LTL model checking using Generalized Testing Automata

Ala Eddine BEN SALEM

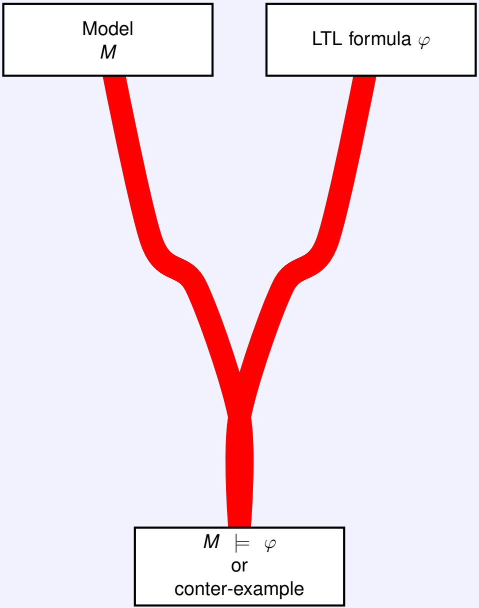
LRDE/LIP6

12 October 2012

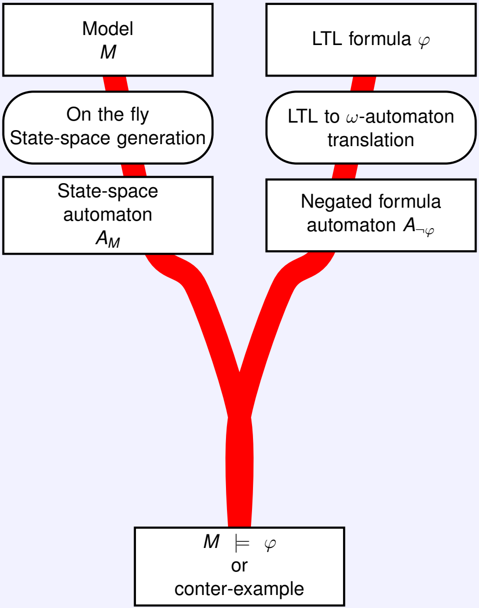
Outline

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

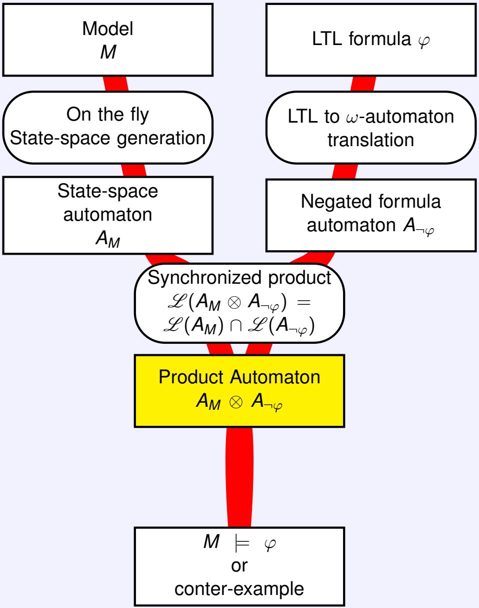
Automata-Theoretic Approach to Model Checking

