Aalta: An LTL Satisfiability Checker over Infinite/Finite Traces

Jianwen Li  
Software Engineering  
East China Normal University  
Shanghai, China  
ljwen2748@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 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, $\text{LTL}_f$ 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 formula $\phi$ to an LTL formula $\phi'$ such that $\phi$ is finite-trace satisfiable iff $\phi'$ 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 $\text{LTL}_f$ satisfiability checking [3]. Thus, a reduction of $\text{LTL}_f$ satisfiability to LTL satisfiability may add unnecessary overhead to $\text{LTL}_f$ satisfiability checking. To overcome this disadvantage, we recently used directly the finite-trace semantics of $\text{LTL}_f$, 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
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 phi solver is a representative of this approach. The temporal-resolution approach explores the unsatisfiable core using a deductive system [10]. 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:

\[ \phi ::= ff \mid tt \mid a \mid \neg \phi \mid \phi \land \phi \mid X \phi \mid U \phi; \]

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

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 \( \neg (\phi_1 U \phi_2) \equiv \phi_1 R \neg \phi_2 \).

Hence, let \( L = AP \cup \{ \neg a | a \in AP \} \), and LTL formulas are interpreted over infinite words in \( \Sigma = 2^L \). Let \( \xi = \omega_0 \omega_1 \ldots \omega_{i-1} \) (\( i \geq 1 \)) to represent the prefix of \( \xi \) 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 \( \xi \) 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 \( i \geq 0 \) such that \( \xi_1 \models \phi_2 \) and for all \( 0 \leq j < i, \xi_j \models \phi_1 \);
- 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 \( 0 \leq i < j \) it holds \( \xi_i \models \phi_2 \);

The cases when \( \phi \) 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 \( \eta \). The semantics are then defined in a rather straightforward way. For example, \( \eta \models \phi_1 U \phi_2 \)

http://users.cecs.anu.edu.au/~tpg/PLTLProvers/
operators and the temporal ones of $X$, $X_w$, $U$ and $R$.  

<table>
<thead>
<tr>
<th>operators</th>
<th>symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg$</td>
<td>!, ~</td>
</tr>
<tr>
<td>$\wedge$</td>
<td>$&amp;$, $&amp;&amp;$</td>
</tr>
<tr>
<td>$\vee$</td>
<td>$</td>
</tr>
<tr>
<td>$X$</td>
<td>$\mathbb{X}$</td>
</tr>
<tr>
<td>$X_w$</td>
<td>$\mathbb{W}$</td>
</tr>
<tr>
<td>$U$</td>
<td>$\mathbb{U}$</td>
</tr>
<tr>
<td>$R$</td>
<td>$\mathbb{R}$</td>
</tr>
<tr>
<td>$G$</td>
<td>$\mathbb{G}$</td>
</tr>
<tr>
<td>$F$</td>
<td>$\mathbb{F}$</td>
</tr>
<tr>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$\leftrightarrow$</td>
<td>$\leftrightarrow$</td>
</tr>
<tr>
<td>$tt$</td>
<td>true, TRUE</td>
</tr>
<tr>
<td>$ff$</td>
<td>false, FALSE</td>
</tr>
</tbody>
</table>

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\(^4\) 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/

\(^{4}\)http://www.rcsg.rice.edu/sharecore/bluebiou/
Table 2: Comparison results on the Schuppan-collected benchmarks for LTL satisfiability checking.

<table>
<thead>
<tr>
<th>Formula Type</th>
<th>pltl</th>
<th>TRP++</th>
<th>NuSMV -BDD</th>
<th>NuSMV -BMC</th>
<th>Aalta</th>
</tr>
</thead>
<tbody>
<tr>
<td>/acacia</td>
<td>367.1</td>
<td>25.7</td>
<td>41.3</td>
<td>1.8</td>
<td>506.8</td>
</tr>
<tr>
<td>/alaska</td>
<td>5800.6</td>
<td>12641.5</td>
<td>7259.1</td>
<td>2797.2</td>
<td>3954.2</td>
</tr>
<tr>
<td>/anzu</td>
<td>3815.1</td>
<td>10729.7</td>
<td>12124.0</td>
<td>1093.2</td>
<td>5202.4</td>
</tr>
<tr>
<td>/rozier</td>
<td>1794.8</td>
<td>53234.0</td>
<td>15224.9</td>
<td>1067.1</td>
<td>3135.5</td>
</tr>
<tr>
<td>/schuppan</td>
<td>3079.8</td>
<td>4149.4</td>
<td>3838.8</td>
<td>4329.1</td>
<td>97.4</td>
</tr>
<tr>
<td>/trp</td>
<td>27475.5</td>
<td>3898.0</td>
<td>34533.8</td>
<td>23849.6</td>
<td>4392.2</td>
</tr>
<tr>
<td>Total</td>
<td>44576.2</td>
<td>84826.2</td>
<td>73030.5</td>
<td>44250.0</td>
<td>20626.4</td>
</tr>
</tbody>
</table>

Table 3: Experimental results on Schuppan-collected formulas for LTL\textsubscript{f} satisfiability checking.

<table>
<thead>
<tr>
<th>Formula Type</th>
<th>Aalta</th>
<th>Polsat</th>
</tr>
</thead>
<tbody>
<tr>
<td>/acacia</td>
<td>4.9</td>
<td>609.3</td>
</tr>
<tr>
<td>/alaska</td>
<td>24.2</td>
<td>7326.9</td>
</tr>
<tr>
<td>/anzu</td>
<td>5727.8</td>
<td>5770.8</td>
</tr>
<tr>
<td>/rozier</td>
<td>2416.1</td>
<td>3526.2</td>
</tr>
<tr>
<td>/schuppan</td>
<td>232.3</td>
<td>1874.8</td>
</tr>
<tr>
<td>/trp</td>
<td>6838.4</td>
<td>30392.7</td>
</tr>
<tr>
<td>Total</td>
<td>15244.2</td>
<td>50038.2</td>
</tr>
</tbody>
</table>

We recall that usual approach to LTL satisfiability checking over finite trace (LTL\textsubscript{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\textsubscript{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. Gegguang 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. 6136136002 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


