Limit-Deterministic Büchi Automata for Probabilistic Model Checking

Javier Esparza    Jan Křetínský
Stefan Jaax    Salomon Sickert
Technische Universität München
PROBABILISTIC MODEL CHECKING

- Markov Decision Process (MDP).

  At each state, a scheduler chooses a probability distribution, and then the next state is chosen stochastically according to the distribution.

  Fixed scheduler: MDP → Markov chain

- Qualitative Model Checking:
  - Input: MDP, LTL formula
  - Does the formula hold for all schedulers with probability 1?

- Quantitative Model Checking:
  - Input: MDP, LTL formula, threshold \( c \)
  - Does the formula hold for all schedulers with probability at least \( c \)?
LIMIT-DETERMINISTIC BÜCHI AUTOMATA

Initial Component

(possibly) non-deterministic

Accepting Component
deterministic

“Jumps”
QUALITATIVE PROB. MODEL CHECKING

- MDP
- LTL
- Büchi
- Product
- Limit-det. Büchi

- Non-optimal: double exponential
- Other algorithms with single exponential complexity

Vardi [85]
Courcoubetis, and Yannakakis [88,95]

Prob=1?
Yes/No
QUANTITATIVE PROB. MODEL CHECKING

- In practice large automata
- Hard to implement efficiently
- Rise of “safraleza” approaches:
  - Acacia, ltl3dra, Rabinizer, ...

Yes/No

Asymp. optimal: double exponential

Safra [89] → Det. Rabin

MDP

LTL

Det. Büchi
Our Construction

MDP

• Optimal: $2^{O(n)}$
• Simpler construction
• Smaller automata
• Same MC algorithm as for det. automata

Product

P ≥ 0.7?
Yes/No

LTL

Limit-det. Büchi

Yes/No
LIMIT-DETERMINISM

In our construction:

Every runs „uses“ nondeterminism at most once
PRELIMINARIES

• Linear Temporal Logic in Negation Normal Form

\[ \varphi ::= \text{tt} | \text{ff} | a | \neg a | \varphi \land \varphi | \varphi \lor \varphi | F\varphi | \varphi U \varphi | X\varphi | G\varphi \]

Only liveness operator.

• Monotonicity of NNF:

    if \( w \) satisfies \( \varphi \)
    \( w' \) satisfies all the subformulas of \( \varphi \) satisfied by \( w \),
    and perhaps more

    then \( w' \) satisfies \( \varphi \)
FIRST STEP: A DETERMINISTIC "TRACKING" AUTOMATON

- The automaton "tracks" the property that must hold now for the original property to hold at the beginning.

- Formulas with $F, X, U$: ✔

- Formulas with $G$: not good enough.
$G$-SUBFORMULAS

- Fix a formula $\varphi$ and a word $w$. Let $G\psi$ be a $G$-subformula of $\varphi$.

- Informally: while reading the word $w$, the set of $G$-subformulas that hold cannot decrease, and eventually stabilizes to a set $\text{TrueGs}(w, \varphi)$. 
SECOND STEP: JUMPING

- We modify the tracking automaton so that at any moment it can nondeterministically jump to an accepting component.

- From each state $\psi$ we add a jump for every set $G$ of $G$-subformulas of $\psi$.

- „Meaning“ of a $G$-jump at state $\psi$: The automaton „guesses“ that the rest of the word satisfies
  
  1. $G$ (every formula of $G$), and

  2. $G \Rightarrow \psi$

  even if no other $G$-subformula of $\psi$ ever becomes true.

- After the jump, the task of the accepting component is to „check that the guess is correct“, i.e., accept iff the guess is correct.
SECOND STEP: JUMPING

• „Meaning“ of the $\mathcal{G}$-jump at state $\psi$: The automaton „guesses“ that the rest of the run satisfies

  1. $\mathcal{G}$ (every formula of $\mathcal{G}$), and
  2. $\mathcal{G} \Rightarrow \psi$

  even if no other $\mathcal{G}$-subformula of $\psi$ ever becomes true.

• $w \models \varphi$ iff the automaton can make a right guess.

