Proving properties for Bounded Model Checking

Mohammad Awedh

Department of Electrical and Computer Engineering
University of Colorado at Boulder
Outline

Introduction
- Formal verification
- Model checking
  - Explicit vs. Symbolic
  - BDD-based vs. SAT-based
  - Bounded vs. Unbounded
- LTL model checking
- Bounded Model Checking (BMC)

Motivation
- Proving properties vs. finding bugs
  - Reachability
  - Language emptiness
  - Liveness
Outline

- Prove termination
  - Termination criteria for language emptiness
  - An improved criterion for language emptiness
  - Automaton reachability analysis
  - Tight automaton and automaton strength
  - Checking Multiple fairness conditions
  - Algorithm for general LTL properties
  - Termination criteria for simple liveness
  - Experimental results
Outline

- Proving invariants
  - Inductive proof
    - Fixpoint-based technique
    - BMC technique
  - Invariant strengthening
    - Auxiliary invariant
- **Automatic Invariant Strengthening**
  - Invariant Strengthening Algorithm
  - Prune the search space
  - Proving the existence of a counterexample
  - Soundness and completeness
- Experimental results
Outline

- Increase the Robustness of BMC
  - Integrating BDD-based and SAT-based methods
  - New model checking algorithm
  - Example of reduced termination depth
  - Counterexample generation
  - Soundness and completeness
  - Experimental results

- Conclusion

- Future work
Formal Verification

- The formal proof of correctness of a design with respect to a certain specification.
Formal Verification

- The formal proof of correctness of a design with respect to a certain specification.
- It is based on some mathematical theories, such as logic, automata or graph theory.
Formal Verification

- The formal proof of correctness of a design with respect to a certain specification
- It is based on some mathematical theories, such as logic, automata or graph theory
- It helps to
  - ensure functional correctness
  - improve design quality and turnaround time
  - reduce the overall production costs
Formal Verification

- The formal proof of correctness of a **design** with respect to a certain **specification**

- It is based on some mathematical theories, such as logic, automata or graph theory

- It helps to
  - ensure functional correctness
  - improve design quality and turnaround time
  - reduce the overall production costs

- Formal verification has a variety of methods:
  - Theorem Proving
  - Equivalence Checking
  - Model Checking
Formal Verification Tree

Formal Verification

- Model Checking
- Theorem Proving
- Equivalence Checking
  - Explicit Symbolic
    - BMC
    - UMC
  - SAT-based
  - BDD-based
  - BDD/SAT-based

- Falsification Verification
Formal Verification Tree

Formal Verification

- Theorem Proving
- Model Checking
  - Explicit
  - Symbolic
    - BDD-based
    - SAT-based
    - BDD/SAT-based

- Equivalence Checking
Formal Verification Tree

- Formal Verification
  - Theorem Proving
  - Model Checking
    - Explicit
      - BDD-based
    - Symbolic
      - SAT-based
      - BDD/SAT-based
  - Equivalence Checking
    - BMC
    - UMC
Why Model Checking?

Have we designed what we intended to design?
Why Model Checking?

- If the answer is yes I guess
Why Model Checking?

- If the answer is yes I guess
  - Financial consequences
    - Pentium Floating-Point Bugs
      - Error in floating-point division. Cost: $500 million
    - ARIANE failure
      - Software failure. Cost: $400 million
  - Loss of human lives
    - The explosion of Ariane 5 rocket

We need to change yes to YES!

Ensure system behaves as intended

Scale with the complexity of VLSI designs

Reduce errors

Lead to higher design quality
Why Model Checking?

- If the answer is yes I guess
  - Financial consequences
    - Pentium Floating-Point Bugs
      - Error in floating-point division. Cost: $500 million
    - ARIANE failure
      - Software failure. Cost: $400 million
  - Loss of human lives
    - The explosion of Ariane 5 rocket

- We need to change yes to YES!
Why Model Checking?

- If the answer is *yes* I guess
  - Financial consequences
    - Pentium Floating-Point Bugs
      - Error in floating-point division. Cost: $500 million
    - ARIANE failure
      - Software failure. Cost: $400 million
  - Loss of human lives
    - The explosion of Ariane 5 rocket
- We need to change *yes* to **YES**!
  - Ensure system behaves as intended
  - Scale with the complexity of VLSI designs
  - Reduce errors
  - Lead to higher design quality
Potential Benefits of MC

- Finds errors in early design stages
- Provides counterexamples that can direct the designer to the problems
- Achieve high quality standards
- Shorten time-to-market
- Reduce manual validation phases

Compare to classical testing and validation techniques, requires users of different skills, discovers few hard to find errors.
Potential Benefits of MC

- Finds errors in early design stages
- Provides counterexamples that can direct the designer to the problems
- Achieve high quality standards
- Shorten time-to-market
- Reduce manual validation phases

Compare to classical testing and validation techniques
  - Requires users of different skills
  - Discovers few hard to find errors
The Need for Model Checking

- Growing complexity of environment and required functionality (bugs are more likely)
- Rising level of integration in digital design
- Safety-critical systems: life support systems
- Money-critical systems
- Market issues: time-to-market, development cost
- System reliability increasingly depends on the correct functionality of its hardware and software
- High cost of correcting errors
Model Checking in a Nutshell

- An **algorithmic** technique to verify that the behavior of a sequential system is complying with its specifications.

- It **automatically** verifies the temporal specifications of the sequential system by **exploring** its reachable states.

- If the sequential system fails to satisfy a specification, it generates a **counterexample** that witnesses the failing of the specification.

- Often expresses requirements using temporal logics or automata.
Temporal Logic

- Is a logic augmented with temporal operators that implicitly express time
- Allows one to express the ordering of events along an execution
- May differ according to queries about the temporal ordering between events
  - A **linear-time** temporal logic (e.g., LTL) describes events over a single execution path
  - A **branching-time** temporal logic (e.g., CTL) describes events along execution paths that are possible from a given event
Explicit Model Checking

Symbolic Model Checking

- Uses Binary Decision Diagrams (BDD-based)
- Increases the capacity of model checking technique to handle in excess of $10^{20}$ states (Burch et al. [1992])
- Is also limited by the state explosion problem with less extent (Cannot handle a lot of variables)

SAT-based Symbolic Model Checking

- Uses Boolean Satisfiability
- Can handle many variables
Model Checking techniques

- Explicit Model Checking
  - Explicitly represents the reachable states of the sequential system
  - The number of states of a sequential system grows exponentially with the number of state variables
  - State explosion problem
  - Does not scale to large systems
  - Cannot handle a lot of states
Model Checking techniques

- Explicit Model Checking
- Symbolic model checking

- Symbolic Model Checking
  - Uses Binary Decision Diagrams
  - Increases the capacity of model checking technique to handle in excess of $10^{20}$ states (Burch et al. [1992])
  - Is also limited by the state explosion problem with less extent (Cannot handle a lot of variables)

- SAT-based Symbolic Model Checking
  - Uses Boolean Satisfiability
  - Can handle many variables
Model Checking techniques

- Explicit Model Checking
- Symbolic model checking
  - Uses Boolean functions to implicitly represent and manipulate sets of states
    - **BDD-based** Symbolic Model Checking
      - Uses Binary Decision Diagrams
      - Increases the capacity of model checking technique to handle in excess of $10^{20}$ states (Burch et al. [1992])
      - Is also limited by the state explosion problem with less extent (Cannot handle a lot of variables)
    - **SAT-based** Symbolic Model Checking
      - Uses Boolean Satisfiability
      - Can handle many variables
LTL Model Checking

- Uses linear-time temporal logic to express specifications

The standard technique constructs a Büchi automaton that accepts all the counterexamples to the LTL formula. Checks the composition of the property automaton and the original model for language emptiness. The language-emptiness check is performed by searching for a fair cycle.
LTL Model Checking

- Uses linear-time temporal logic to express specifications
- The standard technique
  - Constructs a Büchi automaton that accepts all the counterexamples to the LTL formula
  - Checks the composition of the property automaton and the original model for language emptiness
LTL Model Checking

- Uses linear-time temporal logic to express specifications
- The standard technique
  - Constructs a Büchi automaton that accepts all the counterexamples to the LTL formula
  - Checks the composition of the property automaton and the original model for language emptiness
- The language-emptiness check is performed by searching for a fair cycle
Bounded LTL Formulae

$F_p$

$G_p$

$G F_p$
Bounded Model Checking (BMC)

- Is a SAT-based LTL Model Checking technique
Bounded Model Checking (BMC)

- Is a SAT-based LTL Model Checking technique
- Checks for a counterexample of bounded length $k$ to an LTL property $\varphi$ in a model $\mathcal{M}$ ($\mathcal{M} \models_k \varphi$)
Bounded Model Checking (BMC)

- Is a SAT-based LTL Model Checking technique
- Checks for a counterexample of bounded length $k$ to an LTL property $\varphi$ in a model $\mathcal{M} (\mathcal{M} \models_k \varphi)$
- Encodes the counterexample search as propositional satisfiability (SAT)
Bounded Model Checking (BMC)

- Is a SAT-based LTL Model Checking technique
- Checks for a counterexample of bounded length $k$ to an LTL property $\varphi$ in a model $M$ ($M \models_k \varphi$)
- Encodes the counterexample search as propositional satisfiability (SAT)
- Constructs a propositional formula $P_k$ that is satisfiable iff there exits a length-$k$ counterexample $CE_k$
Bounded Model Checking (BMC)

- Is a SAT-based LTL Model Checking technique
- Checks for a counterexample of bounded length $k$ to an LTL property $\varphi$ in a model $\mathcal{M}$ ($\mathcal{M} \models_k \varphi$)
- Encodes the counterexample search as propositional satisfiability (SAT)
- Constructs a propositional formula $\mathcal{P}_k$ that is satisfiable iff there exits a length-$k$ counterexample $CE_k$
- If no counterexample is found, BMC increases $k$ until
Bounded Model Checking (BMC)

- Is a SAT-based LTL Model Checking technique
- Checks for a counterexample of bounded length $k$ to an LTL property $\varphi$ in a model $M$ ($M \models^k \varphi$)
- Encodes the counterexample search as propositional satisfiability (SAT)
- Constructs a propositional formula $P_k$ that is satisfiable iff there exits a length-$k$ counterexample $CE_k$
- If no counterexample is found, BMC increases $k$ until a counterexample is found,
Bounded Model Checking (BMC)

- Is a SAT-based LTL Model Checking technique
- Checks for a counterexample of bounded length $k$ to an LTL property $\varphi$ in a model $\mathcal{M}$ ($\mathcal{M} \models_k \varphi$)
- Encodes the counterexample search as propositional satisfiability (SAT)
- Constructs a propositional formula $P_k$ that is satisfiable iff there exits a length-$k$ counterexample $CE_k$
- If no counterexample is found, BMC increases $k$ until
  - a counterexample is found,
  - the search becomes intractable, or
Bounded Model Checking (BMC)

- Is a SAT-based LTL Model Checking technique
- Checks for a counterexample of bounded length $k$ to an LTL property $\varphi$ in a model $\mathcal{M}$ ($\mathcal{M} \models_k \varphi$)
- Encodes the counterexample search as propositional satisfiability (SAT)
- Constructs a propositional formula $\mathcal{P}_k$ that is satisfiable iff there exits a length-$k$ counterexample $\mathcal{CE}_k$
- If no counterexample is found, BMC increases $k$ until
  - a counterexample is found,
  - the search becomes intractable, or
  - $k$ reaches a certain bound and we conclude $\mathcal{M} \models \varphi$
Bounded Model Checking

\[ k=0 \]

\[ \text{BMC} \]

\[ \text{SAT} \]