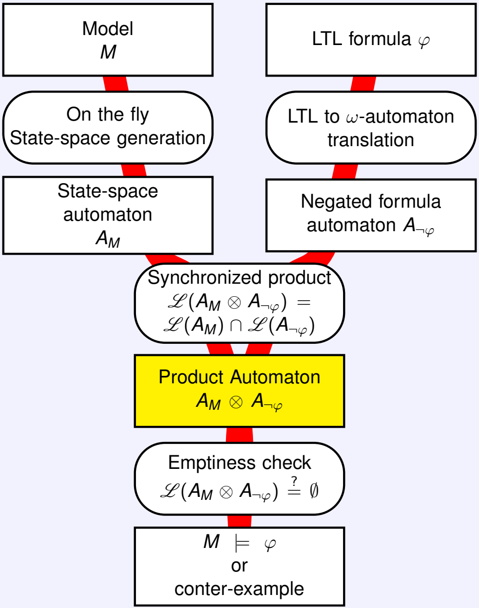


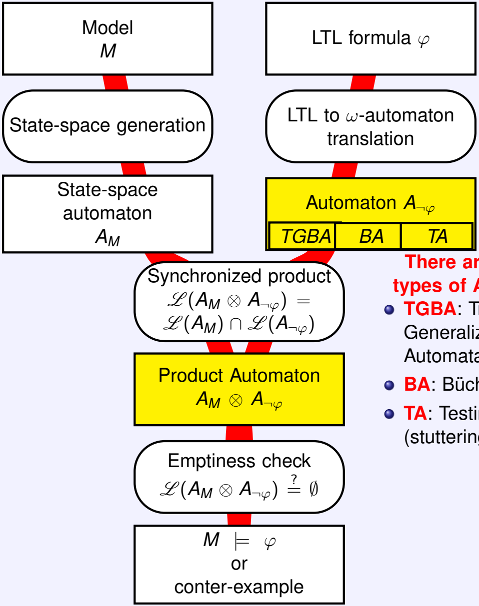
Automata-Theoretic Approach to Model Checking



Automata-Theoretic Approach to Model Checking




Model
M
LTL formula φ
On the fly
State-space generation
LTL to ω-automaton
translation
State-space
automaton
A_M
Negated formula
automaton $A_{\neg\varphi}$
Synchronized product
$\mathscr{L}(A_M \otimes A_{\neg\varphi}) = \mathscr{L}(A_M) \cap \mathscr{L}(A_{\neg\varphi})$
Product Automaton
$A_M \otimes A_{\neg\varphi}$
Emptiness check
$\mathscr{L}(A_M \otimes A_{\neg\varphi}) \stackrel{?}{=} \emptyset$
$M \models \varphi$
or
conter-example

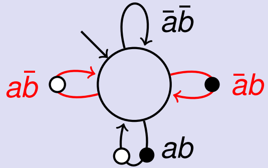


There are different types of Automata:

- **TGBA**: Transition-based Generalized Büchi Automata
- **BA**: Büchi Automata
- **TA**: Testing Automata (stuttering-insensitive)

Approach 1: TGBA (Transition-based Generalized Büchi Automata)

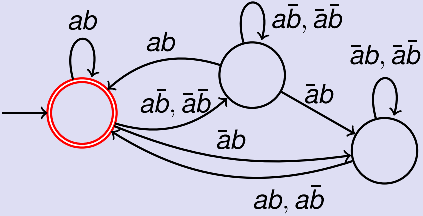
TGBA for the LTL property $\varphi = \mathsf{G}\,\mathsf{F}\,a \wedge \mathsf{G}\,\mathsf{F}\,b$ (Weak-fairness)



- 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 (●, ○, ...) infinitely often.

Approach 2: BA (Büchi Automata)

BA recognizing LTL property $\varphi = \mathsf{G}\,\mathsf{F}\,a \wedge \mathsf{G}\,\mathsf{F}\,b$



Obtained from a TGBA by degeneralization

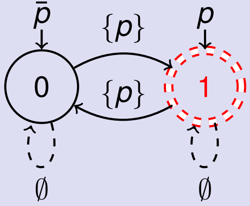
- Has only one acceptance condition that is state-based.
- A BA over the alphabet $K = 2^{AP}$ is a tuple $\langle S, I, R, F \rangle$:
  - $F \subseteq S$ **is a finite set of accepting states**
  - $R \subseteq S \times 2^K \times S$ is the transition relation
- An infinite run of a BA is accepting if it visits at least one accepting state infinitely often.

TA recognizing LTL property F G p

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

TA Run = 0 0 1 1 0 1 1 1 1 ...