  • Right guess before suffix $w' \rightarrow w' \models \psi \rightarrow w \models \varphi$ (tracking!)
  • $w \models \varphi \rightarrow w' \models \text{TrueGs}(w, \varphi)$ for some suffix $w'$
    $\rightarrow$ jump before $w'$ with $\mathcal{G} := \text{TrueGs}(w, \varphi)$ satisfies 1. and 2.
A DBA THAT CHECKS 1. & 2.

- Since DBA are closed under intersection, it suffices to construct two DBAs for 1. and 2.
CHECKING 2.

• “\(G \Rightarrow \psi\) holds even if no other \(G\)-subformula of \(\psi\) ever becomes true”

• Reduces to checking the \(G\)-free formula

\[
\psi[\ G \setminus \text{tt} \ , \ \overline{G} \setminus \text{ff} \ ]
\]

• Example:

\[
\psi = G(a \lor Fb) \land (Gc \lor Xd)
\]

\[
G = \{ G(a \lor Fb) \}
\]

reduces to checking \(Xd\)

• Since the formula is \(G\)-free, use the tracking automaton.
CHECKING 1.

- "\( g \) holds even if no other \( G \)-subformula of \( \psi \) ever becomes true"

- Reduces to checking a formula \( G\rho \) where \( \rho \) is \( G \)-free.

- Example:
  \[
  \psi = Fc \land GF(a \land (Gb \lor FGc))
  \]
  \[
  G = \{ Gb, GF(a \land (Gb \lor FGc)) \}
  \]
  reduces to checking \( Gb \land GFa \equiv G(b \land Fa) \)
We use the well-known breakpoint construction.
A DBA FOR $G(a \lor Fb)$

- Put new goals on hold while tracking current goal
- Accept if infinitely often the current goal is proven
- “Breakpoint Construction”
DBA FOR $G(a \lor Fb)$
COMPLETE LDBS FOR $\varphi = c \lor XG(a \lor Fb)$

1. Tracking DBA for $\varphi$
   (abbr. $\psi := a \lor Fb$)

2. For every set $\mathcal{G}$ add a $\mathcal{G}$-jump to the product of the automata checking $\mathcal{G}$ and the $\mathcal{G}$-remainder
<table>
<thead>
<tr>
<th>Part</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Component</td>
<td>$2^{2^n}$</td>
</tr>
<tr>
<td>G-Monitor</td>
<td>$2^{2^{n+1}}$</td>
</tr>
<tr>
<td>Accepting Component</td>
<td>$2^{2^{O(n)}}$</td>
</tr>
<tr>
<td>Total</td>
<td>$2^{2^{O(n)}}$</td>
</tr>
</tbody>
</table>
**SIZES OF AUTOMATA**

\[ \bigwedge_{i=1}^{j}(GFa_i) \implies \bigwedge_{i=1}^{j}(GFb_i) \]

\[ k: \bigwedge_{i=1}^{k}(GFa_i \lor FGb_i) \]

\[ f(0, j) = (GFa_0)U(X^j b) \]

\[ f(i + 1, j) = (GFa_{i+1})U(Gf(i, j)) \]

<table>
<thead>
<tr>
<th></th>
<th>LDBA</th>
<th>Safra (spot+ltl2dstar)</th>
<th>Rabinizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j = 1 )</td>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>( j = 2 )</td>
<td>4</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>( j = 3 )</td>
<td>5</td>
<td>49</td>
<td>4</td>
</tr>
<tr>
<td>( j = 4 )</td>
<td>6</td>
<td>129</td>
<td>5</td>
</tr>
<tr>
<td>( k = 2 )</td>
<td></td>
<td>4385</td>
<td>13</td>
</tr>
<tr>
<td>( k = 3 )</td>
<td>9</td>
<td>*</td>
<td>198</td>
</tr>
<tr>
<td>( f(0, 0) )</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>( f(0, 2) )</td>
<td>10</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>( f(0, 4) )</td>
<td>16</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>( f(1, 0) )</td>
<td>6</td>
<td>196</td>
<td>17</td>
</tr>
<tr>
<td>( f(1, 2) )</td>
<td>28</td>
<td>109839</td>
<td>33</td>
</tr>
<tr>
<td>( f(1, 4) )</td>
<td>58</td>
<td>*</td>
<td>70</td>
</tr>
<tr>
<td>( f(2, 0) )</td>
<td>10</td>
<td>99793</td>
<td>41</td>
</tr>
<tr>
<td>( f(2, 2) )</td>
<td>46</td>
<td>*</td>
<td>94</td>
</tr>
<tr>
<td>( f(2, 4) )</td>
<td>92</td>
<td>*</td>
<td>139</td>
</tr>
</tbody>
</table>
### MODEL CHECKING RUNTIME

