Aalta: An LTL Satisfiability Checker over Infinite/Finite Traces

Jianwen Li
Software Engineering
East China Normal University
Shanghai, China
lijwen2748@gmail.com

Yinbo Yao
Software Engineering
East China Normal University
Shanghai, China
snowingsea@gmail.com

Geguang Pu*
Software Engineering
East China Normal University
Shanghai, China
ggpu@sei.ecnu.edu.cn

Lijun Zhang
State Key Laboratory of
Computer Science
ISCAS, Beijing, China
zhanglijun79@gmail.com

Moshe Y. Vardi
Computer Science
Rice University
Houston, USA
vardi@cs.rice.edu

Jifeng He
Software Engineering
East China Normal University
Shanghai, China
jifeng@sei.ecnu.edu.cn

ABSTRACT

Linear Temporal Logic (LTL) is been widely used nowadays in verification and AI. Checking satisfiability of LTL formulas is a fundamental step in removing possible errors in LTL assertions. We present in this paper *Aalta*, a new LTL satisfiability checker, which supports satisfiability checking for LTL over both infinite and finite traces. *Aalta* leverages the power of modern SAT solvers. We have conducted a comprehensive comparison between *Aalta* and other LTL satisfiability checkers, and the experimental results show that *Aalta* is very competitive. The tool is available at `www.lab205.org/aalta`.

Categories and Subject Descriptors

F.3.1 [**Logics and Verification**]: Specifying and Verifying and Reasoning about programs

General Terms

Verification, Algorithms

Keywords

Model Checking, Temporal Logic, Satisfiability

1. INTRODUCTION

Linear Temporal Logic (LTL) was introduced into computer science in [8], as a formal property description language for non-terminating reactive systems. There is by now a rich body of knowledge regarding automated-reasoning support for LTL in the formal-verification community. Our main focus here is on the *satisfiability problem*, which asks if a given LTL formula has a satisfying model. This basic problem has attracted a fair amount of attention over the past few years, as a main approach to *property assurance*, which aims at eliminating errors in LTL assertions [9]. Thus, efficient decision procedures for reasoning about LTL formulas are quite desirable in practice.

Researchers in AI are also attracted by the rich expressiveness of LTL, cf. [3]. AI applications, however, are typically interested only in *finite* traces, while verification applications are typically interested in *infinite* traces. For instance, temporally extended goals [2] can be regarded as finite desirable traces of states and a plan is correct if its execution succeeds in yielding one of these desirable traces. Therefore, LTL_f was introduced in [3]; this logic has the *same* syntax as LTL, but is interpreted over finite traces.

Most works on LTL satisfiability focus on infinite-trace semantics. To check satisfiability over finite traces, one can reduce finite-trace satisfiability to infinite-trace satisfiability [3]: one transforms an LTL_f formula ϕ to an LTL formula ϕ' such that ϕ is finite-trace satisfiable iff ϕ' is infinite-trace satisfiable. The transformation is simple (linear blow-up), so an LTL satisfiability checker can be easily converted to an LTL_f satisfiability checker. But LTL satisfiability checking requires searching for a *fair cycle*, which is not required for LTL_f satisfiability checking [3]. Thus, a reduction of LTL_f satisfiability to LTL satisfiability may add unnecessary overhead to LTL_f satisfiability checking. To overcome this disadvantage, we recently used directly the finite-trace semantics of LTL_f, and proposed a very efficient satisfiability-checking procedure [7].

To support satisfiability checking for LTL on both infinite and finite traces, we present here the tool *Aalta*, a new LTL satisfiability checker that leverages the power of modern SAT solvers. The framework of *Aalta* is based on *LTL transition systems*, which we proposed in previous work [5]. To the best of our knowledge, this is the first tool to directly support LTL satisfiability checking over both infinite and finite traces.

We compare our tool empirically with other extant LTL satisfiability solvers. We reach two conclusions from these experiments. First, for satisfiability checking over infinite

*Geguang Pu is the corresponding author.

