LTL model checking of stuttering-insensitive properties

Ala Eddine BEN SALEM

LRDE/LIP6

04 July 2012
\[ G(p \rightarrow Fq) \]

finite-state model

Model Checker

 população

 counterexample
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 $A_M \otimes A_{\neg \varphi}$

Emptiness check

\[
\mathcal{L}(A_M \otimes A_{\neg \varphi}) \not\equiv \emptyset
\]
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)

State-space generation

State-space automaton $A_M$

LTL formula $\varphi$

LTL to $\omega$-automaton translation

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

Synchronized product

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

Emptiness check

$\mathcal{L}(A_M \otimes A_{\neg \varphi}) \not\subseteq \emptyset$
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 $F$ ($\bullet$, $\bigcirc$, ...) infinitely often.
Approach 2: BA

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

Obtained from a TGBA by degeneralization

- 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 acceptance 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 acceptance state infinitely often.
Approach 3: TA (only stuttering-insensitive)

TA recognizing LTL property $F G p$

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

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

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

- $R \subseteq S \times K \times S$: each transition $(s, k, d)$ is labeled by the set of **atomic propositions that change** between $s$ and $d$,
- $F \subseteq S$ is a set of Büchi acceptance states,
- $G \subseteq S$ is a set of livelock acceptance states.

A second way to accept an infinite run: reaches a livelock acceptance 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
Contributions

1. Enhancing TA emptiness check to avoid a second pass when it is possible.

2. **Single-pass Testing Automata (STA):**
   - a transformation of TA that never requires a second pass
   - add an artificial livelock state (the only one)

3. **Transition-based Generalized Testing Automata (TGTA):**
   - new automaton that combines benefits from TA and TGBA
   - no two-pass emptiness check like TA
   - no artificial state like STA
Why does TA emptiness check require two passes?

- Two kinds of accepting SCC: Büchi or livelock (accepted if composed **only 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
We transform a TA into a STA by:

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

Transformation of TA ($F G p$) into STA.
We transform a TA into a STA by:

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

Impact of STA on the product: single-pass emptiness check
STA optimization

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 a new kind of automaton

TGTA combines ideas from TGBA and TA:

- From TGBA:
  - transition-based generalized acceptance conditions,
  - and a one-pass emptiness-check (the same algorithm)

- From TA:
  - reduction of stuttering-transitions
  - without adding livelock-acceptance (because two passes)

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

```
1: a\bar{b} → \{b\}
2: ab → \{b\}
3: \bar{a}b → \{a\}
4: ab → \{a\, b\}, \bar{a}b
```

```
LRDE 04 July 2012
```
TGTA reduction does not add livelock-accepting states (like a TA reduction).

Reduction of stuttering-transitions in TA.
Reduction of stuttering-transitions in TGTA

TGTA reduction does not add livelock-accepting states (like 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
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
- TGTA is our most significant contribution [Ben Salem 2012]
- 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