Bounded Model Checking

Luca Geatti

Free University of Bozen-Bolzano

Automatic System Verification

UniUD, May 23^{th} 2022, Udine, Italy

Introduction

Introduction

- Automatic formal verification techniques: great progress in the last decades;

Introduction

- Automatic formal verification techniques: great progress in the last decades;
- big chip or software companies have integrated them in their development or quality assurance process;

Introduction

- Automatic formal verification techniques: great progress in the last decades;
- big chip or software companies have integrated them in their development or quality assurance process;
- Intel: FDIV bug, error in the floating point division instruction on some Intel®Pentium® processors.

Introduction

- Automatic formal verification techniques: great progress in the last decades;
- big chip or software companies have integrated them in their development or quality assurance process;
- Intel: FDIV bug, error in the floating point division instruction on some Intel®Pentium® processors.
  - it costed $\approx$ US $475 million;

Introduction

- Automatic formal verification techniques: great progress in the last decades;
- big chip or software companies have integrated them in their development or quality assurance process;
- Intel: FDIV bug, error in the floating point division instruction on some Intel®Pentium® processors.
  - it costed $\approx$ US $475 million;
  - big investment in formal verification.

Model Checking

The most used formal verification technique is Model Checking (MC, for short).

Model Checking

The most used formal verification technique is Model Checking (MC, for short).

- the system to verify is modeled as a finite-state machine (*i.e.*, Kripke structure) and the specification is expressed by means of a temporal logic formula;

Model Checking

The most used formal verification technique is Model Checking (MC, for short).

- the system to verify is modeled as a finite-state machine (*i.e.*, Kripke structure) and the specification is expressed by means of a temporal logic formula;
- distinctive features:
  - fully automatic;

Model Checking

The most used formal verification technique is Model Checking (MC, for short).

- the system to verify is modeled as a finite-state machine (*i.e.*, Kripke structure) and the specification is expressed by means of a temporal logic formula;
- distinctive features:
  - fully automatic;
  - exhaustive;

Model Checking

The most used formal verification technique is Model Checking (MC, for short).

- the system to verify is modeled as a finite-state machine (*i.e.*, Kripke structure) and the specification is expressed by means of a temporal logic formula;
- distinctive features:
  - fully automatic;
  - exhaustive;
  - it generates a counterexample trace if the specification does not hold.

Linear Temporal Logic

We consider LTL model checking.

- LTL syntax:

$$\begin{aligned} p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \\ \mid \mathsf{X}\,\phi_1 \mid \phi_1\,\mathcal{U}\,\phi_2 \\ \mid \phi_1\,\mathcal{R}\,\phi_2 \mid \mathsf{F}\,\phi_1 \mid \mathsf{G}\,\phi_1 \end{aligned}$$

Linear Temporal Logic

We consider LTL model checking.

- LTL syntax:

$$
\begin{aligned}
p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \\
\mid \mathsf{X}\,\phi_1 \mid \phi_1\,\mathcal{U}\,\phi_2 \\
\mid \phi_1\,\mathcal{R}\,\phi_2 \mid \mathsf{F}\,\phi_1 \mid \mathsf{G}\,\phi_1
\end{aligned}
$$

- shortcuts:
  - $\phi_1\,\mathcal{R}\,\phi_2 \equiv \neg(\neg\phi_1\,\mathcal{U}\,\neg\phi_2)$,
  - $\mathsf{F}\,\phi_1 \equiv \top\,\mathcal{U}\,\phi_1$
  - $\mathsf{G}\,\phi_1 \equiv \neg\,\mathsf{F}\,\neg\phi_1$

Linear Temporal Logic

- Semantics. LTL formulas are interpreted over infinite *state sequences* $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle \in (2^\Sigma)^\omega$ of sets of propositions $\sigma_i \in 2^\Sigma$:

$$
\begin{array}{lll}
\sigma \models_i p & \text{iff} & p \in \sigma_i \\
\sigma \models_i \mathsf{X}\,\phi & \text{iff} & \sigma \models_{i+1} \phi \\
\sigma \models_i \phi_1\,\mathcal{U}\,\phi_2 & \text{iff} & \text{there exists } j \geq i \text{ such that} \\
 & & \sigma \models_j \phi_2 \text{ and } \sigma \models_k \phi_1 \text{ for all} \\
 & & i \leq k < j \\
\ldots & & \\
\sigma \models_i \mathsf{F}\,\phi & \text{iff} & \exists j \geq i\ .\ \sigma \models_j \phi \\
\sigma \models_i \mathsf{G}\,\phi & \text{iff} & \forall j \geq i\ .\ \sigma \models_j \phi
\end{array}
$$

LTL model checking

- LTL model checking:
  - decide if $\mathcal{M}, s \models \phi$, where $\mathcal{M} = (S, I, T, L)$ is a Kripke structure, $s \in I$ is an initial state and ϕ is an LTL formula; in many contexts, you may find the notation: $\mathcal{M}, s \models A\phi$;
  - PSPACE-complete.