Diagram with labeled nodes and edges.
Bounded Model Checking

\[ S_0 \land p_0 \]

0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8

BMC
\[ k=0 \]

SAT
Bounded Model Checking

\[ S_0 \land p_0 \]

BMC
\[ k=0 \]

SAT

awedh – p. 18
Bounded Model Checking

\[ S_0 \land p_0 \]

0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 8

BMC
\[ k=0 \]

UNSAT

SAT
Bounded Model Checking

\[ S_0 \land T(S_0, S_1) \land p_1 \]
Bounded Model Checking

\[ S_0 \land T(S_0, S_1) \land p_1 \]

\[
\begin{array}{c}
\text{BMC} \\
\kappa = 1
\end{array}
\]

\[
\begin{array}{c}
\text{SAT}
\end{array}
\]

\[
\begin{array}{c}
0 ightarrow 1 ightarrow 2 ightarrow 3 ightarrow 4 ightarrow 5 ightarrow 7 ightarrow 8
\end{array}
\]
Bounded Model Checking

\[ S_0 \land T(S_0, S_1) \land p_1 \]

BMC

\[ k=1 \]

UNSAT

SAT

awedh – p. 18
S₀ \land T(S₀, S₁) \land T(S₁, S₂) \land T(S₂, S₃) \land T(S₃, S₄) \land T(S₄, S₅) \land p₅
Bounded Model Checking

\[ S_0 \land T(S_0, S_1) \land T(S_1, S_2) \land T(S_2, S_3) \land T(S_3, S_4) \land T(S_4, S_5) \land p_5 \]
Bounded Model Checking

\[ S_0 \land T(S_0, S_1) \land T(S_1, S_2) \land T(S_2, S_3) \land T(S_3, S_4) \land T(S_4, S_5) \land p_5 \]
Bounded Model Checking

\[ S_0 \land T(S_0, S_1) \land T(S_1, S_2) \land T(S_2, S_3) \land T(S_3, S_4) \land T(S_4, S_5) \land T(S_5, S_6) \land p_6 \]
Bounded Model Checking

\[ S_0 \land T(S_0, S_1) \land T(S_1, S_2) \land T(S_2, S_3) \land T(S_3, S_4) \land T(S_4, S_5) \land T(S_5, S_6) \land p_6 \]
Bounded Model Checking

\[ S_0 \land T(S_0, S_1) \land T(S_1, S_2) \land T(S_2, S_3) \land T(S_3, S_4) \land T(S_4, S_5) \land T(S_5, S_6) \land p \]
Bounded Model Checking

\[ S_0 \land T(S_0, S_1) \land T(S_1, S_2) \land T(S_2, S_3) \land T(S_3, S_4) \land T(S_4, S_5) \land T(S_5, S_6) \land p \]
SAT vs. BDD

- BDD-based Model Checking
  - Often requires a good variable ordering
  - BDD may grow exponentially large as the number of reachable states increases: space-inefficiency

SAT-based Bounded Model Checking
- Generates the shortest counterexamples
- Its time requirements rapidly increase as the depth of the search increases: time-inefficiency

SAT-based and BDD-based techniques are complementary
It is hard to predict in advance the cases where the SAT-based BMC is more efficient than the BDD-based method

Strichman [2004]
SAT vs. BDD

- BDD-based Model Checking
  - Often requires a good variable ordering
  - BDD may grow exponentially large as the number of reachable states increases: **space-inefficiency**

- SAT-based Bounded Model Checking
  - Generates the shortest counterexamples
  - Its time requirements rapidly increase as the depth of the search increases: **time-inefficiency**
**SAT vs. BDD**

- **BDD-based Model Checking**
  - Often requires a good variable ordering
  - BDD may grow exponentially large as the number of reachable states increases: *space-inefficiency*

- **SAT-based Bounded Model Checking**
  - Generates the shortest counterexamples
  - Its time requirements rapidly increase as the depth of the search increases: *time-inefficiency*

- SAT-based and BDD-based techniques are complementary
SAT vs. BDD

- BDD-based Model Checking
  - Often requires a good variable ordering
  - BDD may grow exponentially large as the number of reachable states increases: space-inefficiency

- SAT-based Bounded Model Checking
  - Generates the shortest counterexamples
  - Its time requirements rapidly increase as the depth of the search increases: time-inefficiency

- SAT-based and BDD-based techniques are complementary

- It is hard to predict in advance the cases where the SAT-based BMC is more efficient than the BDD-based method [Strichman, 2004]
Bounded Model Checking

- Complete in theory
- Limited in practice to falsification: often finds a counterexample if it exists
- Proves an LTL property passes if a tight bound on the model is known
  - If no counterexample of length up to this bound is found, then no counterexamples of any length exist
  - Is difficult to obtain
- Limited in practice to prove reachability
BMC Encoding

- Is defined recursively on the structure of LTL formulae
BMC Encoding

- Is defined recursively on the structure of LTL formulae
- Has a significant impact on the performance of BMC
  - The complexity of the encoding and the generated propositional formula
  - The length of counterexamples and the termination depths
  - The performance of SAT solvers
Encoding Scheme for BMC

- Based on the bounded semantics of LTL formulae 
  (Biere et al. [1999])
- Based on the fixpoint characterizations of LTL formulae 
  (Frisch et al. [2002] and Latvala et al. [2004])
- Based on translation of liveness to safety property 
  (Biere et al. [2002])
- Based on translation of LTL formulae into Büchi automata 
  (Clarke et al. [2004])
- Based on Alternating automaton 
  (Sheridan [2004])
- Customized property encoding for LTL formulae 
  (Ganai et al. [2005])
Encoding Scheme for BMC

- Based on the bounded semantics of LTL formulae (Biere et al. [1999])

- Based on the fixpoint characterizations of LTL formulae (Frisch et al. [2002] and Latvala et al. [2004])
Encoding Scheme for BMC

- Based on the bounded semantics of LTL formulae (Biere et al. [1999])
- Based on the fixpoint characterizations of LTL formulae (Frisch et al. [2002] and Latvala et al. [2004])
- Based on translation of liveness to safety property (Biere et al. [2002])
Encoding Scheme for BMC

- Based on the bounded semantics of LTL formulae (Biere et al. [1999])
- Based on the fixpoint characterizations of LTL formulae (Frisch et al. [2002] and Latvala et al. [2004])
- Based on translation of liveness to safety property (Biere et al. [2002])
- Based on the translation of LTL formulae into Büchi automata (Clarke et al. [2004])
Encoding Scheme for BMC

- Based on the bounded semantics of LTL formulae (Biere et al. [1999])
- Based on the fixpoint characterizations of LTL formulae (Frisch et al. [2002] and Latvala et al. [2004])
- Based on translation of liveness to safety property (Biere et al. [2002])
- Based on the translation of LTL formulae into Büchi automata (Clarke et al. [2004])
- Based on Alternating automaton (Sheridan [2004])
Encoding Scheme for BMC

- Based on the bounded semantics of LTL formulae (Biere et al. [1999])
- Based on the fixpoint characterizations of LTL formulae (Frisch et al. [2002] and Latvala et al. [2004])
- Based on translation of liveness to safety property (Biere et al. [2002])
- Based on the translation of LTL formulae into Büchi automata (Clarke et al. [2004])
- Based on Alternating automaton (Sheridan [2004])
- Customized property encoding for LTL formulae (Ganai et al. [2005])
Falsification vs. Verification

Sometime, proving the absence of an error is more useful than finding one.

When checking an abstract model, then for a universal property,
Finding an error in the abstract model does not imply its existence in the original model.
However, proving that the property passes in the abstract model guarantee the absence of errors in the original model.

BMC efficiency reduces as the search for a counterexample increases.
It is more efficient to prove the property and stop early than keep searching for a counterexample.
Prove Termination with BMC
Proving states unreachable

Reachability: Prove whether there exists a path in a graph from a source state to a target state.
Proving states unreachable

- Reachability: Prove whether there exists a path in a graph from a source state to a target state
- A target state $t$ is not reachable from a source state $s$ if
Proving states unreachable

- Reachability: Prove whether there exists a path in a graph from a source state to a target state
- A target state \( t \) is not reachable from a source state \( s \) if
  - No path whose length is bounded by the longest shortest path starting at \( s \) reaches \( t \) (forward radius), or
Proving states unreachable

- Reachability: Prove whether there exists a path in a graph from a source state to a target state.
- A target state $t$ is not reachable from a source state $s$ if
  - No path whose length is bounded by the longest shortest path starting at $s$ reaches $t$ (forward radius), or
  - No path whose length is bounded by the longest shortest path leading to $t$ starts at $s$ (backward radius).
Proving states unreachable

- Reachability: Prove whether there exists a path in a graph from a source state to a target state

- A target state \( t \) is not reachable from a source state \( s \) if
  - No path whose length is bounded by the longest shortest path starting at \( s \) reaches \( t \) (forward radius), or
  - No path whose length is bounded by the longest shortest path leading to \( t \) starts at \( s \) (backward radius)

- Finding the forward and backward radii requires solving a sequence of Quantified Boolean Formulae (QBF) which are very expensive to solve (Biere et al. [1999])
Proving states unreachable

- An alternative method uses *simple paths*
An alternative method uses simple paths.

If $t$ is reachable from $s$, then it must be reachable via a path whose length is bounded by the longest simple path (forward/backward recurrence radius) (Sheeran et al. [2000])
An alternative method uses simple paths

If \( t \) is reachable from \( s \), then it must be reachable via a path whose length is bounded by the longest simple path (forward/backward recurrence radius) (Sheeran et al. [2000])

Forward/Backward recurrence radius \( \geq \) Forward/Backward radius
An alternative method uses simple paths

If \( t \) is reachable from \( s \), then it must be reachable via a path whose length is bounded by the longest simple path (forward/backward recurrence radius) (Sheeran et al. [2000])

Forward/Backward recurrence radius \( \geq \) Forward/Backward radius

Forward and backward recurrence radii can be found by solving a sequence of SAT instances

- Easier to compute
- Easily integrated in BMC
Proving states unreachable

(McMillan [2003]) described a termination criterion (based on interpolation) that is bound by the backward radius.
Proving states unreachable

(McMillan [2003]) described a termination criterion (based on interpolation) that is bound by the backward radius.

The method does not explicitly compute the backward radius, but it only examines counterexamples of length up to it.
Proving states unreachable

(McMillan [2003]) described a termination criterion (based on interpolation) that is bound by the backward radius.

The method does not explicitly compute the backward radius, but it only examines counterexamples of length up to it.

The use of the backward radius makes this termination criterion property-driven.
Proving language emptiness

Language emptiness: Prove that there exists no reachable fair cycle. Shortest paths can be used to bound the search for fair cycles (Clarke et al. [2004]). A path to a fair cycle is no longer than the longest shortest paths starting from an initial state. The path from $S_p$ to itself is no longer than the longest shortest paths. Simple paths can also be used: A shortest counterexample is no longer than the longest simple paths starting from an initial state.
Language emptiness: Prove that there exists no reachable fair cycle
Language emptiness: Prove that there exists no reachable fair cycle.

