SAT-Based Explicit LTL Reasoning

Jianwen Li[1,2] Shufang Zhu[2] **Geguang Pu**[2] Moshe Y. Vardi[1]

1. Rice University

2. East China Normal University

August 22, 2016

Temporal Reasoning

- Church, 1957: Given a model M and MSO specification ϕ, 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*

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)

$$\phi ::= true \mid false \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi U \phi \mid X\phi$$

Assume LTL formulas are in NNF (Negation Normal Form)

- $X\psi$: ψ must hold in next step
- $\psi_1 U \psi_2$: ψ_2 will eventually hold, and before that ψ_1 must always hold.
- $\psi_1 R \psi_2$: ψ_2 holds until "released" by ψ_1
- LTL formulas are interpreted over infinite traces

LTL explicit model checking

Given model M a specification ϕ

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 τ is found then $M \models \phi$ fails and τ is counterexample.
4. If M is universal (allowing all traces), then model checking $\neg\psi$ checks satisfiabilit of ψ.

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

Automata Generation in Aalta

General idea: *syntactic splitting*

Consider ϕ to be a state:

1. Start from ϕ
2. $\phi \Leftrightarrow \bigvee_i (\alpha_i \wedge X\psi_i)$: (ϕ, α_i, ψ_i) is a transition in the automaton.
    - For Until/Release formula: $\psi_1 U \psi_2 \equiv (\psi_2 \vee (\psi_1 \wedge X(\psi_1 U \psi_2)))$ and $\psi_1 R \psi_2 \equiv (\psi_2 \wedge (\psi_1 \vee X(\psi_1 R \psi_2)))$.
3. For each new state ψ_i, repeat from step 2 until no new states are generated.

Automata Generation in Aalta

- *aUb*

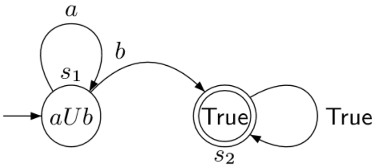

Figure: The Büchi automaton for *aUb*

$aUb = (b \wedge XTrue) \vee (a \wedge X(aUb))$

Bottleneck in Aalta

- Transformation $\phi \equiv \bigvee_i (\alpha_i \wedge X\psi_i)$ 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.

neXt Normal Form (XNF)

Definition 1 (neXt Normal Form)

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

For example,

- $(b \vee (a \wedge (X(aUb))))$ is in XNF.
- $a \wedge (b \vee cUa)$ is not in XNF.

neXt Normal Form (XNF)

Theorem 1

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

Proof.

1. $xnf(\phi) = \phi$ if ϕ is *true*, *false*, a literal l or a Next formula $X\psi$;
2. $xnf(\phi) = xnf(\phi_1) \wedge xnf(\phi_2)$ if $\phi = (\phi_1 \wedge \phi_2)$;
3. $xnf(\phi) = xnf(\phi_1) \vee xnf(\phi_2)$ if $\phi = (\phi_1 \vee \phi_2)$;
4. $xnf(\phi) = (xnf(\phi_2)) \vee (xnf(\phi_1) \wedge X\phi)$ if $\phi = (\phi_1 U \phi_2)$;
5. $xnf(\phi) = xnf(\phi_2) \wedge (xnf(\phi_1) \vee X\phi)$ if $\phi = (\phi_1 R \phi_2)$.

□

Treating LTL formulas Propositionally

- For an LTL formula ϕ in XNF, consider each Next subformula as an "atom", then we can treat ϕ as a propositional formula, denoted as ϕ^p.
- $\phi = (b \vee (a \wedge (X(aUb)))) \Rightarrow \phi^p = b \vee (a \wedge newVar)$, where $newVar = X(aUb)$.
- $\phi = Xa \vee (b \wedge X(cUb)) \Rightarrow \phi^p = newVar1 \vee (b \wedge newVar2)$, where $newVar1 = Xa$ and $newVar2 = X(cUb)$

Generate states via SAT solver

Given an LTL formula ϕ,

- Take $xnf(\phi)^p$ 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 $(\phi, \bigwedge L, \bigwedge \psi_i)((X\psi_i) \in X(A))$ 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 $\phi = (aUb) \wedge (cU\neg b)$.
- $xnf(\phi) = (b \vee (a \wedge X(aUb))) \wedge (\neg b \vee (c \wedge X(cU\neg b)))$
- SAT solver may give us an assignment of $\{a, \neg b, c, X(aUb), \neg X(cU\neg b)\}$
- Assignment indicates $(\phi, a \wedge \neg b \wedge c, (aUb))$ 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 submittd to SAT solver

Syntactic vs. Semantic Splitting: an Old Debate

- Beth, 1955: propositional tableaux – syntactic splitting
- Roth, 1966: ATPG – syntatic splitting
- David-Putnam-Logemann-Loveland, 1958-1963: DPLL (now CDCL) – semantic 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

Application to LTL satisfiability checking

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.

