k^0) \right)$$

BLACK

- we developed this tool based on the idea of *bounded satisfiability checking*
- BLACK = Bounded Ltl sAtisfiability ChecKer
- `https://www.black-sat.org/en/stable/`
- Examples

REFERENCES

Bibliography I

Armin Biere et al. (1999). "Symbolic model checking without BDDs". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, pp. 193–207.