Shortest paths can be used to bound the search for fair cycles (Clarke et al. [2004]).
Proving language emptiness

Language emptiness: Prove that there exists no reachable fair cycle

Shortest paths can be used to bound the search for fair cycles ([Clarke et al. [2004]])

A path to a fair cycle is no longer than the longest shortest paths starting form an initial state.
Proving language emptiness

Language emptiness: Prove that there exists no reachable fair cycle

Shortest paths can be used to bound the search for fair cycles (Clarke et al. [2004])

- A path to a fair cycle is no longer than the longest shortest paths starting from an initial state
- The path from $S_p$ to itself is no longer than the longest shortest paths
Proving language emptiness

Language emptiness: Prove that there exists no reachable fair cycle

Shortest paths can be used to bound the search for fair cycles (Clarke et al. [2004])
- A path to a fair cycle is no longer than the longest shortest paths starting from an initial state
- The path from $S_p$ to itself is no longer than the longest shortest paths

Simple paths can also be used: A shortest counterexample is no longer than the longest simple paths starting from an initial state + 1
Shortest paths and termination

When proving language emptiness:

- The position of the fair states is not taken into account
  
- This holds also for the method based on longest simple paths
Shortest paths and termination

When proving language emptiness:

- The position of the fair states is not taken into account
- This holds also for the method based on longest simple paths
- It is hard to compute
Reduction of liveness to safety

(Biere et al. [2002]) described a method that reduces the checking for language emptiness to the checking of safety.
Reduction of liveness to safety

(Biere et al. [2002]) described a method that reduces the checking for language emptiness to the checking of safety.

Allows one to use the termination criteria for reachability when using BMC.
Reduction of liveness to safety

(Biere et al. [2002]) described a method that reduces the checking for language emptiness to the checking of safety.

- Allows one to use the termination criteria for reachability when using BMC.

The impact of the reduction method:

- Number of states \( = 2|S|(|S| + 1) = O(|S|^2) \)
- Number of reachable states \( \leq 2|R|(|R| + 1) = O(|R|^2) \)
- Diameter \( \leq 4d + 3 \)
- Radius \( \leq r + 3d + 3 \)
Our contribution

Termination criterion for language emptiness
Our contribution

- Termination criterion for language emptiness
- It is different from the other methods
  - Takes into account the position of fair states
  - Easier to check
  - Can be easily integrated in BMC

awedh – p. 31
Our contribution

- Termination criterion for language emptiness
  - It is different from the other methods
    - Takes into account the position of fair states
    - Easier to check
    - Can be easily integrated in BMC
- No blow-up of the model
Proving language emptiness

Proving $F G \neg p$
Proving language emptiness

- Proving $F G \neg p$
- A shortest counterexample to $F G \neg p$ consists of

Diagram:

- $S_0$
- $S_l$
- $S^f_p$
- $S^l_p$
- $S_k$
Proving language emptiness

- Proving $F \neg\neg G \not\subseteq p$
- A shortest counterexample to $F \neg\neg G \not\subseteq p$ consists of
  - A simple path from an initial state $S_0$ to a state that satisfies $p (S_p^l)$
Proving language emptiness

- Proving $F \, G \, \neg p$
- A shortest counterexample to $F \, G \, \neg p$ consists of
  - A simple path from an initial state $S_0$ to a state that satisfies $p \, (S^l_p)$
  - A simple path leading back to a state that satisfies $p \, (S^f_p)$ along which all states satisfy $\neg p$
Proving language emptiness

- Proving $F \text{ G } \neg p$

- A shortest counterexample to $F \text{ G } \neg p$ consists of
  - A simple path from an initial state $S_0$ to a state that satisfies $p (S^l_p)$
  - A simple path leading back to a state that satisfies $p (S^f_p)$ along which all states satisfy $\neg p$

- If no counterexample is found up to the sum of the bounds on the above simple paths, then $F \text{ G } \neg p$ holds
Proving $F G \neg \rho$

If we let $n$ captures the length of the longest initial simple path beyond which $\rho$ does not hold. Then, $n$ is the least value of $k$ for which $(\_ k) \rho k$ is unsatisfiable. $(\_ k) = simplePath k + 1$ $p (s k) p (s k + 1)$
Proving $F G \neg \rho$

If we let $n$ captures the length of the longest initial simple path beyond which $\rho$ does not hold.
Proving $F G \neg p$

If we let $n$ captures the length of the longest initial simple path beyond which $p$ does not hold.

Then, $n$ is the least value of $k$ for which $(\alpha \lor \beta)(k)$ is unsatisfiable.

- $\alpha(k) = I(s_0) \land \text{simplePath}_k \land p(s_k)$
- $\beta(k) = \text{simplePath}_{k+1} \land \neg p(s_k) \land p(s_{k+1})$
If we let $m$ captures the length of the longest simple path that loops back to a state satisfies $p$
Proving $F G \neg p$

- If we let $m$ captures the length of the longest simple path that loops back to a state satisfies $p$

- Then, $m$ is the least value of $k$ for which $\beta'(k)$ is unsatisfiable

  - $\beta'(k) = simplePath_{k+1} \land \bigwedge_{i=0}^{k} \neg p(s_i) \land p(s_{k+1})$
If we let $m$ captures the length of the longest simple path that loops back to a state satisfies $p$

Then, $m$ is the least value of $k$ for which $\beta'(k)$ is unsatisfiable

$\beta'(k) = \text{simplePath}_{k+1} \land \bigwedge_{i=0}^{k} \neg p(s_i) \land p(s_{k+1})$

If there is no counterexample of length $\leq n + m - 1$, then $\mathcal{M} \models F G \neg p$
Efficient implementation

\[ \alpha(k) = I(s_0) \land \text{simplePath}_k \land p(s_k) \]

\[ \beta(k) = \text{simplePath}_{k+1} \land \neg p(s_k) \land p(s_{k+1}) \]

\[ \beta'(k) = \text{simplePath}_{k+1} \land \bigwedge_{i=0}^{k} \neg p(s_i) \land p(s_{k+1}) \]
Efficient implementation

\[
\alpha(k) = I(s_0) \land simplePath_k \land p(s_k)
\]
\[
\beta(k) = simplePath_{k+1} \land \neg p(s_k) \land p(s_{k+1})
\]
\[
\beta'(k) = simplePath_{k+1} \land \land_{i=0}^{k} \neg p(s_i) \land p(s_{k+1})
\]

\(\beta(k)\) is unsatisfiable only if \(\beta'(k)\) is unsatisfiable
Efficient implementation

\[ \alpha(k) = I(s_0) \land \text{simplePath}_k \land p(s_k) \]
\[ \beta(k) = \text{simplePath}_{k+1} \land \neg p(s_k) \land p(s_{k+1}) \]
\[ \beta'(k) = \text{simplePath}_{k+1} \land \bigwedge_{i=0}^{k} \neg p(s_i) \land p(s_{k+1}) \]

- \( \beta(k) \) is unsatisfiable only if \( \beta'(k) \) is unsatisfiable
- \( \alpha(k) \) is irrelevant until \( \beta(k) \) becomes unsatisfiable
Efficient implementation

\[ \alpha(k) = I(s_0) \land simplePath_k \land p(s_k) \]
\[ \beta(k) = simplePath_{k+1} \land \neg p(s_k) \land p(s_{k+1}) \]
\[ \beta'(k) = simplePath_{k+1} \land \land_{i=0}^{k} \neg p(s_i) \land p(s_{k+1}) \]

- \( \beta(k) \) is unsatisfiable only if \( \beta'(k) \) is unsatisfiable
- \( \alpha(k) \) is irrelevant until \( \beta(k) \) becomes unsatisfiable
- Leads to the following efficient implementation:
Efficient implementation

\[ \alpha(k) = I(s_0) \land simplePath_k \land p(s_k) \]
\[ \beta(k) = simplePath_{k+1} \land \neg p(s_k) \land p(s_{k+1}) \]
\[ \beta'(k) = simplePath_{k+1} \land \bigwedge_{i=0}^{k} \neg p(s_i) \land p(s_{k+1}) \]

- \( \beta(k) \) is unsatisfiable only if \( \beta'(k) \) is unsatisfiable
- \( \alpha(k) \) is irrelevant until \( \beta(k) \) becomes unsatisfiable
- Leads to the following efficient implementation:
  - Check \( \beta'(k) \) until it becomes unsatisfiable. Then, set \( m = k \)
Efficient implementation

\[ \alpha(k) = I(s_0) \land \text{simplePath}_k \land p(s_k) \]
\[ \beta(k) = \text{simplePath}_{k+1} \land \neg p(s_k) \land p(s_{k+1}) \]
\[ \beta'(k) = \text{simplePath}_{k+1} \land \bigwedge_{i=0}^{k} \neg p(s_i) \land p(s_{k+1}) \]

- \( \beta(k) \) is unsatisfiable only if \( \beta'(k) \) is unsatisfiable
- \( \alpha(k) \) is irrelevant until \( \beta(k) \) becomes unsatisfiable
- Leads to the following efficient implementation:
  - Check \( \beta'(k) \) until it becomes unsatisfiable. Then, set \( m = k \)
  - Check \( \beta(k) \) until it becomes unsatisfiable. Then, check \( \alpha(k) \) until it becomes unsatisfiable. Then, set \( n = k \)
Our Bound is Tight

\[ S_0 \quad p \quad S_i \quad S_{2i+1} \]
Our Bound is Tight

\[ S_0 \xrightarrow{p} S_i \xrightarrow{} S_{2i+1} \]

\[ n = i + 1 \]
Our Bound is Tight

\[ S_0 \quad S_i \quad p \quad S_{2i+1} \]

- \( n = i + 1 \)
- \( m = i + 1 \)
Our Bound is Tight

- $n = i + 1$
- $m = i + 1$
- The minimum counterexample length is

\[ m + n - 1 = 2i + 1 = 2n - 1 \]
Example

[Clarke04]
Our method
Reduction
Example

[Clarke04] \( rr = 10 \quad r = 6 \quad d = 6 \quad k = 11 \)

Our method

Reduction
[Clarke04] \( rr = 10 \quad r = 6 \quad d = 6 \quad k = 11 \)

Our method \( m = 4 \quad n = 6 \quad k = 9 \)

Reduction
Example

[Clarke04] \( rr = 10 \)  \( r = 6 \)  \( d = 6 \)  \( k = 11 \)

Our method \( m = 4 \)  \( n = 6 \)  \( k = 9 \)

Reduction BMC+simple p. \( k = 15 \)
Example

[Clarke04] \( r_{rr} = 10 \quad r = 6 \quad d = 6 \quad k = 11 \)

Our method \( m = 4 \quad n = 6 \quad k = 9 \)

Reduction BMC+simple p. \( k = 15 \)

check_invariant \ forward radius: 6 to 11

reachable states: 11 to 262
LTL model-checking

- To use our termination criterion we reduce LTL model checking to language emptiness.

- The standard approach to model checking an LTL formula consists of:
  - Constructing a Büchi automaton that accepts all counterexamples to the LTL formula.
  - Building the composition of the model with the property automaton.
  - Checking language emptiness on the composed model.

- This approach does not guarantee shortest counterexamples.
The length of counterexample

\[ A \text{ is the Büchi automaton for the negation of } \varphi = G(r \rightarrow F q). \]
The length of counterexample

\[ A \] is the Büchi automaton for the negation of \( \varphi = G(r \rightarrow Fq) \).

