SAT-Based Explicit LTL Reasoning

Jianwen Li\textsuperscript{1,2}  Shufang Zhu\textsuperscript{2}  \textbf{Geguang Pu}\textsuperscript{2}  Moshe Y. Vardi\textsuperscript{1}

1. Rice University

2. East China Normal University

August 22, 2016
Church, 1957: Given a model $M$ and MSO specification $\phi$, check $M \models \phi$? (Model-Checking Problem)

Pnueli, 1977: Linear Temporal Logic (LTL)

Pnueli-Lichtenstein, 1985: LTL model checking

V. and Wolper, 1986: Automata-theoretic model checking – LTL to Automata
Temporal-Reasoning Tasks

- LTL model checking
- LTL $\rightarrow$ Büchi automata: explicit or symbolic
- LTL $\rightarrow$ runtime monitors
- $LTL$ satisfiability checking
Debug specifications
- Properties and their negations should be satisfiable.
- Conjunction of properties should be satisfiable.

Efficient algorithms may be adaptable to model checking.
- LTL satisfiability is a special case of LTL model checking.
Explicit model checking

- **Gerth-Peled-V.-Wolper, 1995**: Tableau-based construction from LTL formulas to Büchi automata
- **Holzmann 1997**: First explicit model checker – *Spin*
- **Since 1997**: dozens of works on optimization of LTL-to-Büchi translation
- **Duret-Lutz&Poitrenaud, 2004**: Well-performing LTL-to-automata translator – *Spot*
LTL-Satisfiability Checking - History

- Rozier&V., 2007:
  - Reduction to model checking
  - BDD-based symbolic checking (SMV) outperformed explicit checking (Spot+Spin)
- Aalta, 2013: best LTL satisfiability solver – explicit checking
- NuXMV, 2015: SAT-based symbolic model checker outperforms Aalta
- Question: What is best for LTL satisfiability – explicit vs symbolic.
Motivation

- SAT techniques have been widely used in symbolic model checking.
- SAT techniques have not been used in explicit model checking.
- **Question**: Can explicit model checking utilize SAT techniques as well?
Explicit vs Symbolic in MC

Sebastiani, Tonetta, Vardi, CAV’05:

- “Symbolic Systems, Explicit Properties: On Hybrid Approaches for LTL Symbolic Model Checking”
- Hybrid approach dominates symbolic approach.
Linear Temporal Logic (LTL)

\[ \phi ::= true \mid false \mid a \mid \neg \phi \mid \phi \land \phi \mid \phi \lor \phi \mid \phi U \phi \mid X \phi \]

Assume LTL formulas are in NNF (Negation Normal Form)

- \( X \psi \): \( \psi \) must hold in next step
- \( \psi_1 U \psi_2 \): \( \psi_2 \) will eventually hold, and before that \( \psi_1 \) must always hold.
- \( \psi_1 R \psi_2 \): \( \psi_2 \) holds until “released” by \( \psi_1 \)
- LTL formulas are interpreted over infinite traces
LTL explicit model checking

Given model $M$ a specification $\phi$

1. consider $M$ as automaton with no accepting condition.
2. Translate $\neg \phi$ its equivalent Büchi automaton $A_{\neg \phi}$.
3. Check nonemptiness of $M \times A_{\neg \phi}$ – if a witness trace $\tau$ is found then $M \models \phi$ fails and $\tau$ is counterexample.
4. If $M$ is universal (allowing all traces), then model checking $\neg \psi$ checks satisfiability of $\psi$. 
Aalta’s basic Algorithm

- Generate automaton on the fly
- Use DFS search to find a satisfying model as soon as possible
- Sophisticated heuristics speed up search
General idea: *syntactic splitting*

Consider $\phi$ to be a state:

1. Start from $\phi$
2. $\phi \leftrightarrow \bigvee_i (\alpha_i \land X\psi_i)$: $(\phi, \alpha_i, \psi_i)$ is a transition in the automaton.
   - For Until/Release formula: $\psi_1 U \psi_2 \equiv (\psi_2 \lor (\psi_1 \land X(\psi_1 U \psi_2)))$ and $\psi_1 R \psi_2 \equiv (\psi_2 \land (\psi_1 \lor X(\psi_1 R \psi_2)))$.
3. For each new state $\psi_i$, repeat from step 2 until no new states are generated.
Automata Generation in Aalta

- $aUb$

![Büchi automaton for $aUb$]

**Figure:** The Büchi automaton for $aUb$

$$aUb = (b \land X\text{True}) \lor (a \land X(aUb))$$
Bottleneck in Aalta

- Transformation $\phi \equiv \bigvee_i (\alpha_i \land X\psi_i)$ may be very expensive
- Exponential delay before we start generating states
- **Consequence**: even short trace may be very expensive to generate
This Work

- From the current state, do not start by generating all next states.
- Rather, generate states *on the fly*
- **Key:** Use SAT to generate states on the fly.
Definition 1 (neXt Normal Form)

An LTL formula $\phi$ is in neXt Normal Form (XNF) if all Until/Release formulas are preceded by Next.