FSE'14 , November 16–22, 2014, Hong Kong, China

trace, *Aalta* behaves best in the overall performance, but no solver dominates in performance. Second, for satisfiability checking over finite trace, *Aalta* performs best and has significant performance boost compared to other solvers.

Related work. There have been several approaches to LTL satisfiability checking problem. The model-checking approach reduces LTL satisfiability to LTL model checking by checking the *negation* of the given formula against a *universal* model. We use the NuSMV tool [1] as a representative of this strategy. The tableau-based [11] approach applies an on-the-fly search in the underlying automaton transition system. The pltl[1] solver is a representative of this approach. The temporal-resolution approach explores the unsatisfiable core using a deductive system [10]. The tool TRP++ [10] is a representative of this approach. Our own approach [5, 7] follows the automata-theoretic approach [12] and reduces satisfiability checking to automaton emptiness checking, which we perform by using two heuristics: *on-the-fly search* and *obligation sets*.

The rest of this paper is organized as follows. Section 2 introduces LTL and the algorithms implemented in *Aalta*. Section 3 describes the architecture of the tool. Section 4 provides experimental results. Finally, Section 5 offers some concluding remarks.

2. PRELIMINARIES

2.1 Linear Temporal Logic

Let AP be a set of atomic properties, and the original definition for the syntax of LTL (and LTL_f) formulas are as follows:

$$\phi \ ::= \mathsf{ff} \mid \mathsf{tt} \mid \ a \mid \neg\phi \mid \phi\wedge\phi \mid X\ \phi \mid \phi\ U\phi;$$

where $a \in AP$, ϕ is an LTL formula. Obviously every boolean formula is an LTL formula. Besides, LTL also contains two temporal operators X (Next) and U (Until).

In our methodologies, LTL formulas are required to be in NNF (Negative Normal Form), which can be acquired by pushing all negations in front of only atoms. For this purpose, the dual operator of U is introduced, i.e. the R (Release) operator, and it holds that $\neg(\phi_1 U\phi_2) \equiv \neg\phi_1 R\neg\phi_2$.

Hence, let $L = AP \cup \{\neg a | a \in AP\}$, and LTL formulas are interpreted over infinite words in $\Sigma = 2^L$. Let $\xi = \omega_0\omega_1\ldots \in \Sigma^\omega$, and we use the notation $\xi^i = \omega_0\omega_i\ldots\omega_{i-1} (i \geq 1)$ to represent the prefix of ξ before position i (i is not included). Also we use the notation $\xi_i = \omega_i\omega_{i+1}\ldots$ to represent the suffix of ξ from position i (i is included). Then $\xi \models \phi$ iff

- if $\phi = X\phi_1$ then $\xi_1 \models \phi_1$;
- if $\xi \models \phi_1\ U\ \phi_2$, then there exists $i \geqslant 0$ such that $\xi_i \models \phi_2$ and for all $0 \leqslant j < i, \xi_j \models \phi_1$;
- if $\xi \models \phi_1\ R\ \phi_2$, then either for all $i \geq 0$ it holds that $\xi_i \models \phi_2$, or there exists $j \geq 0$ such that $\xi_j \models \phi_1$ and for all $0 \leq i \leq j$ it holds $\xi_i \models \phi_2$;

The cases when ϕ are boolean formulas are trivial, and we omit the definitions here.

For LTL_f formulas interpreted over finite traces, let $\eta \in \Sigma^*$ and $|\eta|$ be the length of η. The semantics are then defined in a rather straightforward way. For example, $\eta \models \phi_1 U\phi_2$ if there exists $0 \leq i < |\eta|$ such that $\eta_i \models \phi_2$ and for all $0 \leqslant j < i, \eta_j \models \phi_1$. For the NNF forms, a weak next operator (X_w) is introduced, readers can refer to [7] for more details.

[1]http://users.cecs.anu.edu.au/ rpg/PLTLProvers/

2.2 Algorithms

Aalta implements the *obligation-based* LTL satisfiability checking algorithms that are proposed in our previous work [5, 6, 7]. We first introduce briefly the *obligation formula*, which is the key concept of our approach and then present the main decision procedure behind *Aalta*. We also discuss the optimizations by SAT solver adopted in the tool.