The shortest counterexample to \( \varphi \) in \( \mathcal{K} \parallel A \) includes three states, \( a0, b1, \) and \( a1 \).
The length of counterexample

\[ \mathcal{A} \text{ is the Büchi automaton for the negation of } \varphi = G(r \rightarrow F q). \]

- The shortest counterexample to \( \varphi \) in \( \mathcal{K} \parallel \mathcal{A} \) includes three states, \( a0, b1, \) and \( a1 \)
- The shortest counterexample to \( \varphi \) in \( \mathcal{K} \) consisting of two states, \( a \) and \( b \)
The length of counterexample

\( A \) is the Büchi automaton for the negation of
\( \varphi = \mathbf{G}(r \rightarrow \mathbf{F} q) \).

The shortest counterexample to \( \varphi \) in \( K \parallel A \) includes three states, \( a0, b1, \) and \( a1 \).

The shortest counterexample to \( \varphi \) in \( K \) consisting of two states, \( a \) and \( b \).

The counterexample in \( K \parallel A \) may be longer than the one found in \( K \) (by BMC).
Algorithm outline

For each counterexample length $k$

- Use the standard BMC on $\mathcal{K}$ to check for the violation of the property
- Apply the termination criteria to $\mathcal{K} \parallel \mathcal{A}$

The ability to terminate sooner for failing properties offsets the overhead of having two models for each $k$. 
Early termination

In addition to checking the termination criterion mentioned before, our algorithm also checks for
\( \chi(k) = I(s_0) \land simplePath_k \) on \( \mathcal{K} \parallel \mathcal{A} \)

if \( \chi(k) \) is unsatisfiable, then no simple path of length \( k \) can be extended to a counterexample

- It is independent of the position of fair states
- It is equal to the longest shortest path in (Clarke et al. [2004])
- Already in (Sheeran et al. [2000])
Experimental results in VIS

- We compare the performance of our algorithm to
  - The standard BMC algorithm
  - BDD-based LTL model checking algorithm

Summary of the results:
- 43 properties: 20 failing and 23 passing
- BMC decides 19 failing and 0 passing properties
- BMC+termination decides 19 failing and 18 passing properties
- BDD-based MC decides 15 failing and 19 passing properties
- BMC+termination together with BDD-based MC decide all 43
- BMC is faster than BMC+termination for failing properties
Experimental results in VIS

- We compare the performance of our algorithm to
  - The standard BMC algorithm
  - BDD-based LTL model checking algorithm

Summary of the results
- 43 properties: 20 failing and 23 passing
- BMC decides 19 failing and 0 passing properties
- BMC+termination decides 19 failing and 18 passing properties
- BDD-based MC decides 15 failing and 19 passing properties
- BMC+termination together with BDD-based MC decide all 43
- BMC is faster than BMC+termination for failing properties
Experimental result scatterplot
## Experimental results

<table>
<thead>
<tr>
<th>state vars</th>
<th>aut_sat</th>
<th>bmc</th>
<th>ltl</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j = k$</td>
<td>$T(s)$</td>
<td>$k$</td>
<td>$T(s)$</td>
</tr>
<tr>
<td>99</td>
<td>$y = 3$</td>
<td>3.03</td>
<td>?</td>
</tr>
<tr>
<td>102</td>
<td>$y = 1$</td>
<td>1.01</td>
<td>?</td>
</tr>
<tr>
<td>506</td>
<td>$y = 0$</td>
<td>0.4</td>
<td>?</td>
</tr>
<tr>
<td>85</td>
<td>$y = 17$</td>
<td>11.21</td>
<td>?</td>
</tr>
<tr>
<td>293</td>
<td>$y = 19$</td>
<td>44.43</td>
<td>?</td>
</tr>
<tr>
<td>27</td>
<td>$y = 5$</td>
<td>13.16</td>
<td>?</td>
</tr>
<tr>
<td>307</td>
<td>$y = 5$</td>
<td>5.53</td>
<td>?</td>
</tr>
<tr>
<td>37</td>
<td>$y = 9$</td>
<td>0.95</td>
<td>?</td>
</tr>
<tr>
<td>74</td>
<td>$y = 3$</td>
<td>0.57</td>
<td>?</td>
</tr>
<tr>
<td>48</td>
<td>$y = 3$</td>
<td>2.5</td>
<td>?</td>
</tr>
<tr>
<td>18</td>
<td>$y = 8$</td>
<td>12.6</td>
<td>?</td>
</tr>
</tbody>
</table>
An Improved Criterion

If $S_8$ is the only state satisfying $p$, then $m = 8$. Such a path cannot be used to close the loop because its initial state does not satisfy $p$.

The longest simple path whose first state satisfies $p$, such that $p$ does not hold in any subsequent state has length 5. This path is sufficient to take $m = 5$.

To capture this observation, we add

$$
\text{simplePath}_{k+1}^p(s_0)^{V_{k+1}}_{i=1}^p(s_i)
$$
An Improved Criterion

If $S_8$ is the only state satisfying $p$, then $m = 8$
If $S_8$ is the only state satisfying $p$, then $m = 8$

Such a path cannot be used to close the loop because its initial state does not satisfy $p$
The longest simple path whose first state satisfies $p$, such that $p$ does not hold in any subsequent state has length 5.
The longest simple path whose first state satisfies $p$, such that $p$ does not hold in any subsequent state has length 5

This path is sufficient to take $m = 5$
The longest simple path whose first state satisfies $p$, such that $p$ does not hold in any subsequent state has length 5.

This path is sufficient to take $m = 5$.

To capture this observation, we add

$$\beta''(k) = \text{simplePath}_{k+1} \land p(s_0) \land \bigwedge_{i=1}^{k+1} \neg p(s_i)$$
An Improved Criterion

\[ \beta''(k) \text{ does not replace } \beta'(k) \]
An Improved Criterion

- $\beta''(k)$ does not replace $\beta'(k)$
- If we move $p$ from state $S_8$ to state $S_2$, ...
An Improved Criterion

- $\beta''(k)$ does not replace $\beta'(k)$
- If we move $p$ from state $S_8$ to state $S_2$,
  - $\beta'_k$ will capture the smallest value of $m$
In general
**In general**

- If the distance between \( p \)-states in the loops < the stem paths leading to them, \( \beta'' \) captures the smaller value of \( m \)
In general

- If the distance between \( p \)-states in the loops < the stem paths leading to them, \( \beta'' \) captures the smaller value of \( m \)
- When the \( p \) states appear only on the stem paths, \( \beta' \) will often capture the smaller value of \( m \)
An Improved Criterion

In general

- If the distance between $p$-states in the loops $< \text{the stem paths leading to them}$, $\beta''$ captures the smaller value of $m$.

- When the $p$ states appear only on the stem paths, $\beta'$ will often capture the smaller value of $m$.

Hence, we check both $\beta'(k)$ and $\beta''(k)$ and set the value of $m$ when either $\beta'(k)$ or $\beta''(k)$ becomes unsatisfiable.
Experimental result scatterplot
Automaton Reachability analysis

<table>
<thead>
<tr>
<th>Model</th>
<th>No</th>
<th>Yes</th>
<th>Model</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheCo</td>
<td>29.28</td>
<td>21.68</td>
<td>D4</td>
<td>29.31</td>
<td>26.34</td>
</tr>
<tr>
<td>Ethernet</td>
<td>109.53</td>
<td>96.7</td>
<td>Heap</td>
<td>513.12</td>
<td>421.92</td>
</tr>
<tr>
<td>HourGlass</td>
<td>504.55</td>
<td>478.87</td>
<td>Needham</td>
<td>1540.1</td>
<td>1369.8</td>
</tr>
<tr>
<td>PI_BUS</td>
<td>377.75</td>
<td>366.92</td>
<td>ProdCell</td>
<td>85.47</td>
<td>118.44</td>
</tr>
<tr>
<td>ReqAck</td>
<td>101.85</td>
<td>113.27</td>
<td>TwoFifo</td>
<td>95.75</td>
<td>87.98</td>
</tr>
<tr>
<td>TwoQ</td>
<td>235.37</td>
<td>225.58</td>
<td>VsaR1</td>
<td>353.99</td>
<td>366.61</td>
</tr>
</tbody>
</table>

It usually reduces runtime, but it does not help in reducing the values of \( m \) and \( n \).
## Automaton Reachability analysis

<table>
<thead>
<tr>
<th>Model</th>
<th>No</th>
<th>Yes</th>
<th>Model</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheCo</td>
<td>29.28</td>
<td>21.68</td>
<td>D4</td>
<td>29.31</td>
<td>26.34</td>
</tr>
<tr>
<td>Ethernet</td>
<td>109.53</td>
<td>96.7</td>
<td>Heap</td>
<td>513.12</td>
<td>421.92</td>
</tr>
<tr>
<td>HourGlass</td>
<td>504.55</td>
<td>478.87</td>
<td>Needham</td>
<td>1540.1</td>
<td>1369.8</td>
</tr>
<tr>
<td>PI_BUS</td>
<td>377.75</td>
<td>366.92</td>
<td>ProdCell</td>
<td>85.47</td>
<td>118.44</td>
</tr>
<tr>
<td>ReqAck</td>
<td>101.85</td>
<td>113.27</td>
<td>TwoFifo</td>
<td>95.75</td>
<td>87.98</td>
</tr>
<tr>
<td>TwoQ</td>
<td>235.37</td>
<td>225.58</td>
<td>VsaR1</td>
<td>353.99</td>
<td>366.61</td>
</tr>
</tbody>
</table>

*It usually reduces runtime, but it does not help in reducing the values of $m$ and $n$*
Tight Büchi Automaton

(Schuppan and Biere [2005]) shows that the construction of (Clarke et al. [1994]) yields a tight Büchi automaton
Tight Büchi Automaton

\[ \text{(Schuppan and Biere [2005]) shows that the construction of (Clarke et al. [1994]) yields a tight Büchi automaton} \]

\[ \text{The composition of tight Büchi automaton and model will map onto shortest counterexamples in the original model} \]
Tight Büchi Automaton

(Schuppan and Biere [2005]) shows that the construction of (Clarke et al. [1994]) yields a tight Büchi automaton.

The composition of tight Büchi automaton and model will map onto shortest counterexamples in the original model.