For example,

- $(b \lor (a \land (X(aUb))))$ is in XNF.
- $a \land (b \lor cUa)$ is not in XNF.
neXt Normal Form (XNF)

Theorem 1

For an LTL formula $\phi$, there is an equivalent formula $\text{xnf}(\phi)$ that is in XNF. Furthermore, the cost of the conversion is polynomial.

Proof.

1. $\text{xnf}(\phi) = \phi$ if $\phi$ is true, false, a literal $l$ or a Next formula $X\psi$;
2. $\text{xnf}(\phi) = \text{xnf}(\phi_1) \land \text{xnf}(\phi_2)$ if $\phi = (\phi_1 \land \phi_2)$;
3. $\text{xnf}(\phi) = \text{xnf}(\phi_1) \lor \text{xnf}(\phi_2)$ if $\phi = (\phi_1 \lor \phi_2)$;
4. $\text{xnf}(\phi) = (\text{xnf}(\phi_2)) \lor (\text{xnf}(\phi_1) \land X\phi)$ if $\phi = (\phi_1 U\phi_2)$;
5. $\text{xnf}(\phi) = \text{xnf}(\phi_2) \land (\text{xnf}(\phi_1) \lor X\phi)$ if $\phi = (\phi_1 R\phi_2)$.  

For an LTL formula \( \phi \) in XNF, consider each Next subformula as an “atom”, then we can treat \( \phi \) as a propositional formula, denoted as \( \phi^p \).

- \( \phi = (b \lor (a \land (X(aUb)))) \Rightarrow \phi^p = b \lor (a \land \text{newVar}) \), where \( \text{newVar} = X(aUb) \).
- \( \phi = Xa \lor (b \land X(cUb)) \Rightarrow \phi^p = \text{newVar}_1 \lor (b \land \text{newVar}_2) \), where \( \text{newVar}_1 = Xa \) and \( \text{newVar}_2 = X(cUb) \)
Generate states via SAT solver

Given an LTL formula $\phi$,

- Take $\text{xnf}(\phi)^P$ as input for SAT solver
- A satisfying assignment describes current state and a successor state
- Let $A$ be an assignment, then $A = L \cup X(A) \cup \neg X(A)$, and $(\phi, \land L, \land \psi_i)((X\psi_i) \in X(A))$ is a transition.
  - $L$ is the set of literals in $A$.
  - $X(A)$ is the set of Next formulas in $A$.
  - $\neg X(A)$ is the set of negative Next formulas in $A$, and is ignored, as formulas are in NNF.
Generate states via SAT solver

Consider $\phi = (a \cup b) \land (c \cup \neg b)$.

$$xnf(\phi) = (b \lor (a \land X(a \cup b))) \land (\neg b \lor (c \land X(c \cup \neg b)))$$

SAT solver may give us an assignment of
{ $a, \neg b, c, X(a \cup b), \neg X(c \cup \neg b)$ }

Assignment indicates $(\phi, a \land \neg b \land c, (a \cup b))$ is a transition.
Advantages of Approach

- We go from *syntactic splitting* to *semantic splitting*, leveraging power of SAT solvers.
- Generate states on-the-fly.
- Search can be guided by adding constraints to formulas submitted to SAT solver.
Syntactic vs. Semantic Splitting: an Old Debate

- Beth, 1955: propositional tableaux – syntactic splitting
- Roth, 1966: ATPG – syntactic splitting

**Final Verdict**: semantic splitting wins!

V., 1989: modal and temporal satisfiability can be based on top of propositional SAT solving.
Searching for a Satisfying Trace

- A DFS lasso search is necessary to find a satisfying trace
- All states may have to be explored for unsatisfiable cases
- Heuristics are used to speed up search in both satisfiable and unsatisfiable cases
Table: Experimental results on the Schuppan-collected benchmarks. Each cell lists a tuple \(\langle t, n \rangle\) where \(t\) is the total checking time (in seconds), and \(n\) is the total number of unsolved formulas.

