Model Checking for Probabilistic Concurrent Systems

- DTMCs and CTL, LTL
- CTMCs and CSL
- MDPs and CTL and LTL
- CTMDPs and CSL
- MAs and CSL
- IMCs and CSL
- Probabilistic Hybrid Systems

LTL Satisfiability Checking Revisited

Jianwen Li[1], Lijun Zhang[2], Geguang Pu[1],
Moshe Vardi[3] and Jifeng He[1]

[1]Software Engineering Institute, East China Normal University

[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences

[3]Computer Science, Rice University, USA

November 6, 2013

Background

- LTL Model Checking [CGP99] has been very useful;
- However, people often make mistakes in writing LTL formulas;
- There are many works on "property assurances";
- Among them, satisfiability checking is a basic check.

Prior Works on LTL Satisfiability Checking

- Model-checking based
  - SPOT (+ SPIN) [RV07]
  - PANDA + CadenceSMV [RV11]
  - NuSMV-BDD, NuSMV-BMC [CCGR00]
- Temporal Resolution
  - trp++ [HK03]
- Tableau Framework
  - pltl [Sch98]
  - lwb [Sch98]
- Others
  - alaska [DDMR08]
  - tspass [LH10]
- see *Evaluating LTL Satisability Solvers* [SD11]

Motivation (1)

1. We follow the automata-theoretic framework
2. ϕ is sat? $\Leftrightarrow$ $\mathcal{A}_\phi$ is not empty?
3. The tableau construction[GPVW95] is well-known from LTL to Büchi automton
4. But the generated automton may be exponential.

Motivation (2)

1. Individual properties are likely to be satisfiable
2. Combined properties are likely to be unsatisfiable
3. Showing satisfiable means finding a model for the property
4. Can we take advantage of that ?
5. Yes !
6. Our approach: On-the-fly search + Obligation Set

Motivation (3)

Consider the following cases:

- $(\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots)Ub$
- $(\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots)Rb$

There exists the core propertie set $\{b\}$ for above formulas!

b^ω satisfies both formulas above.

Our idea: Define the *Obligation Set*, which provides a easy way to check satisfiable formulas!

Obligation Set (1)

Definition (Obligation Set)

For a formula ϕ, we define its obligation set, denoted by $Olg(\phi)$, as follows:

1. $Olg(\mathrm{tt}) = \{\emptyset\}$ and $Olg(\mathrm{ff}) = \{\{\mathrm{ff}\}\}$;
2. If ϕ is a literal, $Olg(\phi) = \{\{\phi\}\}$;
3. If $\phi = X\psi$, $Olg(\phi) = Olg(\psi)$;
4. If $\phi = \psi_1 \vee \psi_2$, $Olg(\phi) = Olg(\psi_1) \cup Olg(\psi_2)$;
5. If $\phi = \psi_1 \wedge \psi_2$, $Olg(\phi) = \{O_1 \cup O_2 \mid O_1 \in Olg(\psi_1) \wedge O_2 \in Olg(\psi_2)\}$;
6. If $\phi = \psi_1 U \psi_2$ or $\psi_1 R \psi_2$, $Olg(\phi) = Olg(\psi_2)$;

For $O \in Olg(\phi)$, we refer to it as an *obligation* of ϕ.

Obligation Set (2)

Example

- $Olg(aUb) = \{\{b\}\}$;
- $Olg(G(bUc \wedge dUe)) = \{\{c, e\}\}$;
- $Olg(G(bUc \vee dUe)) = \{\{c\}, \{e\}\}$.

Definition (Consistent Obligation)

We say an obligation O of ϕ is *consistent* iff for all $a \in O$ we have that $\bigwedge a \not\equiv \mathrm{ff}$.

Theorem (Satisfiability Theorem for Consistent Obligation)

Assume $O \in Olg(\phi)$ is a consistent obligation. Then, $O^\omega \models \phi$.

Obligation Set (3)

- How about if there is no consistent obligation $O \in Olg(\phi)$?
- Then we introduce the on-the-fly checking on the transition system of ϕ.
- Why not check on the (generalized-)Büchi automaton ?
  —- We want to use *Obligation Sets*!