Stuttering transition $\equiv$ transition $\emptyset$



- Each transition (s, k, d) is labeled by a **change set** k = 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*)

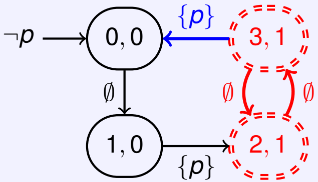
Experimental evaluation comparing the three approaches: TGBA, BA and TA.

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 ?

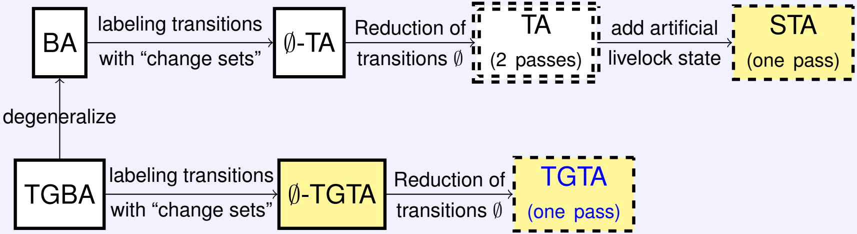
- Two kinds of accepting SCC: Büchi-accepting or livelock-accepting: composed **by stuttering-transitions** $\emptyset$
- first pass may miss to detect livelock-accepting SCCs (depending on order to explore the transitions of $(3, 1)$)



Product between a model and a TA of $(\mathsf{F}\,\mathsf{G}\,p)$. 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

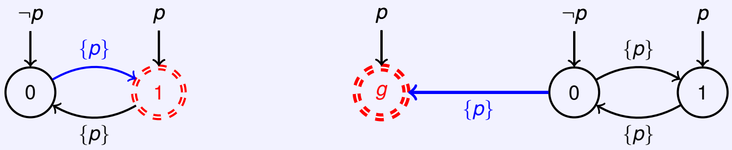


1. *Single-pass Testing Automata* (STA):
   - a transformation of TA that never requires a second pass
   - add an artificial livelock state (that captures all livelock runs during the first pass)
2. *Transition-based Generalized Testing Automata* (TGTA):
   - new automaton that combines benefits from TA and TGBA
   - no two-pass emptiness check (unlike TA)
   - no artificial state added (unlike STA)

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

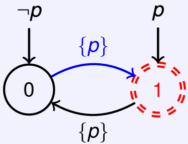


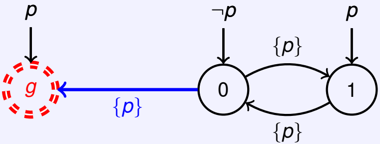
Transfomation of TA ($\mathsf{F}\,\mathsf{G}\,p$) into STA

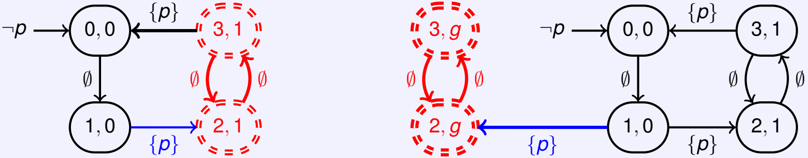
Single-pass Testing Automata (STA)