Classical Approach to LTL MC

In order to decide if $\mathcal{M}, s \models \phi$:

Classical Approach to LTL MC

In order to decide if $\mathcal{M}, s \models \phi$:

- build the Büchi automaton $\mathcal{A}_{\mathcal{M}}$ that accepts all and only the words corresponding to computations of $\mathcal{M}$;

Classical Approach to LTL MC

In order to decide if $\mathcal{M}, s \models \phi$:

- build the Büchi automaton $\mathcal{A}_{\mathcal{M}}$ that accepts all and only the words corresponding to computations of $\mathcal{M}$;
- build the Büchi automaton $\mathcal{A}_{\neg\phi}$ that accepts all and only the words corresponding to models of $\neg\phi$;

Classical Approach to LTL MC

In order to decide if $\mathcal{M}, s \models \phi$:

- build the Büchi automaton $\mathcal{A}_{\mathcal{M}}$ that accepts all and only the words corresponding to computations of $\mathcal{M}$;
- build the Büchi automaton $\mathcal{A}_{\neg\phi}$ that accepts all and only the words corresponding to models of $\neg\phi$;
- check the (non-)emptiness of the product automaton $\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\neg\phi}$.

  MC=universal problem, EMPTINESS= existential problem

Classical Approach to LTL MC

In order to decide if $\mathcal{M}, s \models \phi$:

- build the Büchi automaton $\mathcal{A}_{\mathcal{M}}$ that accepts all and only the words corresponding to computations of $\mathcal{M}$;
- build the Büchi automaton $\mathcal{A}_{\neg\phi}$ that accepts all and only the words corresponding to models of $\neg\phi$;
- check the (non-)emptiness of the product automaton $\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\neg\phi}$.

  MC=universal problem, EMPTINESS= existential problem
- if $L(\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\neg\phi}) \neq \emptyset$, then $\mathcal{M}, s \overset{?}{\models} \phi$.

Classical Approach to LTL MC

In order to decide if $\mathcal{M}, s \models \phi$:

- build the Büchi automaton $\mathcal{A}_{\mathcal{M}}$ that accepts all and only the words corresponding to computations of $\mathcal{M}$;
- build the Büchi automaton $\mathcal{A}_{\neg\phi}$ that accepts all and only the words corresponding to models of $\neg\phi$;
- check the (non-)emptiness of the product automaton $\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\neg\phi}$.

  MC=universal problem, EMPTINESS= existential problem
- if $L(\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\neg\phi}) \neq \emptyset$, then $\mathcal{M}, s \overset{?}{\models} \phi$.

$$\mathcal{M}, s \not\models \phi$$

State-space Explosion Problem

- the previous algorithm belongs to the class of explicit model checking algorithms:
  - the Kripke Structure $\mathcal{M}$ is represented as a set of memory locations, pointers ecc...

State-space Explosion Problem

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

$$\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2 \times \cdots \times \mathcal{M}_n$$

is exponential in n;

State-space Explosion Problem

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

$$\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2 \times \cdots \times \mathcal{M}_n$$

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

Tackling the explosion...

Three main techniques have been proposed:

- BDD-based symbolic model checking
- partial order reduction
- SAT-based symbolic model checking, aka Bounded Model Checking.

Tackling the explosion...

Three main techniques have been proposed:

- BDD-based symbolic model checking
- partial order reduction
- SAT-based symbolic model checking, aka Bounded Model Checking.

They allowed for the verification of systems with $> 10^{20}$ states.

Symbolic Transition Systems

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?

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

The SAT problem

- 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;

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 $\bar{s} := \{s(0), \ldots, s(n)\}$ 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 $\bar{s} := \{s(0), \dots, s(n)\}$ 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 $(\bar{s}, f_I, f_T, \{f_{p_1}, \dots, f_{p_k}\})$.

Symbolic Transition Systems

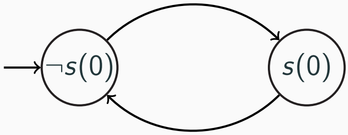
- we will write simply $\mathcal{M} = (S, I, T, L)$, meaning a symbolic transition system
- a path (or trace) $\pi = m_0, m_1, \dots$ is an infinite sequence of assignment to the state variables such that:
  - $m_0 \models I(s)$;
  - $m_i, m'_{i+1} \models T(s, s')$ holds, for all $i \geq 0$.

  where $\bar{s}' := \{s'(0), \dots, s'(n)\}$.

Example 1

simple-example.smv

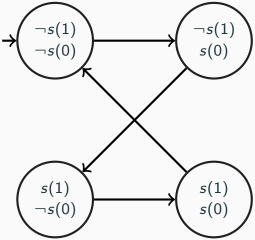
Example 1 - SMV




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

modulo-4-counter.smv

Example 2 - SMV




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

Bounded Model Checking

Bounded Model Checking

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

Bounded Model Checking

- 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

- 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

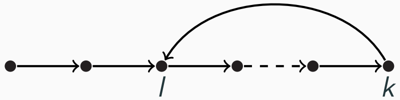
- BMC checks only bounded/finite traces of the system;
- ...but LTL formulas are defined over infinite state sequences;

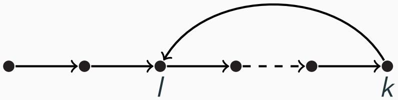
Loop-backs

- 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:

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

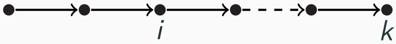
We call π a k-loop if there exists $l \leq k$ for which π is a (k, l)-loop.