Question: Would tight Büchi automata benefits our termination criterion?
### Tight and non-tight automata

<table>
<thead>
<tr>
<th>Model</th>
<th>Non-tight</th>
<th>Tight</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m$</td>
<td>$n$</td>
</tr>
<tr>
<td>Coherence</td>
<td>17</td>
<td>-</td>
</tr>
<tr>
<td>Ifetch1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>FPMult1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Microwave</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pathfinder</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PI_BUS</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>s1269-2</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>s1423</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

Most of the times the use of tight automata increases the termination length (increases both $m$ and $n$).
### Tight and non-tight automata

<table>
<thead>
<tr>
<th>Model</th>
<th>Non-tight</th>
<th></th>
<th>Tight</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m$</td>
<td>$n$</td>
<td>$T(s)$</td>
<td>$m$</td>
</tr>
<tr>
<td>Coherence</td>
<td>17</td>
<td>-</td>
<td>TO</td>
<td>24</td>
</tr>
<tr>
<td>Ifetch1</td>
<td>0</td>
<td>2</td>
<td>0.31</td>
<td>-</td>
</tr>
<tr>
<td>FPMult1</td>
<td>0</td>
<td>3</td>
<td>5.77</td>
<td>2</td>
</tr>
<tr>
<td>Microwave</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>Pathfinder</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>PI_BUS</td>
<td>0</td>
<td>1</td>
<td>0.57</td>
<td>0</td>
</tr>
<tr>
<td>s1269-2</td>
<td>0</td>
<td>8</td>
<td>0.20</td>
<td>0</td>
</tr>
<tr>
<td>s1423</td>
<td>0</td>
<td>4</td>
<td>0.16</td>
<td>3</td>
</tr>
</tbody>
</table>

Most of the times the use of tight automata increases the termination length (increases both $m$ and $n$)
The reasons

- Sharp increase in the number of states and transitions
The reasons

- Sharp increase in the number of states and transitions
- A tight automaton has exactly one acceptance condition for each $U$ operator in the LTL formula
The reasons

- Sharp increase in the number of states and transitions
- A tight automaton has exactly one acceptance condition for each $U$ operator in the LTL formula
- Tight automata are almost always strong

\begin{figure}
\centering
\begin{tikzpicture}
  \node [state] (p) {$p$};
  \node [state] (negp) {$\neg p$};
  \path[->] (negp) edge [bend left, above] (p);
  \path[->] (negp) edge [bend right, below] (p);
  \path[->] (p) edge [bend left, above] (negp);
  \path[->] (p) edge [bend right, below] (negp);

  \node [state] (true) at (2,0) {true};
  \path[->] (true) edge [bend right, above] (true);
  \path[->] (true) edge [bend left, below] (true);

  \node [state] (strong) at (-3,0) {strong};
  \node [state] (weak) at (0,0) {weak};
\end{tikzpicture}
\end{figure}
Strength and $m$

\[ \neg G F p \]
\[ \neg (p \cup q) \]
\[ \neg G(p \rightarrow F q) \]
Multiple Fairness Conditions

- Our approach for checking language emptiness only considers Büchi automata with one fair set.
Multiple Fairness Conditions

- Our approach for checking language emptiness only considers Büchi automata with one fair set.
- Büchi automata with multiple fair sets are known as generalized Büchi automata.
Multiple Fairness Conditions

One solution is to convert a generalized Büchi automaton to an equivalent Büchi automaton.
Multiple Fairness Conditions

- One solution is to convert a generalized Büchi automaton to an equivalent Büchi automaton
- Counter-based approach
Multiple Fairness Conditions

- One solution is to convert a generalized Büchi automaton to an equivalent Büchi automaton
- Counter-based approach
  - The conversion expands the size of the automaton by a factor related to the number of fair sets
Multiple Fairness Conditions

- One solution is to convert a generalized Büchi automaton to an equivalent Büchi automaton
- Counter-based approach
  - The conversion expands the size of the automaton by a factor related to the number of fair sets
  - The effectiveness of our termination criterion depends on the order at which the fair sets are visited

Flag-based approach
- Assign a flag $x_j$ for each fair set $F_j$ which is set when a state $p_j$ in $F_j$ is visited
- The value of $x_j$ at time $t$ is given as:
  - $x_j(0) = p_j(0)$
  - $x_j(t) = p_j(t) \cup x_j(t-1)^\wedge$

A fair state in the Büchi automaton is one that satisfies $x = V_r = 1$.
Multiple Fairness Conditions

- One solution is to convert a generalized Büchi automaton to an equivalent Büchi automaton
  - Counter-based approach
  - Flag-based approach
Multiple Fairness Conditions

- One solution is to convert a generalized Büchi automaton to an equivalent Büchi automaton.
- Counter-based approach
- Flag-based approach
  - Assign a flag $x_j$ for each fair set $F_j$ which is set when a state $p_j$ in $F_j$ is visited.
Multiple Fairness Conditions

- One solution is to convert a generalized Büchi automaton to an equivalent Büchi automaton
  - Counter-based approach
  - Flag-based approach
    - Assign a flag $x_j$ for each fair set $F_j$ which is set when a state $p_j$ in $F_j$ is visited
    - The value of $x_j$ at time $t$ is given as

\[
x_j(0) = p_j(0)
\]
\[
x_j(t) = p_j(t) \lor x_j(t - 1) \land \neg x(t - 1)
\]
Multiple Fairness Conditions

- One solution is to convert a generalized Büchi automaton to an equivalent Büchi automaton
- Counter-based approach
- Flag-based approach
  - Assign a flag $x_j$ for each fair set $F_j$ which is set when a state $p_j$ in $F_j$ is visited
  - The value of $x_j$ at time $t$ is given as
    \[
    x_j(0) = p_j(0) \\
    x_j(t) = p_j(t) \lor x_j(t - 1) \land \neg x(t - 1) .
    \]

- A fair state in the Büchi automaton is one that satisfies $\mathbf{x} = \bigwedge_{j=1}^{r} x_j$. 
Example

Counter:

\[
x_1 x_2 x : 000 \quad 000 \quad 000 \quad 010 \quad 010 \quad 111
\]
Multiple Fairness Conditions

Another solution is to check language emptiness for the fairness condition $\bigcup_{F \in \mathcal{F}} F$ (Or approach).
Multiple Fairness Conditions

- Another solution is to check language emptiness for the fairness condition $\bigcup_{F \in \mathcal{F}} F$ (Or approach).
- It is in general conservative in estimating the termination length.
Multiple Fairness Conditions

- Another solution is to check language emptiness for the fairness condition \( \bigcup_{F \in \mathcal{F}} F \) (Or approach)
  - It is in general conservative in estimating the termination length
  - It does not require any increase in the number of states of the automaton
Multiple Fairness Conditions

Another solution is to check language emptiness for the fairness condition $\bigcup_{F \in \mathcal{F}} F$ (Or approach)

- It is in general conservative in estimating the termination length
- It does not require any increase in the number of states of the automaton
- It may decrease the distance between fair states along the loops and hence reduce the value of $m$
Multiple Fairness Conditions

- Another solution is to check language emptiness for the fairness condition $\bigcup_{F \in \mathcal{F}} F$ (Or approach)
  - It is in general conservative in estimating the termination length
  - It does not require any increase in the number of states of the automaton
  - It may decrease the distance between fair states along the loops and hence reduce the value of $m$

- Another approach is to apply out termination criteria to each fairness condition in turn (One approach)
Multiple Fairness Conditions

- Another solution is to check language emptiness for the fairness condition $\bigcup_{F \in \mathcal{F}} F$ (Or approach)
  - It is in general conservative in estimating the termination length
  - It does not require any increase in the number of states of the automaton
  - It may decrease the distance between fair states along the loops and hence reduce the value of $m$

- Another approach is to apply out termination criteria to each fairness condition in turn (One approach)
  - It produces conservative values for $m$ and $n$
Multiple Fairness Conditions

- Another solution is to check language emptiness for the fairness condition $\bigcup_{F \in \mathcal{F}} F$ (Or approach)
  - It is in general conservative in estimating the termination length
  - It does not require any increase in the number of states of the automaton
  - It may decrease the distance between fair states along the loops and hence reduce the value of $m$

- Another approach is to apply out termination criteria to each fairness condition in turn (One approach)
  - It produces conservative values for $m$ and $n$
  - It is more effective than the Or approach when one fairness constraint cannot be satisfied in isolation
Or, One, and Flag

- 20 out of 25 properties are decided passed by at least one method within the given limit of time and value of $k$.
- Both methods One and Flag are the fastest in 8 experiments.
- Or is the fastest in only 3 experiments.
- No methods dominates the other in reducing the length of $m$ and $n$.
- The Flag method proves properties pass in zero value of $k$ in 6 experiments.
<table>
<thead>
<tr>
<th>#</th>
<th>Or</th>
<th>One</th>
<th>Flag</th>
<th>Counter</th>
<th>Trans.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(k)</td>
<td>(T(s))</td>
<td>(k)</td>
<td>(T(s))</td>
<td>(k)</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>8.5</td>
<td>13</td>
<td>25.05</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>720.51</td>
<td>0</td>
<td>0.05</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>0.15</td>
<td>5</td>
<td>0.64</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0.3</td>
<td>2</td>
<td>0.12</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
<td>101.86</td>
<td>0</td>
<td>299.74</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
A counterexample to a simple liveness, \( Fp \), is an infinite initialized path along which \( p \) never holds.
A counterexample to a simple liveness, $F_p$, is an infinite initialized path along which $p$ never holds

A counterexamples to $F_p$ is bounded by the longest simple path starting from an initial state (Kröning and Strichman [2003])
A counterexample to a simple liveness, $Fp$, is an infinite initialized path along which $p$ never holds.

A counterexamples to $Fp$ is bounded by the longest simple path starting from an initial state (Kröning and Strichman [2003])

Reduction of liveness to safety (Biere et al. [2002]) can also be used.
A counterexample to a simple liveness, $\mathcal{F}p$, is an infinite initialized path along which $p$ never holds.

A counterexamples to $\mathcal{F}p$ is bounded by the longest simple path starting from an initial state (Kröning and Strichman [2003]).

Reduction of liveness to safety (Biere et al. [2002]) can also be used.

(Schuppan and Biere [2004]) presents a tight bound:

- $\neg p$-predicated radius + $\neg p$-predicated diameter
Our contribution

\( \text{If we let} \ n \ \text{captures the length of the longest initial simple path along which} \ p \ \text{holds.} \)

Then, \( n \) is the least value of \( k \) for which

\[
(\forall k \geq n) \left( \neg \left( \exists s \in \text{simplePath}_k \right) \right) = I(s_0)^\text{simplePath}_k^V_0
\]

\( \text{If there is no counterexample of length} \ n \), then \( M_j = F_p \).
Our contribution

\[ Fp \] passes if starting from initial states and applying the transition relation always reaches \( p \)
If we let $n$ captures the length of the longest initial simple path along which $\neg p$ holds.
Our contribution

If we let $n$ captures the length of the longest initial simple path along which $\neg p$ holds

Then, $n$ is the least value of $k$ for which $\theta(k)$ is unsatisfiable

$\theta(k) = I(s_0) \land \text{simplePath}_k \land \land_{0 \leq i \leq k} \neg p(s_i)$
If we let $n$ captures the length of the longest initial simple path along which $\neg p$ holds.

Then, $n$ is the least value of $k$ for which $\theta(k)$ is unsatisfiable.

$\theta(k) = I(s_0) \land \text{simplePath}_k \land \bigwedge_{0 \leq i \leq k} \neg p(s_i)$

If there is no counterexample of length $\leq n$, then $\mathcal{M} \models Fp$.
## Experimental results

<table>
<thead>
<tr>
<th>state vars</th>
<th>General case</th>
<th>$F_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\models k$</td>
<td>$Time(s)$</td>
</tr>
<tr>
<td>102</td>
<td>yes 1</td>
<td>2.4</td>
</tr>
<tr>
<td>230</td>
<td>? 30</td>
<td>356.7</td>
</tr>
<tr>
<td>506</td>
<td>yes 1</td>
<td>14.66</td>
</tr>
<tr>
<td>307</td>
<td>yes 6</td>
<td>18.89</td>
</tr>
<tr>
<td>18</td>
<td>yes 9</td>
<td>22.09</td>
</tr>
</tbody>
</table>
Summary

- We have presented a termination criterion for language emptiness that takes into account the position of fair states.
- We also have presented a termination check for simple liveness properties \((\mathbf{F} p)\).
- The experimental results show that BMC augmented with our termination criteria is a complementary technique to BDD-based LTL model checking.
- Taking into consideration the positions of fair states reduces the termination depth.
- The termination check should be done while checking for a counterexample.
Summary

- We have presented an improved criterion for termination in Bounded Model Checking, which significantly reduces termination length.

- We have also shown that the use of the reachability analysis of the property automaton speeds up the termination check, but does not reduce the termination length.

- Even though tight automata find shortest $k$-loop counterexamples, they increase the termination length.

- An improvement to our termination check could be restricting the search using the Flag method to paths that end in a state that satisfies at least one fairness constraint.
Summary

- We have presented different methods for checking multiple fairness conditions.