Formula type	ls4		TRP++		NuXmv-BMCINC		Aalta_v1.2		NuXmv-IC3-Klive		Aalta_v2.0	
/acacia/example	155	0	192	0	1	0	1	0	8	0	1	0
/acacia/demo-v3	68	0	2834	38	3	0	660	0	30	0	3	0
/acacia/demo-v22	60	0	67	0	1	0	2	0	4	0	1	0
/alaska/lift	2381	27	15602	254	1919	26	4084	63	867	5	1431	18
/alaska/szymanski	27	0	283	4	1	0	1	0	2	0	1	0
/anzu/amba	5820	92	6120	102	536	7	2686	40	1062	8	928	4
/anzu/genbuf	2200	30	7200	120	782	11	3343	54	1350	13	827	4
/rozier/counter	3934	62	4491	44	3865	64	3928	60	3988	65	2649	40
/rozier/formulas	167	0	37533	523	1258	19	1372	20	664	0	363	0
/rozier/pattern	2216	38	15450	237	1505	8	8	0	3252	17	8 9	0
/schuppan/O1formula	2193	34	2178	35	14	0	2	0	95	0	2	0
/schuppan/O2formula	2284	35	2566	41	1781	28	2	0	742	7	2	0
/schuppan/phltl	1771	27	1793	29	1058	15	1233	21	753	11	767	13
/trp/N5x	144	0	46	0	567	9	309	0	187	0	15	0
/trp/N5y	448	10	95	1	2768	46	116	0	102	0	16	0
/trp/N12x	3345	52	45739	735	3570	58	768	48	705	0	175	0
/trp/N12y	3811	56	19142	265	4049	67	7413	110	979	0	154	0
/forobots	990	0	1303	0	1085	18	2280	32	37	0	524	0
Total	32014	463	163142	2428	24769	376	31208	450	14261	126	7868	79

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 propery-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 *specification-pattern formulas*[1] (3000 random-conjunction formulas tested)

[1] http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml

Experiments on Random-Conjunction Formulas

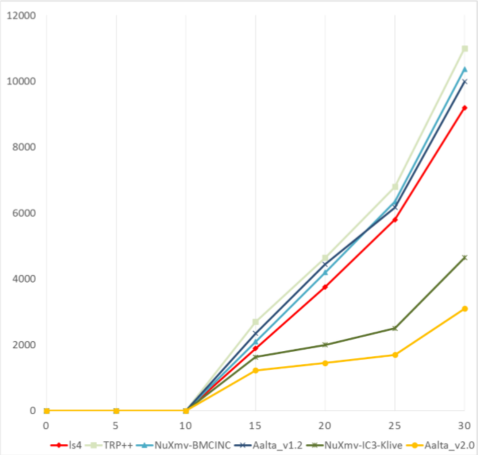

Figure: Results for LTL-satisfiability checking on random-conjunction formulas.

Handling LTL assertions

- By replacing SAT solver with SMT solver, we can also handle *assertional LTL*.
- Consider the formula $\phi = (F(k = 1) \wedge F(k = 2))$.
- 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.

SAT vs SMT

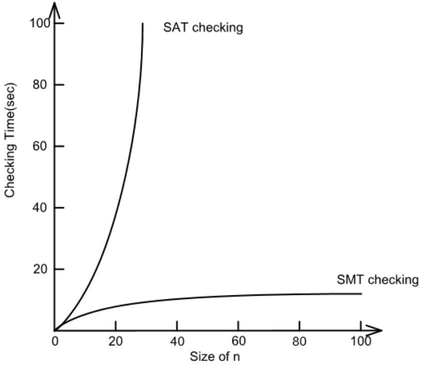

Figure: Results for LTL-satisfiability checking on $\bigwedge_{1 \leq i \leq n} F(k = i)$.

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.