Symbolic Model Checking without BDDs

Armin Biere, Alessandro Cimatti, Edmund M. Clarke,
Yunshan Zhu (1999)

Presentation by Stavros Aronis
Reading Group: Seminal Papers in Verification

18 May 2012

UPPSALA
UNIVERSITET

- **Model checking**:
  - Specification is given in a temporal logic (LTL, CTL, ...)
  - System is modelled as a finite state machine
- **Symbolic model checking**:
  - Encodes the finite state machine with boolean formulas
  - Can handle more than 10^{20} states
  - Originally done with **Binary Decision Diagrams**:
    - Canonical form
    - Can become too large for large systems
    - Size and complexity is affected by the ordering of variables
- **SAT solvers** also operate on boolean expressions:
  - Do not require a different canonical form
  - Efficient with thousands of variables

Idea

- The basic idea of *bounded model checking* is to consider only a *finite prefix* of a path that is a counterexample of the property that we want to prove
- For LTL, this is a solution to an existential model checking problem for the negation of a formula
- If we search *all possible finite prefixes* without finding a solution, then such solution does not exist and the property holds

Overview

1. Paths, Bounded Prefixes and Loops
2. Equivalence between bounded and unbounded
3. Finding a path through SAT
   - Translation of the Finite State Machine
   - Translation of the LTL formula
4. Determining the bound
5. Evaluation of the method

- **Sequence**: $\pi = (s_0, s_1, ...)$, $\pi(i) = s_i$, $\pi^i = (s_i, s_{i+1}, ...)$
- **Path**: π, where $\pi(i) \to \pi(i+1)$ for all $i \in \mathbb{N}$
- Semantics of LTL for paths π:

$$
\begin{array}{lll}
\pi \models p & \text{iff} & p \in l(\pi(0)) \\
\pi \models \neg p & \text{iff} & p \notin l(\pi(0)) \\
\pi \models f \wedge g & \text{iff} & \pi \models f \text{ and } \pi \models g \\
\pi \models f \vee g & \text{iff} & \pi \models f \text{ or } \pi \models g \\
\pi \models \mathbf{G} f & \text{iff} & \forall i . \pi^i \models f \\
\pi \models \mathbf{F} f & \text{iff} & \exists i . \pi^i \models f \\
\pi \models \mathbf{X} f & \text{iff} & \pi^1 \models f \\
\pi \models f \mathbf{U} g & \text{iff} & \exists i [\pi^i \models g \text{ and } \forall j, j < i . \pi^j \models f] \\
\pi \models f \mathbf{R} g & \text{iff} & \forall i [\pi^i \models g \text{ or } \exists j, j < i . \pi^j \models f]
\end{array}
$$

Bounded Prefixes and Loops

- We check only **bounded prefixes** of a path
- The prefix might be finite, but it can represent an infinite path if it has a *back loop* from the last state of the prefix to a previous state
- These back loops are essential if the path should be a witness of an infinite behaviour (e.g. in $\mathbf{G}p$)

- We check only **bounded prefixes** of a path
- The prefix might be finite, but it can represent an infinite path if it has a *back loop* from the last state of the prefix to a previous state
- These back loops are essential if the path should be a witness of an infinite behaviour (e.g. in $\mathbf{G}p$)
- For $l \leq k$ we call a path π a **(k,l)-loop** if:
  - $\pi(k) \rightarrow \pi(l)$ and
  - $\pi = u \cdot v^{\omega}$ with $u = (\pi(0), \ldots, \pi(l-1))$ and $v = (\pi(l), \ldots, \pi(k))$.
- We call π simply a **k-loop** if there is an $l \in \mathbb{N}$ with $l \leq k$ for which π is a (k, l)-loop

- We can define $\pi \models_k f$ with the expected semantics
- $\mathbf{G}p$ can only hold for a path with a loop

Equivalence between bounded and unbounded

- If h is an LTL formula and π a path, then $\pi \models_k h \Rightarrow \pi \models h$
- Let f be an LTL formula and M a Kripke structure. If $M \models \mathbf{E}f$ then there exists $k \in \mathbb{N}$ with $M \models_k \mathbf{E}f$

Equivalence between bounded and unbounded

- If h is an LTL formula and π a path, then $\pi \models_k h \Rightarrow \pi \models h$
- Let f be an LTL formula and M a Kripke structure. If $M \models \mathbf{E}f$ then there exists $k \in \mathbb{N}$ with $M \models_k \mathbf{E}f$

  Proof sketch:

  1. *Initially*: Existential LTL model checking problem
  2. *Transform*: Fair CTL model checking problem for $\mathbf{EG}\texttt{true}$ in a product Kripke structure

Equivalence between bounded and unbounded

- If h is an LTL formula and π a path, then $\pi \models_k h \Rightarrow \pi \models h$
- Let f be an LTL formula and M a Kripke structure. If $M \models \mathbf{E}f$ then there exists $k \in \mathbb{N}$ with $M \models_k \mathbf{E}f$

  Proof sketch:

  1. *Initially*: Existential LTL model checking problem
  2. *Transform*: Fair CTL model checking problem for $\mathbf{EG}\texttt{true}$ in a product Kripke structure
  3. *Assertion*: f being existentially valid in M **implies** a path in the product structure that
     - starts with an initial state
     - ends with a cycle in the SCC of fair states

Equivalence between bounded and unbounded

- If h is an LTL formula and π a path, then $\pi \models_k h \Rightarrow \pi \models h$
- Let f be an LTL formula and M a Kripke structure. If $M \models \mathbf{E}f$ then there exists $k \in \mathbb{N}$ with $M \models_k \mathbf{E}f$