BMC

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 <u>any extension of the path</u> is a model of ϕ under standard semantics (conservative semantics)

Bounded Semantics for LTL



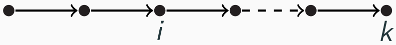
If π is not a k-loop, we introduce bounded semantics for LTL.

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:

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

Bounded Semantics for LTL



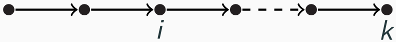
If π is not a k-loop, we introduce bounded semantics for LTL.

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



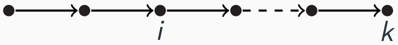
If π is not a k-loop, we introduce bounded semantics for LTL.

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* $\pi \models_k^{i+1} \phi_1$
- $\pi \models_k^i \phi_1 \,\mathcal{U}\, \phi_2$ *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



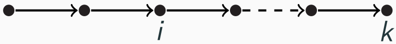
If π is not a k-loop, we introduce bounded semantics for LTL.

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* *???*

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:

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

SAT-based encoding of BMC

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.

SAT-based encoding of BMC

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:

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

SAT-based encoding of BMC

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:

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

What does a model of $[\![\mathcal{M}]\!]_k$ represent?

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

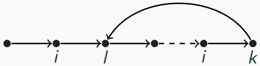
So far, we have seen how to encode paths of length k of the model $\mathcal{M}$.

- 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

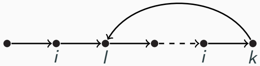


Definition (Loop Encoding)

Let $l \leq k$. We define:

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

Encoding of a loop



Definition (Loop Encoding)

Let $l \leq k$. We define:

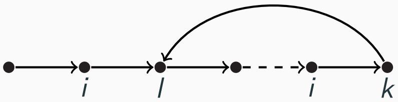
- ${}_lL_k := T(s_k, s_l)$
- $L_k := \bigvee_{l=0}^{k} {}_lL_k$

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 $i = k$.*

Encoding in case of Loop

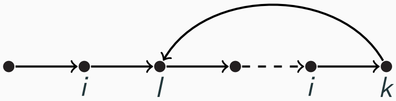


Definition (Encoding of an LTL formula for a (k, l)-loop)

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:

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

Encoding in case of Loop

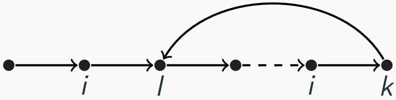


Definition (Encoding of an LTL formula for a (k, l)-loop)

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:

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

Encoding in case of Loop

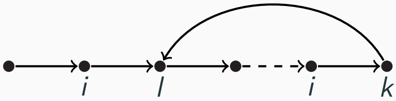


Definition (Encoding of an LTL formula for a (k, l)-loop)

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:

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

Encoding in case of Loop

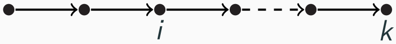


Definition (Encoding of an LTL formula for a (k, l)-loop)

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:

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

Encoding in case of NO Loops

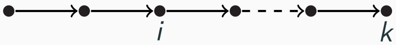


Definition (Encoding of an LTL formula for a loop-free path)

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

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

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:

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

with $i \leq k$

Encoding in case of NO Loops

$$\bullet \longrightarrow \bullet \longrightarrow \underset{i}{\bullet} \longrightarrow \bullet \dashrightarrow \bullet \longrightarrow \underset{k}{\bullet}$$

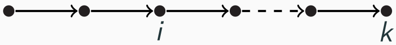
Definition (Encoding of an LTL formula for a loop-free path)

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 $i \leq k$

Encoding in case of NO Loops



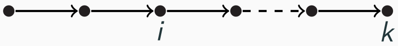
Definition (Encoding of an LTL formula for a loop-free path)

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

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

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 $k \geq 0$:

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

Overall encoding

Definition (Overall encoding)

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

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

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 $\mathcal{M} \not\models \phi$?

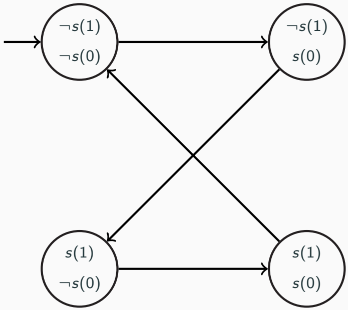
- the procedure does not terminate
- in order to be complete, BMC needs to compute the recurrence diameter: very costly
- BMC is mainly used as a bug finder, rather than as a prover.

Questions?

Appendix

Example

modulo-4-counter.smv



- $\phi_1 \coloneqq \mathsf{G}\,\mathsf{F}(s(0) \wedge s(1))$ ✓
- $\phi_2 \coloneqq \mathsf{F}\,\mathsf{G}(\neg s(0) \wedge \neg s(1))$ ✗

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 $\sigma \models \phi$.

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:

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

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 [1]

[1] https://github.com/black-sat/black