- The Flag and One methods are the best.

- Both Flag and Counter methods are based on recording the visiting of fair states.

- The performance of the Counter method depends on the order of visit of the fair sets.

- The Flag method limits the search for a simple path to the ones that satisfy all fairness constraints.

- It finds small values for $m$ and $n$, but it may increase the searching time.

- The efficiency of the One method also depends on the order in which fair sets are checked.

- In practice, it is more efficient to check the fair states that come from the property automaton before those supplied with the model.
Summary

- We have presented different methods for checking multiple fairness conditions
- The Flag and One methods are the best
Summary

- We have presented different methods for checking multiple fairness conditions
  - The Flag and One methods are the best
  - Both Flag and Counter methods are based on recording the visiting of fair states
Summary

- We have presented different methods for checking multiple fairness conditions
  - The Flag and One methods are the best
  - Both Flag and Counter methods are based on recording the visiting of fair states
  - The performance of the Counter method depends on the order of visit of the fair sets
Summary

- We have presented different methods for checking multiple fairness conditions
  - The Flag and One methods are the best
  - Both Flag and Counter methods are based on recording the visiting of fair states
  - The performance of the Counter method depends on the order of visit of the fair sets
  - The Flag method limits the search for a simple path to the ones satisfy all fairness constraints
Summary

- We have presented different methods for checking multiple fairness conditions
  - The Flag and One methods are the best
  - Both Flag and Counter methods are based on recording the visiting of fair states
  - The performance of the Counter method depends on the order of visit of the fair sets
  - The Flag method limits the search for a simple path to the ones satisfy all fairness constraints
    - it finds small values for $m$ and $n$, but it may increase the searching time
Summary

- We have presented different methods for checking multiple fairness conditions
  - The Flag and One methods are the best
  - Both Flag and Counter methods are based on recording the visiting of fair states
  - The performance of the Counter method depends on the order of visit of the fair sets
  - The Flag method limits the search for a simple path to the ones satisfy all fairness constraints
  - it finds small values for $m$ and $n$, but it may increase the searching time
  - The efficiency of the One method also depends on the order in which fair sets are checked
Summary

- We have presented different methods for checking multiple fairness conditions
  - The Flag and One methods are the best
  - Both Flag and Counter methods are based on recording the visiting of fair states
  - The performance of the Counter method depends on the order of visit of the fair sets
  - The Flag method limits the search for a simple path to the ones satisfy all fairness constraints
    - it finds small values for $m$ and $n$, but it may increase the searching time
  - The efficiency of the One method also depends on the order in which fair sets are checked
    - in practice, it is more efficient to check the fair states that come from the property automaton before those supplied with the model
Automatic Invariant Strengthening
Proving Invariant

- A property $\psi$ is an invariant property if it holds in all reachable states.
Proving Invariant

- A property \( \psi \) is an invariant property if it holds in all reachable states
- Specified in LTL as \( G\ p \), where \( p \) is a propositional formula
A property $\psi$ is an invariant property if it holds in all reachable states.

Specified in LTL as $G \, p$, where $p$ is a propositional formula.

An invariant is inductive if it holds in all initial states of the model and in all successors of states that satisfy the invariant.
A property \( \psi \) is an invariant property if it holds in all reachable states.

Specified in LTL as \( \text{G} p \), where \( p \) is a propositional formula.

An invariant is inductive if it holds in all initial states of the model and in all successors of states that satisfy the invariant.

Invariants may be proved inductive by
Proving Invariant

- A property $\psi$ is an invariant property if it holds in all reachable states.

- Specified in LTL as $\mathsf{G} p$, where $p$ is a propositional formula.

- An invariant is **inductive** if it holds in all initial states of the model and in all successors of states that satisfy the invariant.

- Invariants may be proved inductive by
  - BDD-based fixpoint technique
  - backward reachability started from the states violating the invariant.
A property $\psi$ is an invariant property if it holds in all reachable states.

Specified in LTL as $\text{G } p$, where $p$ is a propositional formula.

An invariant is **inductive** if it holds in all initial states of the model and in all successors of states that satisfy the invariant.

Invariants may be proved inductive by:

- BDD-based fixpoint technique
  - backward reachability started from the states violating the invariant
- SAT-based technique
  - $\text{UNSAT}(I(s) \land \neg p(s))$ and,
  - $\text{UNSAT}(p(s) \land T(s, t) \land \neg p(t))$
Proving Invariant

- Invariants in real designs are rarely inductive.

- Need to make the invariants stronger (smaller) by excluding unreachable good stats with bad successors.

- One fundamental technique is to use an auxiliary invariant whose conjunction with the desired invariant makes it inductive. The ultimate auxiliary invariant is the set of reachable states, which is often prohibitively expensive other resources.

- The termination criteria commonly employed by BMC is 

awedh – p. 66
Proving Invariant

- Invariants in real designs are rarely inductive
- Need to make the invariants stronger (smaller) by excluding unreachable **good** stats with **bad** successors
Invariants in real designs are rarely inductive.

Need to make the invariants stronger (smaller) by excluding unreachable good stats with bad successors.

One fundamental technique is to use an auxiliary invariant.
Proving Invariant

- Invariants in real designs are rarely inductive.
- Need to make the invariants stronger (smaller) by excluding unreachable good stats with bad successors.
- One fundamental technique is to use an auxiliary invariant.
  - its conjunction with the desired invariant makes it inductive.
Proving Invariant

- Invariants in real designs are rarely inductive
- Need to make the invariants stronger (smaller) by excluding unreachable good stats with bad successors
- One fundamental technique is to use an auxiliary invariant
  - its conjunction with the desired invariant makes it inductive
  - the ultimate auxiliary invariant is the set of reachable states which is often prohibitively expensive
Proving Invariant

- Invariants in real designs are rarely inductive.
- Need to make the invariants stronger (smaller) by excluding unreachable good stats with bad successors.
- One fundamental technique is to use an auxiliary invariant.
  - Its conjunction with the desired invariant makes it inductive.
  - The ultimate auxiliary invariant is the set of reachable states which is often prohibitively expensive.
  - Other resources is the termination criteria commonly employed by BMC.
A counterexample to an invariant is a finite prefix path to a state that satisfies $\neg p$ (bad state).

Proving Invariant with BMC

If a counterexample exists, then there is a simple path from an initial state to a bad state that goes through no other initial or bad state.
Proving Invariant with BMC

- A counterexample to an invariant is a finite prefix path to a state that satisfies $\neg p$ (bad state).

- If a counterexample exists, then there is a simple path from an initial state to a bad state that goes through no other initial or bad state.
Proving Invariant with BMC

- A counterexample to an invariant is a finite prefix path to a state that satisfies $\neg p$ (bad state).
- If a counterexample exists, then there is a simple path from an initial state to a bad state that goes through no other initial or bad state.
- An invariant holds \textit{Sheeran et al. [2000]} if:
A counterexample to an invariant is a finite prefix path to a states that satisfies $\neg p$ (bad state).

If a counterexample exists, then there is a simple path from an initial state to a bad state that goes through no other initial or bad state.

An invariant holds Sheeran et al. [2000] if:
- there is no counterexample of length $k$ to $\neg p$, and
Proving Invariant with BMC

- A counterexample to an invariant is a finite prefix path to a states that satisfies $\neg p$ (bad state)
- If a counterexample exists, then there is a simple path from an initial state to a bad state that goes through no other initial or bad state
- An invariant holds Sheeran et al. [2000] if:
  - there is no counterexample of length $k$ to $\neg p$, and
  - no simple path of length $k + 1$ to $\neg p$ that is not going through any other states satisfy $\neg p$
Eliminating states that have some bad state among their successors at distance $k$ or less
Inv. Strengthening: Our Approach

- Eliminating states that have some bad state among their successors at distance $k$ or less
- Obtaining sets of unreachable states
Inv. Strengthening: Our Approach

- Eliminating states that have some bad state among their successors at distance $k$ or less
- Obtaining sets of unreachable states
  - as byproducts of failed attempts to prove termination, and
Inv. Strengthening: Our Approach

- Eliminating states that have some bad state among their successors at distance $k$ or less
- Obtaining sets of unreachable states
  - as byproducts of failed attempts to prove termination, and
  - extending those byproducts with SAT-based unbounded model checking techniques

Knowing that a set of states is not reachable allows to exclude it from the search for a counterexample and simple paths. Knowing that a set of states is reachable signals that the invariant fails.
Inv. Strengthening: Our Approach

- Eliminating states that have some bad state among their successors at distance $k$ or less
- Obtaining sets of unreachable states
  - as byproducts of failed attempts to prove termination, and
  - extending those byproducts with SAT-based unbounded model checking techniques
- Knowing that a set of states is not reachable allows to exclude it from the search for a counterexample and simple paths
Inv. Strengthening: Our Approach

- Eliminating states that have some bad state among their successors at distance $k$ or less
- Obtaining sets of unreachable states
  - as byproducts of failed attempts to prove termination, and
  - extending those byproducts with SAT-based unbounded model checking techniques

Knowing that a set of states is not reachable allows to exclude it from the search for a counterexample and simple paths

Knowing that a set of states is reachable signals that the invariant fails
while (true) {[q = ∅; C = ∅; R = ∅; k = 0; flag = 0]
    \(c_e_k = \text{searchForCounterExample}(\neg p \cup C, R)\);
    if (\(c_e_k\))
        if (\(c_e_k[k] \in \neg p\)) return (fail, \(c_e_k\))
        else flag = 1;
    if (flag == 1) continue;
    t = (q == ∅)?\neg p : q;
    \(s_{p_{k+1}} = \text{searchForSimplePath}(p, t, R)\);
    if (\(s_{p_{k+1}}\))
        if (q == ∅) {
            q = \(s_{p_{k+1}}[k + 1]\);
            C = \{\(s_{p_{k+1}}[0, \ldots, k + 1]\}\};
        } else C = C ∪ \{\(s_{p_{k+1}}[0, \ldots, k]\}\};
    } else if (q == ∅) return (pass);
    else R = R ∪ \{E(\neg (R \cup p) \cup C)\};
    \(q == ∅; C = ∅;\)
    k = k + 1;
Prune the search space

- Shows unreachable states of a model
Prune the search space

Shows unreachable states of a model

States violate $p$
Prune the search space

- Shows unreachable states of a model
- States violate $p$
- The longest simple path
Prune the search space

- Pick up a state
Prune the search space

- Pick up a state
- Prove it is useless
Prune the search space

Compute backward reachability
Prune the search space

- Compute backward reachability
- Remove from the search space
Prune the search space

- Compute backward reachability
- Remove from the search space
- The new longest simple path
Prune the search space

Pick up another state
Prune the search space

- Pick up another state
- Prove it is useless
Prune the search space

Compute backward reachability
Prune the search space

- Compute backward reachability
- Remove from the search space
Prune the search space

- Compute backward reachability
- Remove from the search space
- The new longest simple path
Proving the Existence of a $\mathcal{CE}$
Proving the Existence of a CE

No counterexample of length 0
Proving the Existence of a $\mathcal{CE}$

- No counterexample of length 0
- Find a simple path of length 1
Proving the Existence of a $CE$

- No counterexample of length 0
- Find a simple path of length 1
- Add the states in the simple path to the target states
Proving the Existence of a $CE$

$\exists I$

No counterexample of length 1
No counterexample of length 1
Find a simple path of length 2
Proving the Existence of a $CE$

- No counterexample of length 1
- Find a simple path of length 2
- Add the states in the simple path to the target states
Proving the Existence of a \text{CE}:

No counterexample of length 2
Proving the Existence of a $\mathcal{CE}$

- No counterexample of length 2
- Find a simple path of length 3
Proving the Existence of a CE

- No counterexample of length 2
- Find a simple path of length 3
- Add the states in the simple path to the target states
Proving the Existence of a $CE$

Find a path of length 3 to a target state
Proving the Existence of a $CE$

- Find a path of length 3 to a target state
- Signal the existence of a counterexample
Soundness and Completeness

Lemma 1

If no counterexample to $G_p$ of length $k$, and no simple path into state $s$ of length $k + 1$, then no counterexample to $G_p$ has $s$ as the first state violating the invariant $I$.