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





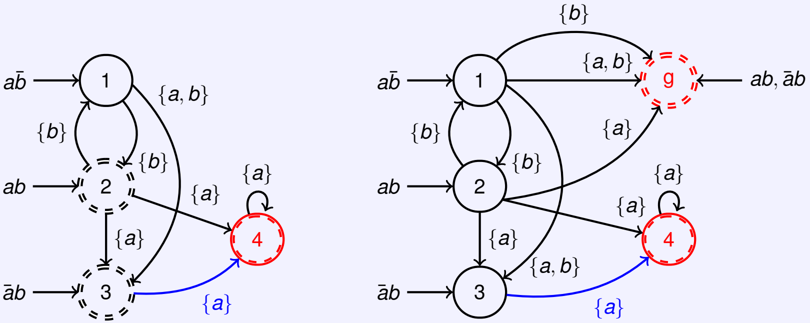


Impact of STA on the product: single-pass emptiness check

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



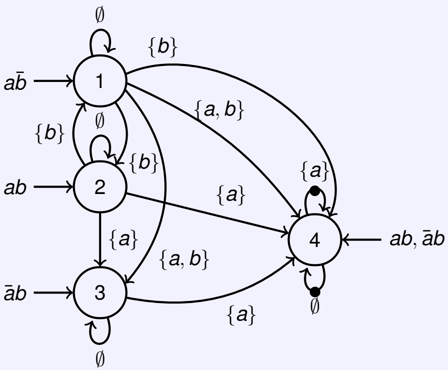
Transformation of TA recognizing $(a \, \mathrm{U} \, \mathrm{G} \, b)$ into optimized STA. The state 4 is both livelock and Büchi accepting

TGTA: Transition-based Generalized Testing Automata

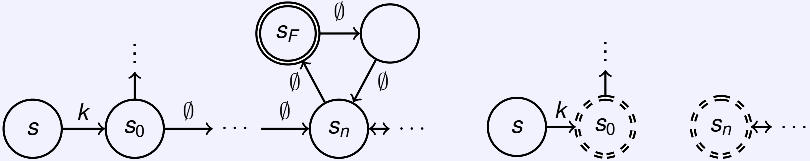
TGTA: new automaton that combines ideas from TGBA and TA:

- 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 $(a \, \mathsf{U} \, \mathsf{G} \, b)$:



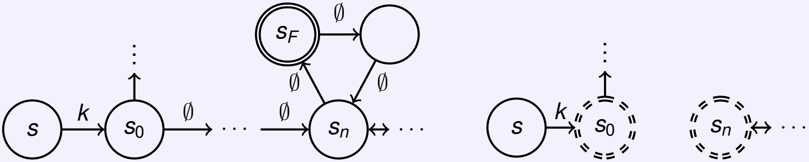
TGTA reduction does not add livelock-accepting states (unlike a TA reduction).



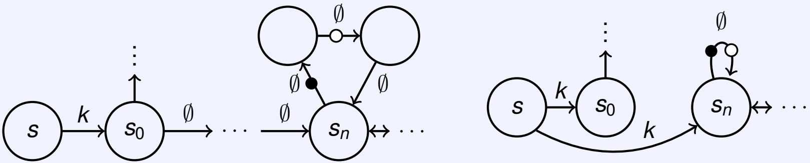
Reduction of stuttering-transitions in **TA**.

Reduction of stuttering-transitions in TGTA versus TA

TGTA reduction does not add livelock-accepting states (unlike a TA reduction).

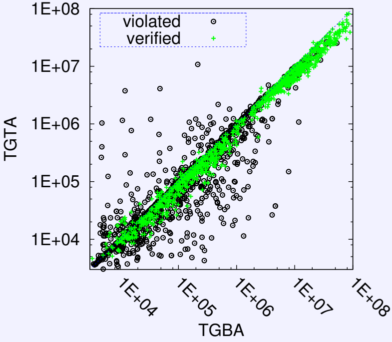


Reduction of stuttering-transitions in **TA**.



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

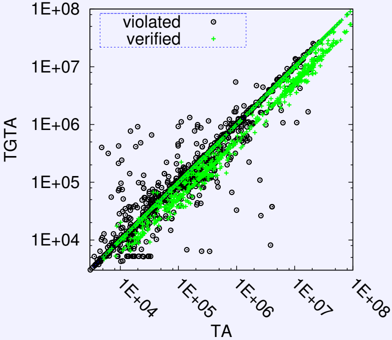
Experimental evaluation of TGTA against TGBA



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 cheking of stuttering-insensitive properties
- with some contributions: enhancing TA emtiness 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