#### PNUELI-ZUCK MUTEX PROTOCOL

<table>
<thead>
<tr>
<th>Our Implementation</th>
<th>Symbolic, state-based</th>
<th>Rabinizer state-based</th>
<th>CMC state-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>explicit, transition-based</td>
<td>PRISM symbolic, state-based</td>
<td>PRISM+Rabinizer</td>
<td></td>
</tr>
</tbody>
</table>

#### #Clients

| Step | Formula | #Clients | PRISM | PRISM+Rabinizer | PRISM+Rabinizer
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(6)</td>
<td>$P_{max} = ?[ GFp1=0 \lor FGp2 \neq 0 \land (GFp2=0 \lor FGp3 \neq 0) \land (GFp3=0 \lor GFp1 \neq 0) ]$</td>
<td>4</td>
<td>&lt; 1</td>
<td>78</td>
<td>9</td>
</tr>
<tr>
<td>(7)</td>
<td>$P_{max} = ?[ GFp1=0 \land (GFp2=0 \lor FGp3 \neq 0) \land (GFp3=0 \lor GFp1 \neq 0) ]$</td>
<td>5</td>
<td>10</td>
<td>1293</td>
<td>137</td>
</tr>
<tr>
<td>(8)</td>
<td>$P_{min} = ?[ GFp1 \neq 0 \lor GFp2 \neq 0 \land GFp1=0 \land GFp2=1 ]$</td>
<td>4</td>
<td>&lt; 1</td>
<td>&lt; 1</td>
<td>61</td>
</tr>
<tr>
<td>(9)</td>
<td>$P_{min} = ?[ GFp1 \neq 0 \lor GFp2 \neq 0 \lor GFp3 \neq 0 \land GFp1 \neq 0 \land GFp2=1 \land GFp3=1 ]$</td>
<td>5</td>
<td>1</td>
<td>&lt; 1</td>
<td>1077</td>
</tr>
<tr>
<td>(10)</td>
<td>$P_{min} = ?[ GFp1 \neq 0 \lor GFp2 \neq 0 \land GFp3 \neq 0 \land GFp2=1 \land GFp3=1 ]$</td>
<td>4</td>
<td>5</td>
<td>-</td>
<td>1195</td>
</tr>
<tr>
<td>(11)</td>
<td>$P_{min} = ?[ GFp1=0 \lor GFp2 \neq 0 \land GFp3=0 \land GFp2=1 ]$</td>
<td>5</td>
<td>24</td>
<td>-</td>
<td>486</td>
</tr>
<tr>
<td>(12)</td>
<td>$P_{max} = ?[ f_{0,4} ]$</td>
<td>4</td>
<td>&lt; 1</td>
<td>728</td>
<td>33</td>
</tr>
<tr>
<td>(13)</td>
<td>$P_{min} = ?[ GFp1=10 \lor GFp2=10 ]$</td>
<td>5</td>
<td>11</td>
<td>17</td>
<td>40</td>
</tr>
<tr>
<td>(14)</td>
<td>$P_{max} = ?[ f_{0,4} ]$</td>
<td>5</td>
<td>5</td>
<td>20</td>
<td>&lt; 1</td>
</tr>
</tbody>
</table>

#Clients
CONCLUSION

• We have presented a translation from LTL to LDBA that
  • uses formulas as states
  • is modular
    • optimisations of any module helps to reduce state space!
  • yields in practice small $\omega$-automata
  • is usable for quantitative prob. model checking without changing the algorithm!

• Website: https://www7.in.tum.de/~sickert/projects/Ltl2Ldba/