Tagging formulas (1)

- Transition systems are similar to tableau-based GBA
- This makes the checking easier
- But we simplified too much, the transition system does not carry enough information
- We use formula tagging to mark satisfaction of until formulas on the edges of transition systems

Tagging formulas (2)

Given a formula ϕ, we denote $U(\phi)$ the set of until subformulas of ϕ. S_a is the set of occurrances of atom a, and $right(\psi)$ is the set of right subformulas of ψ. Then:

Definition (Tagging Formula)

Let $a \in AP$ be an atom appearing in ϕ. Then, the tagging function $F_a : S_a \to 2^{U(\phi)}$ is defined as: $\psi \in F_a(a_i)$ iff a_i appears in $right(\psi)$. We define the *tagged formula* ϕ_t as the formula obtained by replacing a_i by $a_{F_a(a_i)}$ for each $a_i \in S_a$.

Example

Consider $\phi = aU(a \wedge aU\neg a)$. Let $\psi_u = aU\neg a$, and $S_a = \{a_1, a_2, a_3, a_4\}$. From the definition we know $F_a(a_1) = \emptyset, F_a(a_2) = F_a(a_3) = \{\phi\}$, and $F_a(a_4) = \{\phi, \phi_u\}$.

LTL Transition System (LTS) (1)

Definition (Normal Form Expansion)

The *normal form* of an LTL formula ϕ, denoted as $NF(\phi)$, is :

1. $NF(\phi) = \{\phi \wedge X(\mathrm{tt})\}$ if $\phi \not\equiv \mathrm{ff}$ is a propositional formula. If $\phi \equiv \mathrm{ff}$, we define $NF(\mathrm{ff}) = \emptyset$;
2. $NF(X\phi) = \{\mathrm{tt} \wedge X(\psi) \mid \psi \in DF(\phi)\}$;
3. $NF(\phi_1 U \phi_2) = NF(\phi_2) \cup NF(\phi_1 \wedge X(\phi_1 U \phi_2))$;
4. $NF(\phi_1 R \phi_2) = NF(\phi_1 \wedge \phi_2) \cup NF(\phi_2 \wedge X(\phi_1 R \phi_2))$;
5. $NF(\phi_1 \vee \phi_2) = NF(\phi_1) \cup NF(\phi_2)$;
6. $NF(\phi_1 \wedge \phi_2) = \{(\alpha_1 \wedge \alpha_2) \wedge X(\psi_1 \wedge \psi_2) \mid \forall i = 1, 2.\ \alpha_i \wedge X(\psi_i) \in NF(\phi_i)\}$;

Note: Let $\phi = \bigvee_{1 \leq i \leq n} \phi_i$ and we define $DF(\phi) = \{\phi_i | 1 \leq i \leq n\}$

LTL Transition System (LTS) (2)

Definition (LTL Transition System)

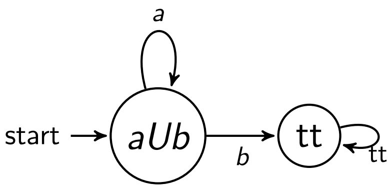
The labelled transition system T_ϕ generated from the formula ϕ is a tuple $\langle \Sigma, S_\phi, \rightarrow, \phi \rangle$ where ϕ is the initial state, and:

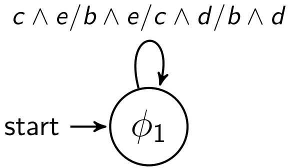
1. the transition relation $\rightarrow$ is defined by: $\psi_1 \xrightarrow{\alpha} \psi_2$ iff there exists $\alpha \wedge X(\psi_2) \in NF(\psi_1)$;
2. S_ϕ is the smallest set of formulas such that $\phi \in S_\phi$, and $\psi_1 \in S_\phi$ and $\psi_1 \xrightarrow{\alpha} \psi_2$ implies $\psi_2 \in S_\phi$.