If $t$ has a path reaching $s$ without any other state satisfying $p$ besides $s$ if $t$ is on any shortest counterexample to $G_p$, it is preceded by one state violating the invariant $I$. 

\( \neg p \)
Soundness and Completeness

Lemma 1

If no counterexample to $G_p$ of length $k$, and

- If $s$ has a path reaching $t$ without any other state $s$ satisfying $p$ besides $s$ if $t$ is on any shortest counterexample to $G_p$, it is preceded by one state violating the invariant.
Lemma 1

If no counterexample to $Gp$ of length $k$, and
No simple path into state $s$ of length $k + 1$
Lemma 1

If no counterexample to $Gp$ of length $k$, and
No simple path into state $s$ of length $k + 1$
Then, no counterexample to $Gp$ has $s$ as the first state violating the invariant
Soundness and Completeness

Lemma 2

If \( t \) has a path reaching \( s \) without any other state satisfying \( \neg p \) besides \( s \)
Soundness and Completeness

Lemma 2

If \( t \) has a path reaching \( s \) without any other state satisfying \( \neg p \) besides \( s \)

If \( t \) is on any shortest counterexample to \( Gp \), it is preceded by one state violating the invariant
Soundness and Completeness

- From the assumption that the standard BMC augmented with termination condition is sound and complete, and
Soundness and Completeness

- From the assumption that the standard BMC augmented with termination condition is sound and complete, and

- From Lemma2
Soundness and Completeness

- From the assumption that the standard BMC augmented with termination condition is sound and complete, and

- From Lemma2
  - No Shortest counterexample is missed by adding $R$ as a constraint on the search
Soundness and Completeness

- From the assumption that the standard BMC augmented with termination condition is sound and complete, and

- From Lemma2
  - No Shortest counterexample is missed by adding $R$ as a constraint on the search
  - Any missed simple paths are not reachable because $R$ is known to consist of unreachable states only
Soundness and Completeness

- From the assumption that the standard BMC augmented with termination condition is sound and complete, and
- From Lemma2
  - No Shortest counterexample is missed by adding \( R \) as a constraint on the search
  - Any missed simple paths are not reachable because \( R \) is known to consist of unreachable states only
- If a path exists from an initial state to a state in \( C \), this path is a counterexample because every state in \( C \) is either violating \( p \) or has a path to a state violating \( p \)
Experimental results in VIS

- We compare the performance of our algorithm to
  - The standard BMC algorithm
  - BDD-based LTL model checking algorithms
    - ci
    - mc -D0

- Summary of the results
  - 26 properties: 12 failing, 12 passing and 2 undecided
  - 9 properties are proving by BMC; our new methods is faster in 8
  - our method is also faster in 11 out of 12 failing properties
Experimental results in VIS

- We compare the performance of our algorithm to
  - The standard BMC algorithm
  - BDD-based LTL model checking algorithms
    - ci
    - mc -D0

Summary of the results
- 26 properties: 12 failing, 12 passing and 2 undecided
- 9 properties are proving by BMC; our new methods is faster in 8
- our method is also faster in 11 out of 12 failing properties
Summary

- A new method that improves the performance of BMC
  - it attempts to learn the reachability of states while searching for counterexamples and simple paths
  - it uses reachability information to strengthen the invariant so as to improve the search

Our method may prove the existence of a counterexample very early so that one can stop the check for termination.

Our experimental results show that often our method performs substantially better for both passing and failing properties.

The principles presented in this work may be applied to checking general linear-time temporal properties according to the approach of Awedh and Somenzi [2004, 2005] (Our future work).
A new method that improves the performance of BMC

- it attempts to learn the reachability of states while searching for counterexamples and simple paths
- it uses reachability information to strengthen the invariant so as to improve the search

Our method may prove the existence of a counterexample very early so that one can stop the check for termination.
Summary

- A new method that improves the performance of BMC
  - it attempts to learn the reachability of states while searching for counterexamples and simple paths
  - it uses reachability information to strengthen the invariant so as to improve the search

- Our method may prove the existence of a counterexample very early so that one can stop the check for termination.

- Our experimental results show that often our method performs substantially better for both passing and failing properties
Summary

- A new method that improves the performance of BMC
- it attempts to learn the reachability of states while searching for counterexamples and simple paths
- it uses reachability information to strengthen the invariant so as to improve the search

Our method may prove the existence of a counterexample very early so that one can stop the check for termination.

Our experimental results show that often our method performs substantially better for both passing and failing properties.

The principles presented in this work may be applied to checking general linear-time temporal properties according to the approach of Awedh and Somenzi [2004, 2005] (Our future work)
Increase the Robustness of BMC
Motivation

- Three major factors affect the performance of BMC:
  - The encoding of the model checking problem into a propositional formula
  - The performance of SAT solvers
  - The length of the counterexample and the termination depth
Our technique

- Integrates BDD-based reachability analysis with SAT-based BMC:
  - It uses BDD-based reachability analysis to find cheap lower bounds on the reachable states and the states that reach the “bad” states
  - Then, it uses these lower bounds to reduce the amount of unrolling of the transition relation needed to find the counterexample or decide termination
- It simplifies the SAT instances, while using BDDs only for manageable subsets of the reachable states
The bdd_sat algorithm

$R = I$
The bdd_sat algorithm

\[ R = I \cup N_1 \]
The `bdd_sat` algorithm

\[ R = I \cup N_1 \]
The bdd_sat algorithm

\[ R = I \cup N_1 \cup N_2 \]
The bdd_sat algorithm

\[ R = I \cup N_1 \cup N_2 \]
The bdd_sat algorithm

\[ R = I \cup N_1 \cup N_2 \ldots N_i \]
The bdd_sat algorithm
The bdd_sat algorithm

\[ S_i = \text{Image}(R) - R \]
The bdd_sat algorithm

\[ S_i = \text{Image}(R) - R \]

\[ S_t = \text{Image}(R) - R \]
The bdd_sat algorithm

\[ S_i = \text{Image}(R) - R \]

\[ S_t = \text{Image}(R) - R \]
The bdd_sat algorithm

\[ S_i = \text{Image}(R) - R \]

\[ S_t = \text{Image}(R) - R \]
The bdd_sat algorithm

\[ bdd\_sat(I, T) \{ \]
\[ \quad (status, newI) = ComputeReachFwd(I, T); \]
\[ \quad \text{if} \ (status == \text{undecided}) \{ \]
\[ \quad \quad (status, newT) = ComputeReachBwd(T, newI); \]
\[ \quad \quad \text{if} \ (status == \text{undecided}) \{ \]
\[ \quad \quad \quad status = BMC^+(newI, newT); \]
\[ \quad \quad \} \]
\[ \quad \} \]
\[ \quad \text{return} \ status; \]
\[ \} \]
\textbf{ComputeReachFwd}(I, T)

\begin{align*}
\text{if}(I \cap T \neq \emptyset) & \text{ return}(\text{fail}); \\
R &= N ew_0 = F = I; \\
\text{while}(\text{size}(R) \leq \text{threshold}_1) & \\
& \quad \text{New}_i = \text{computeImageFwd}(F) - R; \\
& \quad \text{if}(\text{New}_i == \emptyset) \text{ break}; \\
& \quad \text{if}(\text{New}_i \cap T \neq \emptyset) \text{ return}(\text{fail}); \\
R &= R \cup \text{New}_i; \\
F &= \text{BddBetween}(\text{New}_i, R); \\
F &= \text{BddUnderApprox}(F); \\
\text{if}(\text{size}(F) > \text{threshold}_2) & \\
& \quad F = \text{ComputeCloseCube}(F, T); \\
\text{if}(R \cap T \neq \emptyset) & \text{ return}(\text{fail}); \\
B &= \text{computeImageFwd}(R) - R \\
\text{if}(B == \emptyset) & \text{ return}(\text{pass}); \\
\text{return}(\text{undecided});
\end{align*}
Example

$F$ is a BDD that represents a set of states
Example

We compute an under-approximation of $F$, $F^{-}$
We extract a cube $c$ in $F^{-}$ that is closest to the target states.
States 0, . . . , 9 are unreachable from the initial states

State 9 is a bad state
Reducing termination depth

- States 0, . . . , 9 are unreachable from the initial states
- State 9 is a bad state
- The longest simple path reaching state 9 is of length 9
Reducing termination depth

- States 0, ..., 9 are unreachable from the initial states
- State 9 is a bad state
- The longest simple path reaching state 9 is of length 9
- When we apply backward reachability for two steps, states 8, 3, and 7 are added to the target states
States 0, . . ., 9 are unreachable from the initial states

State 9 is a bad state

The longest simple path reaching state 9 is of length 9

When we apply backward reachability for two steps, states 8, 3, and 7 are added to the target states

As a result, the length of the longest simple paths to a target state that does not go through another target state decreases to 3
Reducing termination depth

States 0, ..., 9 are unreachable from the initial states.
State 9 is a bad state.
The longest simple path reaching state 9 is of length 9.
When we apply backward reachability for two steps, states 8, 3, and 7 are added to the target states.
As a result, the length of the longest simple paths to a target state that does not go through another target state decreases to 3.
Counterexample Generation

A counterexample can be constructed from

This is may not be the optimal counterexample
A counterexample can be constructed from:
- A path from \( s^n \in S_i \) to \( s^m \in S_t \) when \( S_i \cap S_t = \emptyset \)
A counterexample can be constructed from

- A path from $s^n \in S_i$ to $s^m \in S_t$ when $S_i \cap S_t = \emptyset$
- A path from $s^i \in I$ to $s^n$ when $s^n \not\in I$
A counterexample can be constructed from

- A path from \( s^n \in S_i \) to \( s^m \in S_t \) when \( S_i \cap S_t = \emptyset \)
- A path from \( s^i \in I \) to \( s^n \) when \( s^n \not\in I \)
- A path from \( s^m \) to \( s^t \in T \) when \( s^m \not\in T \)
Counterexample Generation

- A counterexample can be constructed from
  - A path from $s^n \in S_i$ to $s^m \in S_t$ when $S_i \cap S_t = \emptyset$
  - A path from $s^i \in I$ to $s^n$ when $s^n \not\in I$
  - A path from $s^m$ to $s^t \in T$ when $s^m \not\in T$
- This is may not be the optimal counterexample
Soundness and Completeness

Any path starting from \( I \) must go through the boundary states \( S_i \).

Any path leading to \( T \) must go through the boundary states \( S_t \).