  Proof sketch:

  1. *Initially*: Existential LTL model checking problem
  2. *Transform*: Fair CTL model checking problem for $\mathbf{EG}\texttt{true}$ in a product Kripke structure
  3. *Assertion*: f being existentially valid in M **implies** a path in the product structure that
     - starts with an initial state
     - ends with a cycle in the SCC of fair states
  4. *Reverse*: Transform cycle to a k-loop in the original M (which also satisfies $\pi \models f$)
  5. By definition, $\pi \models_k f$

Finding a path with SAT

- The solution (if it exists) will be a path
- We encode the states of the FSM with boolean vectors
- The solution of the bounded model checking problem will appear as a path encoded in the variables of the SAT problem
- The path will have to satisfy:
  - Initial states
  - Transition relation
  - Certain predicates for each state in the path

Translation of the Finite State Machine

If we pick a specific bound k then the Kripke structure can be translated to the following boolean formula:

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

- Not all LTL formulas can hold in a bounded non-looping path ($\mathbf{G}p$ can never hold)
- There exist different SAT translations for the LTL operators depending on whether the path has a loop or not

- Not all LTL formulas can hold in a bounded non-looping path ($\mathbf{G}p$ can never hold)
- There exist different SAT translations for the LTL operators depending on whether the path has a loop or not
- The existence of a loop can be decided with another boolean formula:

$$_lL_k = T(s_k, s_l)$$

$$L_k := \bigvee_{l=0}^{k} {_lL_k}$$

Translation of the LTL formula

- The simplest case: $\pi \vDash p$ *iff* $p \in l(\pi(0))$
- This means that the respective state must satisfy the predicate
- We can construct a SAT term for this using the boolean encoding

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

Translation of the LTL formula

- The simplest case: $\pi \vDash p$ *iff* $p \in l(\pi(0))$
- This means that the respective state must satisfy the predicate
- We can construct a SAT term for this using the boolean encoding

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

- For the other LTL operators more complex terms need to be constructed (taking bounds and loops into account)

  Example:

$$[\![\mathbf{F}p]\!]_k^i := \bigvee_{j=i}^{k} [\![p]\!]_k^j$$

$$_l[\![\mathbf{F}p]\!]_k^i := \bigvee_{j=min(i,l)}^{k} {}_l[\![p]\!]_k^j$$

For a Kripke structure M and an LTL formula f:

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

Determining the bound

- Searching for all possible bounds is of course intractable
- There exist several theorems for the higher necessary bound k depending on the particular temporal logic

- Searching for all possible bounds is of course intractable
- There exist several theorems for the higher necessary bound k depending on the particular temporal logic
- **ECTL**:
  - $|M|$, the number of states in M

- Searching for all possible bounds is of course intractable
- There exist several theorems for the higher necessary bound k depending on the particular temporal logic
- **ECTL**:
  - $|M|$, the number of states in M
  - *Better:* Diameter of the Kripke structure (if a state is reachable from another, then it can be reached with a path of this length at most)

- Searching for all possible bounds is of course intractable
- There exist several theorems for the higher necessary bound k depending on the particular temporal logic
- **ECTL**:
  - $|M|$, the number of states in M
  - *Better:* Diameter of the Kripke structure (if a state is reachable from another, then it can be reached with a path of this length at most)
- **LTL**:
  - $|M| \cdot 2^{|f|}$

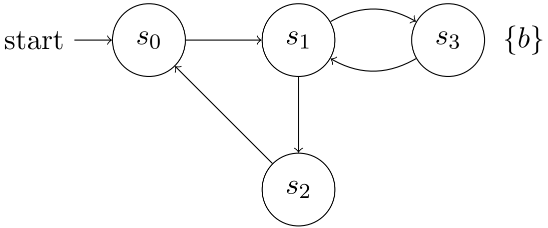
- Searching for all possible bounds is of course intractable
- There exist several theorems for the higher necessary bound k depending on the particular temporal logic
- **ECTL**:
  - $|M|$, the number of states in M
  - *Better:* Diameter of the Kripke structure (if a state is reachable from another, then it can be reached with a path of this length at most)
- **LTL**:
  - $|M| \cdot 2^{|f|}$
  - LTL model checking is PSPACE-complete.
  - Polynomial-time reduction to SAT $\rightarrow$ LTL $\in$ NP.
  - Therefore, a polynomial bound on k with respect to the size of M and f is unlikely to be found unless PSPACE $=$ NP.

Results from *Bounded Model Checking* (2003) by Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, Yunshan Zhu

- Several groups report that SAT based Bounded Model Checking is typically faster in finding bugs compared to BDDs
- The deeper the bug is (i.e. the longer the shortest path leading to it is), the less advantage BMC has.
- With state of the art SAT solvers and typical hardware designs (as of 2003), it usually cannot reach bugs beyond 80 cycles in a reasonable amount of time, although there are exceptions
- In any case, BMC can solve many of the problems that cannot be solved by BDD based model checkers.

- It is possible to tune SAT solvers by exploiting the structure of the problem being encoded in order to increase efficiency.
- Notable contributions are:
  - use of problem-dependent variable ordering and splitting heuristics in the SAT solver
  - pruning the search space by exploiting the regular structure of BMC formulas
  - reusing learned information between the various SAT instances
- Incremental SAT solver:
  - Rather than generating a new SAT instance for each attempted bound clauses are added and removed from a single SAT instance
  - retain the learned information from the previous instances

In the following Kripke structure you are asked to check if the LTL property $\mathbf{G}\neg b$ holds using bounded model checking:



Thank you!