LTL Transition System (LTS) (3)

Example

- aUb:
  1. $NF(aUb) = \{b \wedge X\mathrm{tt}, a \wedge X(aUb)\}$;
  2. $NF(\mathrm{tt}) = \mathrm{tt} \wedge X(\mathrm{tt})$.
- $\phi_1 = G(bUc \wedge dUe)$:
  1. $NF(\phi_1) = \{c \wedge e \wedge X\phi_1, b \wedge e \wedge X\phi_2, c \wedge d \wedge X\phi_3, b \wedge d \wedge X\phi_4\}$: here $\phi_2 = bUc \wedge \phi_1$, $\phi_3 = dUe \wedge \phi_1$, and $\phi_4 = bUc \wedge dUe \wedge \phi_1$.
  2. $NF(\phi_2) = NF(\phi_3) = NF(\phi_4)$.





On-the-fly Satisfiability Checking

Theorem

$SAT(\phi)$ iff there exists a SCC B of TS_ϕ and a state ψ in B such that ϕ can expand to ψ and, $L(B)$ is a superset of some obligation $O \in Olg(\psi)$.

Note: $L(B)$ denotes the set of literals across the SCC B.

On-the-fly Satisfiability Checking

The whole framework of our new algorithm is as follows:

1. We first tag the formula ϕ. Then we construct T_ϕ, where we explore the states in an on-the-fly manner, by performing nested depth-first [CVWY92],
2. Whenever a formula is found, we compute the obligation set. In case that it contains a consistent obligation set, we return *true*,
3. If a SCC B is reached, $\phi \in B$, and $L(B)$ is a superset of some obligation set $O \in Olg(\phi)$, we return *true*,
4. If all SCCs are explored, but do not have the property in step 3, we return *false*.

Tool : *Aalta*[1].

[1] www.lab205.org/aalta

Experimental Platform

- Platform: SUG@R cluster[2] : 2.83GHz Intel Xeon Harpertown CPUs with 16GB RAM per node; Red Hat 4.1.2
- Benchmarks:
  1. From [RV07]: more than 100,000 random, 8 pattern, 3 counter formulas;
  2. Random conjunction formulas: $\bigwedge_{1 \leq i \leq n} P_i$, where P_i is a random specification pattern[3](totally 44 types).
- Timeout is set to be 300 seconds.

[2]http://www.rcsg.rice.edu/sharecore/sugar/

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

Why random conjunctions?

- To check scaling we need large formulas
- But typical properties are not large
- Thus we propose the new random conjunction of specification patterns
- Corresponds to checking the interaction of properties

Experimental Methods

- Compare *Aalta* to model-checking-based LTL satisfiability solvers;
- Explicit : SPOT [DLP04] + SPIN [Hol03]
- Symbolic: PANDA + CadenceSMV[RV11]
- Compare the solvers' scalability on large formulas
- Study the impact of heuristic strategies
- Separate the satisfiable and unsatisfiable formulas

Experimental Results (1)

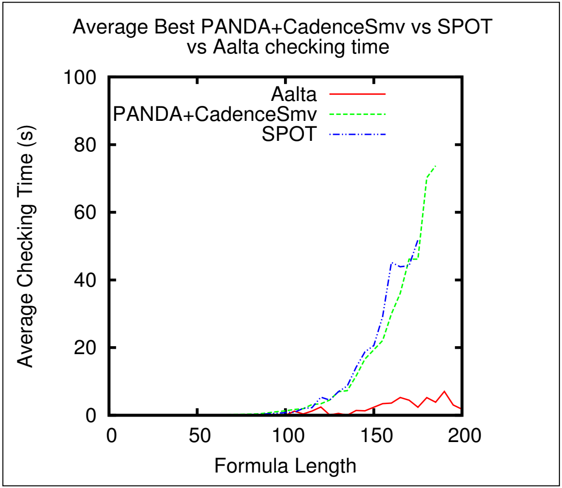


Figure : Experimental results for random formulas with 3 variables.