Obligation Formula. Given an LTL (and LTL_f) formula ϕ, we can extract an obligation formula $of(\phi)$ by eliminating the temporal operators and keeping only the right part of temporal subformulas. So $of(\phi)$ is essentially a boolean formula. For example, $of(X(aUb)) = b$, in which all temporal operators are eliminated and only the right parts of temporal subformulas are extracted. (Here is b.) For more complicated cases, we have $of(aR(bUc\wedge bRd)) = c\wedge d$, $of(XXXXa) = a$ and etc. The obligation formula plays a key role in the reduction from LTL satisfiability checking to Boolean SAT one.

Decision Procedure. By leveraging the obligation formula, we can achieve a new LTL satisfiability checking framework based on the *LTL (LTL_f) transition system* we proposed. Summarily, the general checking process can be organized as follows:

1. Construct the *LTL (LTL_f) transition system* (each state is a formula) in the on-the-fly manner;
2. For LTL checking, if a SCC (Strong Connected Component) containing a formula ψ is found and the literals collected along the SCC is an assignment of $of(\psi)$, then we can conclude the input formula is satisfiable;
3. In the worst case, the algorithm returns unsatisfiable after the whole system has been explored.

Optimizations by SAT Solver. The power of *obligation formula* for the LTL (LTL_f) satisfiability checking is to introduce some optimizations by utilizing SAT solvers to accelerate the checking process. For example, we have proven that if $of(\phi)$, i.e. the obligation formula of ϕ, is satisfiable then ϕ is also satisfiable. This potentially facilitates us to check the formula before the generation of transition system. Moreover, if we add the positional information into the obligation formula, which we denote as ofp, we can also get the acceleration for checking unsatisfiable formulas. Additionally, a complete reduction from LTL_f checking to SAT solving is proposed for the global formulas (with the form of $G\phi$). By defining the global obligation formula $\mathsf{ofg}(\phi)$ for the global LTL_f formula ϕ, we have shown that ϕ is satisfiable iff $\mathsf{ofg}(\phi)$ is satisfiable.

3. TOOL ARCHITECTURE

Figure 1 shows the architecture of *Aalta* tool. *Aalta* provides the interfaces of *input* and *output*, and the core modules that are composed of *Parser*, *NNF Converter*, checkers for *LTL and LTL_f*, and the SAT solver which is called by checkers.

Input: *Aalta* accepts the strings starting with letters as formula atoms. Besides, *Aalta* recognizes all propositional

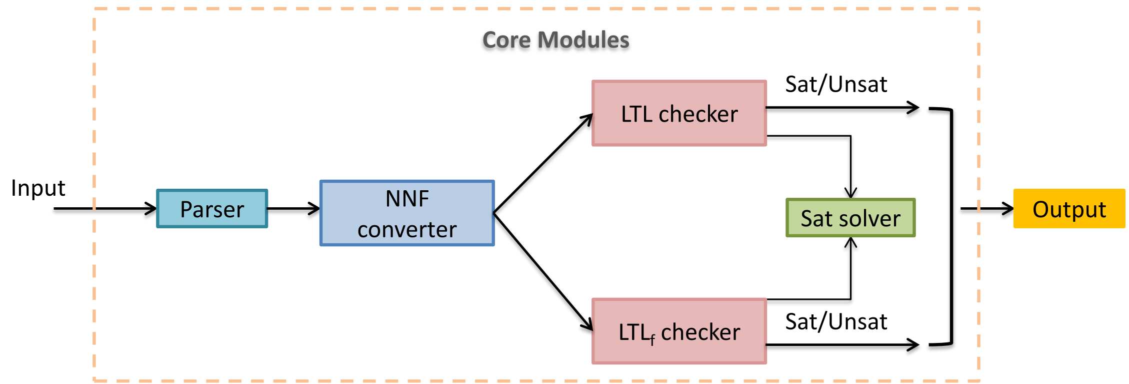

Figure 1: The Architecture of *Aalta*.

