LTL model checking using Generalized Testing Automata

Ala Eddine BEN SALEM

LRDE/LIP6

12 October 2012
1. Automata-Theoretic Approach to Model Checking
2. Automata-Theoretic Approach to Model Checking
3. Comparison of three approaches
   - TGBA: Transition-based Generalized Büchi Automata
   - BA: Büchi Automata
   - TA: Testing Automata (only stuttering-insensitive languages)
4. The problem of the second pass in TA approach
5. New automata to avoid the second pass
   - Single-pass Testing Automata (STA)
   - TGTA: Transition-based Generalized Testing Automata
Automata-Theoretic Approach to Model Checking

Model $M$

LTL formula $\varphi$

$M \models \varphi$

or

counter-example
Automata-Theoretic Approach to Model Checking

Model $M$  
On the fly State-space generation  
State-space automaton $A_M$  

LTL formula $\varphi$  
LTL to $\omega$-automaton translation  
Negated formula automaton $A_{\neg \varphi}$

$M \models \varphi$ or counter-example
Automata-Theoretic Approach to Model Checking

Model $M$

On the fly State-space generation

State-space automaton $A_M$

LTL formula $\varphi$

LTL to $\omega$-automaton translation

Negated formula automaton $A_{\neg \varphi}$

Synchronized product $\mathcal{L}(A_M \otimes A_{\neg \varphi}) = \mathcal{L}(A_M) \cap \mathcal{L}(A_{\neg \varphi})$

Product Automaton $A_M \otimes A_{\neg \varphi}$

$M \models \varphi$ or conter-example
Automata-Theoretic Approach to Model Checking

- Model $M$
- LTL formula $\varphi$
- On the fly State-space generation
- LTL to $\omega$-automaton translation
- State-space automaton $A_M$
- Negated formula automaton $A_{\neg \varphi}$
- Synchronized product $L(A_M \otimes A_{\neg \varphi}) = L(A_M) \cap L(A_{\neg \varphi})$
- Product Automaton $A_M \otimes A_{\neg \varphi}$
- Emptiness check $L(A_M \otimes A_{\neg \varphi}) \not= \emptyset$
- $M \models \varphi$ or counter-example
Automata-Theoretic Approach to Model Checking

There are different types of Automata:
- **TGBA**: Transition-based Generalized Büchi Automata
- **BA**: Büchi Automata
- **TA**: Testing Automata (stuttering-insensitive)

Model $M$

State-space generation

State-space automaton $A_M$

LTL formula $\varphi$

LTL to $\omega$-automaton translation

Automaton $A_{\neg \varphi}$

Synchronized product

$$\mathcal{L}(A_M \otimes A_{\neg \varphi}) = \mathcal{L}(A_M) \cap \mathcal{L}(A_{\neg \varphi})$$

Product Automaton $A_M \otimes A_{\neg \varphi}$

Emptiness check

$$\mathcal{L}(A_M \otimes A_{\neg \varphi}) \not= \emptyset$$

Model $M \models \varphi$ or counter-example
Approach 1: TGBA (Transition-based Generalized Büchi Automata)

TGBA for the LTL property $\varphi = GF a \land GF b$ (Weak-fairness)

Let $AP$ = the set of atomic proposition.
A TGBA over the alphabet $K = 2^{AP}$ is a tuple $\langle S, I, R, F \rangle$:
- $S$ is finite set of states,
- $I \subseteq S$ is the set of initial states,
- $F$ is a finite set of acceptance conditions,
- $R \subseteq S \times 2^K \times 2^F \times S$ is the transition relation.

An infinite run of a TGBA is accepting if it visits each accepting condition from $F$ ($\bullet$, $\circ$, $\ldots$) infinitely often.
Approach 2: BA (Büchi Automata)

BA recognizing LTL property $\varphi = GF a \land GF b$

Has only one acceptance condition that is state-based.

A BA over the alphabet $K = 2^{AP}$ is a tuple $\langle S, I, R, F \rangle$:

- $F \subseteq S$ is a finite set of accepting states
- $R \subseteq S \times 2^K \times S$ is the transition relation

An infinite run of a BA is accepting if it visits at least one accepting state infinitely often.
Approach 3: TA (Testing Automata) / stuttering-insensitive

TA recognizing LTL property $F G p$

Model Execution = $\bar{p} \bar{p} p p \bar{p} p p p \ldots$

TA Run = $0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ \ldots$

Stuttering transition $\equiv$ transition $\emptyset$

- Each transition $(s, k, d)$ is labeled by a **change set** $k = \text{the set of atomic propositions that change}$ between $s$ and $d$. If $s \neq d$ then $k \neq \emptyset$

- Two kinds of accepting states:
  - $F \subseteq S$ is a set of Büchi-accepting states,
  - $G \subseteq S$ is a set of livelock-accepting states.