Experimental Results (2)

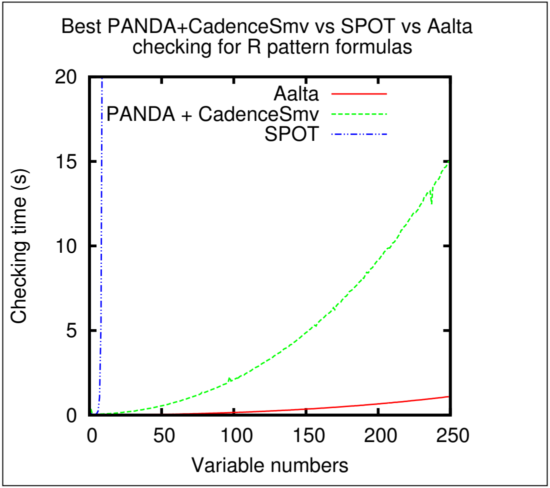


Figure : Experimental results for $R(n) = \bigwedge_{i=1}{}^{n}(GFp_i \vee FGp_{i+1})$.

Experimental Results (3)

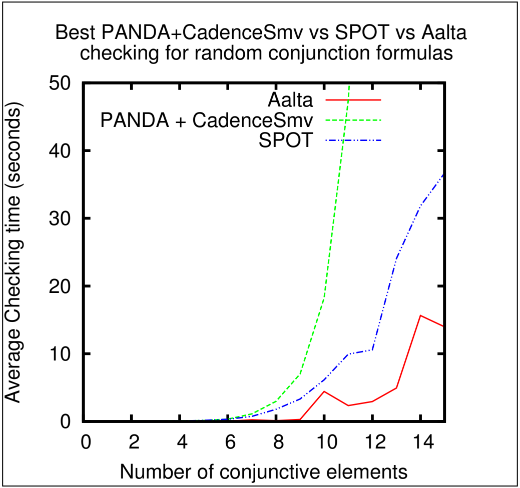


Figure : Experimental results for random conjunctive formulas.

Experimental Results (4)

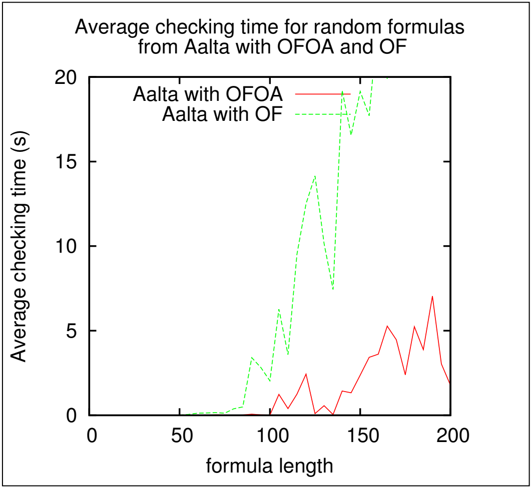


Figure : Experimental results for 3-variable random formulas from *Aalta* with OFOA and OF.

Experimental Results (5)

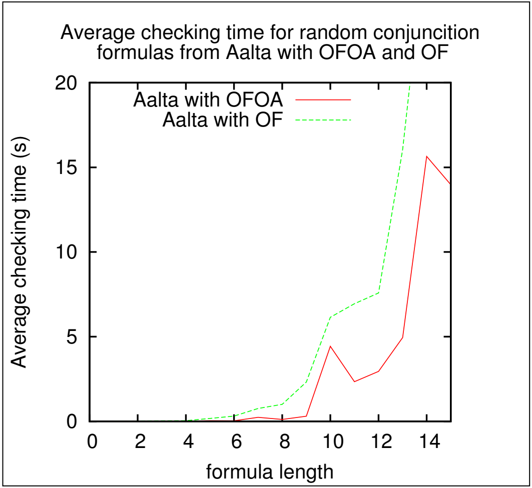


Figure : Experimental results for random conjunction formulas from *Aalta* with OFOA and OF.