Table 1: The operator representations in *Aalta*.

operators	symbols
$\neg$	!, $\sim$
$\wedge$	& , &&
$\vee$	|, ||
X	X
X_w	N
U	U
R	R , V
G	G, []
F	F, $\langle\rangle$
$\rightarrow$	$\rightarrow$
$\leftrightarrow$	$\leftrightarrow$
tt	true, TRUE
ff	false, FALSE

operators and the temporal ones of X, X_w, U and R. Specially, it also recognizes the G and F operators which are defined as $G\phi \equiv \mathsf{ff}R\phi$ and $F\phi \equiv \mathsf{tt}U\phi$. Table 1 shows the mapping between the formula operators and their representations in *Aalta*. Note that we use the symbol 'N' to represent the weak next operator X_w which is appeared in LTL_f.

Output: *Aalta* produces the satisfiability checking results (sat/unsat) for the input formula. If the result is satisfiable (sat) and the parameter "-e" is given, *Aalta* also shows an evidence satisfying the input formula. To represent such an infinite witness, *Aalta* uses the "(s)" to denote the infinite appearance of string s. For example, the output string "a(bc)" of *Aalta* actually represents the infinite word "$a(bc)^\omega$".

Core Modules: In the architecture, a *Parser* is integrated to initialize the satisfiability checking process in *Aalta* by recognizing the input string. As our approach requires the formula under check in NNF form, the *NNF converter* is implemented to achieve the goal. Note that the converter automatically generates the corresponding NNF form for satisfiability checking over infinite or finite trace according to the given parameter("-l" for LTL formulas and "-f" for LTL_f formulas.) *Aalta* launches the different checker (LTL or LTL_f checker) based on the given parameter.

SAT Solver: The *SAT solver* module utilizes the open-source Minisat solver[2], and is invoked by the LTL and LTL_f checkers which implement the SAT-based algorithms we proposed in our earlier work. Minisat is seamlessly integrated as a part of *Aalta* rather than an external solver. Note that Minisat requires the DIMACS format (see Minisat tutorials) representing CNF clauses for input. We thus implement in *Aalta* a translation algorithm from boolean formulas to its CNF formatted by DIMACS.

The *SAT Solver* module is invoked in the following way. The main procedure computes the transition system of the input formula under check in an on-the-fly manner, and the corresponding *obligation formulas* are extracted for each new formula (state) generated. Then the obligation formulas are passed to the Minisat and checked by the *SAT Solver* module. Several heuristics are designed to decide the satisfiability of the original formula based on the results of Minisat tool. If the heuristics cannot decide the satisfiability according to the results of *SAT Solver* module, the main procedure repeatedly processes on the whole transition system until it obtains the result.

4. EXPERIMENTS

In this section we show the comparison results between *Aalta* and other representing tools on LTL satisfiability checking over infinite or finite trace respectively.

We use the BlueBioU cluster[3] in Rice university as the experimental platform. The cluster consists of 47 IBM Power 755 nodes, each of which contains four eight-core POWER7 processors running at 3.86GHz. Every tested tool occupies a unique node, which guarantees all tools are run in the same environment. The time is measured by Unix time command, and each test case has the maximal limitation of 60 seconds.

For LTL satisfiability checking performance, we select the existed tools, i.e. pltl [11], TPR++ [10], NuSMV [1], for comparison with *Aalta*. Note that NuSMV implements the BDD-based model checking and BMC (bounded model checking based on SAT), we do the comparison for both of the techniques, which is denoted as NuSMV-BDD and NuSMV-BMC separately in Table 2. We use the benchmarks called *schuppan-collected* in [4]. The corresponding checking costs (seconds) are displayed explicitly in Table 2. From the table we can see *Aalta* takes the advantage and performs best in total (see the last cell of the table), though none of the solvers can dominate in every case.

[2]http://minisat.se/MiniSat.html

[3]`http://www.rcsg.rice.edu/sharecore/bluebiou/`

Table 2: Comparison results on the Schuppan-collected benchmarks for LTL satisfiability checking.