<table>
<thead>
<tr>
<th>Formula type</th>
<th>ls4</th>
<th>TRP++</th>
<th>NuXmv-BMCINC</th>
<th>Aalta_v1.2</th>
<th>NuXmv-IC3-Klive</th>
<th>Aalta_v2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>/acacia/example</td>
<td>155</td>
<td>0</td>
<td>192</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>/acacia/demo-v3</td>
<td>68</td>
<td>0</td>
<td>2834</td>
<td>38</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>/acacia/demo-v22</td>
<td>60</td>
<td>0</td>
<td>67</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>/alaska/lift</td>
<td>2381</td>
<td>27</td>
<td>15602</td>
<td>254</td>
<td>1919</td>
<td>26</td>
</tr>
<tr>
<td>/alaska/szymanski</td>
<td>27</td>
<td>0</td>
<td>283</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>/anzu/amba</td>
<td>5820</td>
<td>92</td>
<td>6120</td>
<td>102</td>
<td>536</td>
<td>7</td>
</tr>
<tr>
<td>/anzu/genbuf</td>
<td>2200</td>
<td>30</td>
<td>7200</td>
<td>120</td>
<td>782</td>
<td>11</td>
</tr>
<tr>
<td>/rozier/counter</td>
<td>3934</td>
<td>62</td>
<td>4491</td>
<td>44</td>
<td>3865</td>
<td>64</td>
</tr>
<tr>
<td>/rozier/formulas</td>
<td>167</td>
<td>0</td>
<td>37533</td>
<td>523</td>
<td>1258</td>
<td>19</td>
</tr>
<tr>
<td>/rozier/pattern</td>
<td>2216</td>
<td>38</td>
<td>15450</td>
<td>237</td>
<td>1505</td>
<td>8</td>
</tr>
<tr>
<td>/schuppan/O1formula</td>
<td>2193</td>
<td>34</td>
<td>2178</td>
<td>35</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>/schuppan/O2formula</td>
<td>2284</td>
<td>35</td>
<td>2566</td>
<td>41</td>
<td>1781</td>
<td>28</td>
</tr>
<tr>
<td>/schuppan/phtlt</td>
<td>1771</td>
<td>27</td>
<td>1793</td>
<td>29</td>
<td>1058</td>
<td>15</td>
</tr>
<tr>
<td>/trp/N5x</td>
<td>144</td>
<td>0</td>
<td>46</td>
<td>0</td>
<td>567</td>
<td>9</td>
</tr>
<tr>
<td>/trp/N5y</td>
<td>448</td>
<td>10</td>
<td>95</td>
<td>1</td>
<td>2768</td>
<td>46</td>
</tr>
<tr>
<td>/trp/N12x</td>
<td>3345</td>
<td>52</td>
<td>45739</td>
<td>735</td>
<td>3570</td>
<td>58</td>
</tr>
<tr>
<td>/trp/N12y</td>
<td>3811</td>
<td>56</td>
<td>19142</td>
<td>265</td>
<td>4049</td>
<td>67</td>
</tr>
<tr>
<td>/forobots</td>
<td>990</td>
<td>0</td>
<td>1303</td>
<td>0</td>
<td>1085</td>
<td>18</td>
</tr>
<tr>
<td>Total</td>
<td>32014</td>
<td>463</td>
<td>163142</td>
<td>2428</td>
<td>24769</td>
<td>376</td>
</tr>
</tbody>
</table>

Jianwen Li, Shufang Zhu, Geguang Pu, Moshe Y. Vardi (Rice University)
Application to LTL satisfiability checking

- Total formulas checked: 7448
- IC3-Klive is more than twice as fast as Aalta_1.2
- Aalta_2.0 is almost twice as fast as IC3-Klive
- No other approach is competitive
- *Truth in Advertising*: IC3-Klive is faster on unsatisfiable formulas.
Experiments on Random-Conjunction Formulas

- For property-based design, need also to check that conjunction of temporal properties is satisfiable.
- \( RC(n) = \bigwedge_{1 \leq i \leq n} P_i \)
- \( P_i \): randomly chosen \textit{specification-pattern formulas}\(^1\) (3000 random-conjunction formulas tested)

---

\(^1\)http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml
Figure: Results for LTL-satisfiability checking on random-conjunction formulas.
By replacing SAT solver with SMT solver, we can also handle *assertional LTL*.

Consider the formula $\phi = (F(k = 1) \land F(k = 2))$.

If we use a SAT solver, we can obtain an assignment such as $A = \{(k = 1), (k = 2)\}$, which is consistent propositionally, but inconsistent theory-wise.
**Figure:** Results for LTL-satisfiability checking on $\bigwedge_{1 \leq i \leq n} F(k = i)$. 
In Conclusion

- We proposed a *SAT-based explicit LTL reasoning* framework.
- We applied to LTL-satisfiability checking, and got a *best-of-breed* LTL-Satisfiability solver.
- We adapted to LTL assertional formulas, getting an *exponential* performance improvement.
- **Future Work**: Extend to other LTL-reasoning tasks: LTL-to-automata, LTL model checking, etc.