Experimental Results (6)

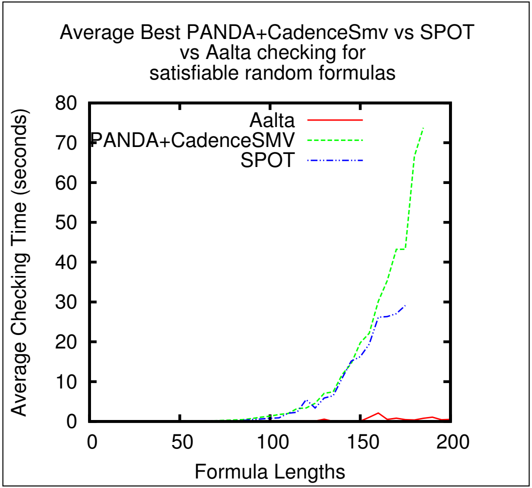


Figure : Experimental results for satisfiable random formulas.

Experimental Results (7)

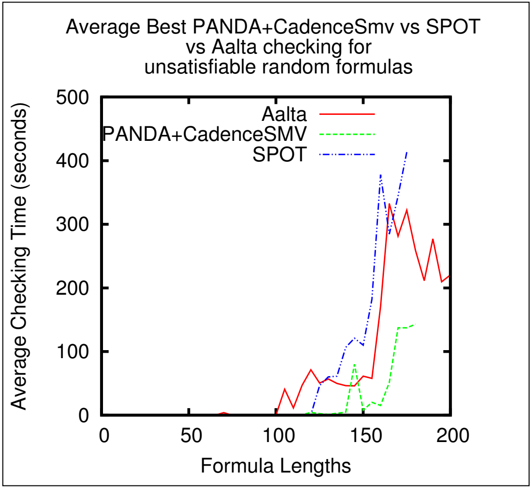


Figure : Experimental results for unsatisfiable random formulas.

Conclusion

- Pro-SAT heuristic strategies are effective
- What about pro-UNSAT heuristics ?
- “Mirror Mirror on The Wall, who is the fastest of them all”?
- More work is needed.

Thanks!

Reference I

- A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri, *NuSMV: a new symbolic model checker*, It'l J. on Software Tools for Technology Transfer **2** (2000), no. 4, 410–425.
- E.M. Clarke, O. Grumberg, and D. Peled, *Model checking*, MIT Press, 1999.
- C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis, *Memory efficient algorithms for the verification of temporal properties*, Formal Methods in System Design **1** (1992), 275–288.
- M. De Wulf, L. Doyen, N. Maquet, and J.-F. Raskin, *Antichains: Alternative algorithms for ltl satisfiability and model-checking*, Proc. 14th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol. 4963, Springer, 2008, pp. 63–77.

Reference II

A. Duret-Lutz and D. Poitrenaud, *SPOT: An extensible model checking library using transition-based generalized büchi automata*, Proc. 12th Int'l Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, IEEE Computer Society, 2004, pp. 76–83.

R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper, *Simple on-the-fly automatic verification of linear temporal logic*, Protocol Specification, Testing, and Verification (P. Dembiski and M. Sredniawa, eds.), Chapman & Hall, 1995, pp. 3–18.

Ullrich Hustadt and Boris Konev, *Trp++ 2.0: A temporal resolution prover*, In Proc. CADE-19, LNAI, Springer, 2003, pp. 274–278.

G.J. Holzmann, *The spin model checker: Primer and reference manual*, Addison-Wesley, 2003.

Reference III

Michel Ludwig and Ullrich Hustadt, *Implementing a fair monodic temporal logic prover*, AI Commun. **23** (2010), no. 2-3, 69–96.

K.Y. Rozier and M.Y. Vardi, *LTL satisfiability checking*, Proc. 14th International SPIN Workshop, Lecture Notes in Computer Science, vol. 4595, Springer, 2007, pp. 149–167.

______, *A multi-encoding approach for LTL symbolic satisfiability checking*, Proc. 17th Int'l Symp. on Formal Methods, Lecture Notes in Computer Science, vol. 6664, Springer, 2011, pp. 417–431.

S. Schwendimann, *A new one-pass tableau calculus for pltl*, Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX '98, Springer-Verlag, 1998, pp. 277–292.

Reference IV

V. Schuppan and L. Darmawan, *Evaluating ltl satisfiability solvers*, Proceedings of the 9th international conference on Automated technology for verification and analysis, AVTA'11, Springer-Verlag, 2011, pp. 397–413.