- A second way to accept an infinite run: reaches a livelock-accepting state and from that point only stuttering.
Preliminary work: Experimental comparison of the three approaches

Hypothesis: LTL\(\setminus X\) formulas (*stuttering-insensitive*)

Experimental evaluation comparing the three approaches: TGBA, BA and TA.

Results [Ben Salem 2011]:

- Verified properties (complete exploration of the product):
  - TA requires **two-pass emptiness check**
  - It is therefore better to use the **TGBA** approach.
- Violated properties (partial exploration of the product):
  - **TA** approach is the most efficient to detect counterexample
  - TGBA is more efficient than BA in all cases
Why does TA emptiness check require two passes?

- Two kinds of accepting SCC: Büchi-accepting or livelock-accepting: composed by stuttering-transitions $\emptyset$
- First pass may miss to detect livelock-accepting SCCs (depending on order to explore the transitions of (3, 1))

![Diagram](image)

Product between a model and a TA of ($F G p$). The red SCC is livelock-accepting.

Problem: mixing of non-stuttering and stuttering transitions in the same SCC (which contains livelock-accepting states)
New automata to avoid the second pass

1. **Single-pass Testing Automata (STA):**
   - a transformation of TA that never requires a second pass
   - add an artificial livelock state (that captures all livelock runs during the first pass)

2. **Transition-based Generalized Testing Automata (TGTA):**
   - new automaton that combines benefits from TA and TGBA
   - no two-pass emptiness check (unlike TA)
   - no artificial state added (unlike STA)
We transform a TA into a STA by:

- adding a unique livelock-accepting state \( g \) and
- adding a transition \((s, k, g)\) for any transition \((s, k, s')\) that goes into a livelock-accepting state \(s'\) in TA

Transformation of TA \((F G p)\) into STA
Single-pass Testing Automata (STA)

We transform a TA into a STA by:

- adding a unique livelock-accepting state $g$ and
- adding a transition $(s, k, g)$ for any transition $(s, k, s')$ that goes into a livelock-accepting state $s'$ in TA

Impact of STA on the product: single-pass emptiness check
During the TA to STA transformation:

- don’t add transition \((s, k, g)\) for transition \((s, k, s')\) where \(s'\) is both livelock and Büchi accepting,
- because in the product, any SCC containing \(s'\) is accepting

Transformation of TA recognizing \((a U G b)\) into optimized STA. The state 4 is both livelock and Büchi accepting.
TGTA: new automaton that combines ideas from TGBA and TA:

- **From TGBA:**
  - Transition-based generalized acceptance conditions.
  - A one-pass emptiness-check (the same algorithm)
- **From TA:**
  - Labeling transitions with change sets.
  - **Reduction of transitions \( \emptyset \) (but without adding livelock)

TGTA of \( (a \cup G b) \):

```
TGTA for a \cup G b, with F = \{0\}.
```

```
1 —> 2
2 —> 4
4 —> 4
```

```
1: 
   - a\bar{b}
   - b
   - \{b\}
   - \{a, b\}

2: 
   - ab
   - \{a\}
   - \{a, b\}

3: 
   - \bar{a}b
   - \{a\}
   - \{b\}

4: 
   - ab, \bar{a}b

```

Ala Eddine BEN SALEM

LTL model checking using Generalized Testing Automata
TGTA reduction does not add livelock-accepting states (unlike a TA reduction).

Reduction of stuttering-transitions in TA.
TGTA reduction does not add livelock-accepting states (unlike a TA reduction).

Reduction of stuttering-transitions in TA.

Reduction of stuttering-transitions in TGTA.
Experimental evaluation of TGTA against TGBA

Number of transitions explored by the emptiness check of TGTA against TGBA. Axes in logarithmic scale

- Verified properties (green crosses): TGTA is more efficient
- Violated properties (black circles): harder to interpret
Experimental evaluation of TGTA against TA

Number of transitions explored by the emptiness check of TGTA against TA. (Axes in logarithmic scale)

- **Verified properties:** TGTA more efficient, because TA requires two-pass
- **Violated properties:** same problem as for TGTA against TGBA
Conclusion

- We improved the model checking of stuttering-insensitive properties
- with some contributions: enhancing TA emptiness check, proposing STA and TGTA
- Our benchmarks show that TGTA outperform TA and TGBA

We plan additional work to:
- enable symbolic model checking with TGTA
- provide direct conversion of LTL to TGTA
- combine partial order reduction with TGTA

Questions