Given enough resources (time and memory), our method will eventually find a counterexample or prove termination.
Soundness and Completeness

Any path starting from \( I \) must go through the boundary states \( S_i \).
Soundness and Completeness

- Any path starting from $I$ must go through the boundary states $S_i$.
- Any path leading to $T$ must go through the boundary states $S_t$.
Given enough resources (time and memory), our method will
Given enough resources (time and memory), our method will eventually find a counterexample or ...
Given enough resources (time and memory), our method will
- eventually find a counterexample or
- prove termination
Related work

- Baumgartner et al. [2002] use target enlargement to prove reachability
  - Use BDDs to represent the enlarged target states
  - Use SAT-based method to check for the reachability of the enlarged target states
  - Use composition-based pre-image computations to compute new target states that is limited by the size of BDD
  - The enlargement of the target states is bounded by an over-approximation of the diameter of the circuit
  - This requires a model to have special structure; hence, this bound is not always useful
Related work

- A work close to ours is that of Bischoff et al. [2004]
- In our work, we put more work in keeping BDD small
  - We extract states that are at minimum Hamming distance from the destination states
    - thus selecting states expected to be closer to the destination states
  - We compute boundary states
    - If the set of boundary states is empty, we could decide the property passes without using the SAT-based BMC
Related work

- Cabodi et al. [1994] describes a technique that combines over-approximated forward reachability and exact backward reachability.

- This technique is limited by the exact backward traversal step which may become very expensive.
Related work

- Cabodi et al. [1994] describes a technique that combines over-approximated forward reachability and exact backward reachability.
- This technique is limited by the exact backward traversal step which may become very expensive.

- BDD-based and SAT-based methods also combined in the context of abstraction refinement. E.g., Clarke et al. [2002], McMillan and Amla [2003]
  - Apply BDD-based method on the abstract model
  - Use SAT-based method to check if the counterexample is spurious or not
  - Use the proofs of unsatisfiability derived from the SAT-based method as a guide for refinement
Experimental results in VIS

We apply our algorithm $\texttt{bdd_sat}$ on safety properties $Gp$ and $G(p \rightarrow X q)$ where $p$ and $q$ are propositional.
Experimental results in VIS

- We apply our algorithm \textit{bdd_sat} on safety properties $G\, p$ and $G(p \rightarrow X\, q)$ where $p$ and $q$ are propositional.

- We compare the performance of \textit{bdd_sat} to
  - The SAT-based method BMC+termination \textit{bmc}
  - The BDD-based methods check\_invariant \textit{ci} and LTL model checking \textit{ltl}
Experimental results in VIS

- We apply our algorithm *bdd_sat* on safety properties $Gp$ and $G(p \rightarrow X q)$ where $p$ and $q$ are propositional.
- We compare the performance of *bdd_sat* to:
  - The SAT-based method BMC+termination *bmc*
  - The BDD-based methods check_invariant *ci* and LTL model checking *ltl*
- We performed two sets of experiments:
  - For invariant properties (*ci*)
  - For $G(p \rightarrow X q)$ properties (*ltl*)
Invariant properties

BDD+SAT: time (s)

BMC: time (s)

Check Invariant: time (s)
Invariant properties

Check 59 properties

<table>
<thead>
<tr>
<th></th>
<th>bdd_sat</th>
<th>outperforms</th>
<th>ties</th>
<th>loses</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmc</td>
<td>30</td>
<td>9</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>ci</td>
<td>45</td>
<td>5</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

awedh – p. 90
Invariant properties

![Graphs showing time (s) for BDD+SAT, BMC, and Check Invariant methods with data points and trend lines.](image)

- *ci* times out in 23 experiments (timeout = 1800s)
Invariant properties

Our method performs better for the passing properties than for the failing properties
Discussions

- Our method could prove properties that are hard for both the BDD-based and the SAT-based methods.
  - \textbf{bdd\_sat} fails to decide 5 out of 43 invariant properties, whereas \textbf{bmc} fails to decide 11 and \textbf{ci} fails to decide 24.
  - \textbf{bdd\_sat} decides all the 14 properties of the form $G(p \rightarrow X q)$, whereas \textbf{bmc} fails to decide 10 of them and \textbf{ltl} fails to decide 3 of them.
Discussions

- Our method could prove properties that are hard for both the BDD-based and the SAT-based methods.
  - `bdd_sat` fails to decide 5 out of 43 invariant properties, whereas `bmc` fails to decide 11 and `ci` fails to decide 24.
  - `bdd_sat` decides all the 14 properties of the form $G(p \rightarrow X q)$, whereas `bmc` fails to decide 10 of them and `ltl` fails to decide 3 of them.

Examples

<table>
<thead>
<tr>
<th>state vars</th>
<th>bdd_sat</th>
<th>bmc</th>
<th>BDD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>img</td>
<td>pre</td>
<td>st</td>
</tr>
<tr>
<td>37</td>
<td>6</td>
<td>4</td>
<td>F</td>
</tr>
<tr>
<td>215</td>
<td>2</td>
<td>2</td>
<td>P</td>
</tr>
</tbody>
</table>

`a`Time Out
Discussions

- Our method reduces the lengths of the paths to be examined by the SAT solver to find a counterexample or to prove termination.

Examples

<table>
<thead>
<tr>
<th>state vars</th>
<th>bdd_sat</th>
<th>bmc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>img</td>
<td>pre</td>
</tr>
<tr>
<td>115</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>238</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>85</td>
<td>26</td>
<td>2</td>
</tr>
</tbody>
</table>
**Discussions**

- Our method explores longer paths than *BMC*

<table>
<thead>
<tr>
<th>state vars</th>
<th>bdd_sat</th>
<th>bmc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>img</td>
<td>pre</td>
</tr>
<tr>
<td>85</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>
Discussions

- Our method explores longer paths than *BMC*

<table>
<thead>
<tr>
<th>state vars</th>
<th>bdd_sat</th>
<th>bmc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>img</td>
<td>pre</td>
</tr>
<tr>
<td>85</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

- Our method can go much deeper than *BMC* within the same amount of memory

<table>
<thead>
<tr>
<th>state vars</th>
<th>bdd_sat</th>
<th>bmc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>img</td>
<td>pre</td>
</tr>
<tr>
<td>157</td>
<td>37</td>
<td>4</td>
</tr>
</tbody>
</table>

- If we run *bmc* for $k = 36$, the SAT solver ran out of memory
Limitations of our approach

- Our methods inherits some of the drawbacks of both BDD-based and SAT-based methods
  - Sometimes BDD-based produces a very large BDDs, and hence SAT-instances become hard for SAT solvers
  - The size of BDD exceeds the threshold sooner
    - the amount of reduction in the unrolling of the transition relation is minor
    - increases the efforts on SAT solvers
- Sometimes backward reachability is much cheaper than forward reachability, so it is more efficient to apply backward reachability first
  - Alternate between forward and backward reachability analysis
## Limitations of our approach

<table>
<thead>
<tr>
<th></th>
<th>bdd_sat</th>
<th></th>
<th></th>
<th></th>
<th>bmc</th>
<th></th>
<th></th>
<th></th>
<th>BDD</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>img</td>
<td>pre</td>
<td>st</td>
<td>k</td>
<td>t (s)</td>
<td></td>
<td>st</td>
<td>k</td>
<td>t (s)</td>
<td></td>
<td>st</td>
<td>t (s)</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td></td>
<td></td>
<td><strong>TO</strong>(^a)</td>
<td>U</td>
<td>20</td>
<td>90.7</td>
<td></td>
<td>P</td>
<td>238.6</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>F</td>
<td>28</td>
<td>777.46</td>
<td>F</td>
<td>36</td>
<td>88.57</td>
<td></td>
<td>F</td>
<td>8.5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>F</td>
<td>0</td>
<td>312.84</td>
<td>F</td>
<td>5</td>
<td>36.17</td>
<td></td>
<td>F</td>
<td>1620.3</td>
<td></td>
</tr>
</tbody>
</table>

\(^a\)Time Out
Summary

- A new symbolic model checking algorithm that integrates BDD-based and SAT-based methods to prove or disprove reachability.
  - BDD-based provides a cheap lower bounds on the enlarged source and target states.
  - SAT-based proves or disproves the connection between the two enlarged sets.

- Prove properties that are hard for both methods.
- Reduces the amount of unrolling of the transition relation when using BMC.
- Allows BMC to explore longer paths.
- Helps detect termination sooner.
Summary

- A new symbolic model checking algorithm that integrates BDD-based and SAT-based methods to prove or disprove reachability
  - BDD-based provides a cheap lower bounds on the enlarged source and target states
  - SAT-based proves or disproves the connection between the two enlarged sets
- Prove properties that are hard for both methods
Summary

- A new symbolic model checking algorithm that integrates BDD-based and SAT-based methods to prove or disprove reachability
  - BDD-based provides a cheap lower bounds on the enlarged source and target states
  - SAT-based proves or disproves the connection between the two enlarged sets
- Prove properties that are hard for both methods
- Reduces the amount of unrolling of the transition relation when using BMC
Summary

- A new symbolic model checking algorithm that integrates BDD-based and SAT-based methods to prove or disprove reachability
  - BDD-based provides a cheap lower bounds on the enlarged source and target states
  - SAT-based proves or disproves the connection between the two enlarged sets
- Prove properties that are hard for both methods
- Reduces the amount of unrolling of the transition relation when using BMC
- Allows BMC to explore longer paths
Summary

- A new symbolic model checking algorithm that integrates BDD-based and SAT-based methods to prove or disprove reachability
  - BDD-based provides a cheap lower bounds on the enlarged source and target states
  - SAT-based proves or disproves the connection between the two enlarged sets
- Prove properties that are hard for both methods
- Reduces the amount of unrolling of the transition relation when using BMC
- Allows BMC to explore longer paths
- Helps detect termination sooner
Future Work: Interleaving

Search for a counterexample of length 5 starting at $s_0$
Future Work: Interleaving

- Search for a counterexample of length 5 starting at $s_0$
- SAT solver will try all paths from $s_0$ to $s_{10}$
Future Work: Interleaving

- Search for a counterexample of length 5 starting at $s_0$
- SAT solver will try all paths from $s_0$ to $s_{10}$
Future Work: Interleaving

- Search for a counterexample of length 5 starting at $s_0$
- SAT solver will try all paths from $s_0$ to $s_{10}$
Future Work: Interleaving

- Search for a counterexample of length 5 starting at $s_0$
- SAT solver will try all paths from $s_0$ to $s_{10}$
Future Work: Interleaving

- Search for a counterexample of length 5 starting at $s_0$
- SAT solver will try all paths from $s_0$ to $s_{10}$
Future Work: Interleaving

- Search for a counterexample of length 5 starting at $s_0$
- SAT solver will try all paths from $s_0$ to $s_{10}$
Future Work: Interleaving

What if SAT solver keeps track of visited states
Future Work: Interleaving

What if SAT solver keeps track of visited states
Future Work: Interleaving

What if SAT solver keeps track of visited states
Future Work: Interleaving

- What if SAT solver keeps track of visited states
Future Work: Interleaving

What if SAT solver keeps track of visited states

SAT solver will only search two paths
Future work

- Apply the application of the invariant strengthening technique to checking general linear-time temporal properties according to our approach
- Use SAT-based method to improve BDD-based method
  - Use SAT solver to get an over-approximation of the reachable states. This information could be used to reduce the size of BDDs in the BDD-based method
  - Analyze the proof of unsatisfiability produced by the SAT solver to extract hints for BDD-based guided search, or variable orders for the BDDs
Future work

- Use an incremental SAT solver to improve the performance of our termination criteria.
- Use Hybrid SAT solver (e.g., CirCUs).
- Study the use of Quantified Boolean Formulae (QBF) in finding the bound in our termination based on shortest paths.
- Apply the techniques presented to Software Model Checking.
Discussion