Formula Type	pltl	TRP++	NuSMV -BDD	NuSMV -BMC	*Aalta*
/acacia	367.1	25.7	41.3	1.8	506.8
/alaska	5800.6	12641.5	7259.1	2797.2	3954.2
/anzu	3815.1	10729.7	12124.0	1093.2	5202.4
/rozier	1794.8	53234.0	15224.9	10676.1	3135.5
/schuppan	3079.8	4149.4	3838.8	4329.1	97.4
/trp	27475.5	3898.0	34533.8	23849.6	4392.2
Total	44576.2	84826.2	73030.5	44250.0	20626.4

Table 3: Experimental results on Schuppan-collected formulas for LTL_f satisfiability checking.

Formula type	*Aalta*	*Polsat*
/acacia	4.9	609.3
/alaska	24.2	7326.9
/anzu	5727.8	5770.8
/rozier	2416.1	3526.2
/schuppan	232.3	1874.6
/trp	6838.4	30392.7
Total	15244.2	50038.2

We recall that usual approach to LTL satisfiability checking over finite trace (LTL_f satisfiability checking) is to reduce it to standard LTL satisfiability checking over infinite trace [3]. We compare *Aalta* with *Polsat* [4] tool which implements this reduction approach. Note *Polsat* is a portfolio LTL solver which has integrated most of the existing LTL satisfiability solvers (mentioned above), and it takes the best checking result among them for the input formula. Since LTL formulas are also LTL_f formulas, we use the *schuppan-collected* benchmarks as well. Table 3 shows the comparing results between *Aalta* and *Polsat*. The unit of the checking cost shown in the table is also in second. We can see clearly that *Aalta* performs better than *Polsat*, with more than 3 times speed-up.

5. CONCLUSION

In this paper we present the *Aalta* tool, which is an LTL satisfiability checker over both infinite and finite traces. We compare the performance between *Aalta* and other off-the-shelf tools and the empirical results show that *Aalta* is a very competitive solver.

6. ACKNOWLEDGEMENTS

We thank anonymous reviewers for the useful comments. Geguang Pu is partially supported by Shanghai Knowledge Service Platform No. ZF1213. Jianwen Li is partially supported by SHEITC Project 130407. Jifeng He is partially supported by NSFC Project No. 61021004. Lijun Zhang is supported by NSFC projects No. 61361136002 and 91118007, and by the CAS/SAFEA International Partnership Program for Creative Research Teams. Moshe Vardi is supported in part by NSF grants CNS 1049862 and CCF-1139011, by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering", by BSF grant 9800096, and by gift from Intel.

7. REFERENCES

[1] A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, pages 359–364. Springer, 2002.

[2] G. De Giacomo and M.Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *Proc. European Conf. on Planning*, pages 226–238. Springer, 1999.

[3] G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, IJCAI'13, pages 2000–2007. AAAI Press, 2013.

[4] J. Li, G. Pu, L. Zhang, M. Y. Vardi, and J. He. Polsat: A portfolio ltl satisfiability solver. *CoRR*, abs/1311.1602, 2013.

[5] J. Li, L. Zhang, G. Pu, M. Y. Vardi, and J. He. LTL satisfibility checking revisited. In *The 20th International Symposium on Temporal Representation and Reasoning*, pages 91–98, 2013.

[6] J. Li, G. Pu, L. Zhang, M. Y. Vardi, and J. He. Fast LTL Satisfiability Checking by SAT Solvers. *CoRR*, abs/1401.5677, 2014.

[7] J. Li, L. Zhang, G. Pu, M. Y. Vardi, and J. He. LTL_f satisfibility checking. In *The 21th European Conference on Artificial Intelligence*, pages 91–96, 2014.

[8] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. on Foundations of Computer Science*, pages 46–57, 1977.

[9] K. Y. Rozier and M. Y. Vardi. LTL satisfiability checking. In *Int'l J. on Software Tools for Technology Transfer*, pages 123–137, 2010.

[10] V. Schuppan. Towards a notion of unsatisfiable cores for LTL. In *Fundamentals of Software Engineering*, pages 129–145, 2010.

[11] S. Schwendimann. A new one-pass tableau calculus for pltl. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 277–292. Springer-Verlag, 1998.

[12] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, pages 238–266. Springer, 1996.