A Practical Transformation from LTL to Automata *

Cong Tian and Zhenhua Duan*

ICTT and ISN Lab, Xidian University, Xi' an, 710071, P.R. China

Abstract. A new linear transformation from PLTL formulas to alternating automata is proposed in this paper. To this end, C-F normal forms and Normal Form Graphs (NFGs) are defined for PLTL formulas. Further, based on them, Generalized Alternating Biichi Automata (GABA) of PLTL formulas are built. In addition to the conciseness in theoretical aspect, the new transformation is useful in improving the scalability of LTL model checking tools in practise.

1 Introduction

Model checking is an important approach for the verification of hardware, softwares, multi-agent systems, communication protocols, embedded systems and so forth. In the last two decades, several model checkers such as SPIN and SMV were developed with success. It is fair to say that automated verification is one of the most successful applications of automated reasoning in computer science [1].

Generally, to check the linear-time properties of systems, Linear Temporal Logic (LTL) plays important roles for the specification purpose. The automata-theoretic approach offers a uniform algorithmic framework for model checking Propositional LTL (PLTL). The seminal theory of this approach was worked out in the 80s and the basic algorithms were developed during the 90s. Transforming from PLTL formulas to automata is a key step in PLTL model checking. Originally, the transformation [2, 3] was on mathematical simplicity, it was not appropriate for explicit model checking, since the automata constructed were always exponential in the size of the formula. On a demand-driven basis, an optimized translation [4] that avoided the exponential blow-up in many cases of practical interest was developed and used in the explicit model checker SPIN. Based on the optimization on the original transformation [5], symbolic model checkers such as NuSMV were developed. Also, a transformation via alternating automata was described in [6] motivated by mathematical simplicity [1].

As model-checking pervading into the industrial applications, efficient and applicable model checking algorithms are appealed. Thus, several improvements have been given to get a better translation from PLTL to automata during the last few years [7-12]. However, most LTL translation tools are research prototypes and cannot be considered industrial quality tools. Factually, when dealing with complex formulas, most of the translators do not work. Motivated by this, a new linear transformation from PLTL formulas to alternating automata is proposed in this paper.

* This research is supported by the NSFC Grant No. 61003078, 61133001 and 60910004, 973 Program Grant No. 2010CB328102 and ISN Lab Grant No. ISN1102001. t Corresponding author.

The paper is organized as follows. The next section briefly presents the preliminaries concerning PLTL and Alternating Biichi Automata (ABA). In section 3, C-F normal forms are defined for PLTL formulas. Based on it, a transformation form PLTL formulas to ABAs is presented.

2 Preliminaries

2.1 Propositional Linear Temporal Logic

We use the terminology from [18]. Given a nonempty finite set **AP** of atomic propositions, the set of all PLTL formulas over AP is the set of formulas built from elements from **AP** using negation H, disjunction (V), next operator (0), and the until operator (U). The abbreviations *true, false,* A, —, and <--> are defined as usual. In particular, *true* $\overset{def}{=}$ *0 v -,0* and *false* $\overset{def}{=}$ *0* A —10 for any formula *0.* In addition, eventually (0) and always (0) temporal operators can be derived by *00* ^{d}g *true* **U** *0* and 00 ^{ci}g ¯10-10 respectively.

It has been proved that any PLTL formula can be transformed into a negation normal form where all negations are adjacent to atomic propositions by employing an auxiliary temporal operator **U.** Negation normal form of PLTL is defined in Definition 1.

Definition 1 (Negation Normal Form) For *P* E **AP,** the set of PLTL formula in negation normal form is defined by: 0::=PHP10v010A01 *0010U010* 0 *0,* where -,(0 U *0)* **E (-10** $_{U}$ **-10),** and -,*(0* $_{U}$ *0)* **E (¯10U¯'0).** 0

The worst-case time complexity of transforming a PLTL formula into negation normal form is linear in the length of *0,* denoted by 101.

Similar to the typical propositional logic, we also define disjunction normal forms for PLTL formulas by treating formulas without 'v' and 'A' being the main operators as literals. Formally, *P, -4*^{3}, where *P* E **AP,** are literals; and *00, 0* **U** *fp* as well as *0* U *yo,* where *0* and *co* are PLTL formulas, are literals.

Definition 2 (Disjunction Normal Form, DNF) A PLTL formula is in disjunction normal form iff it is a disjunction of some conjuncts of some literals. **o**

It is obvious that any PLTL formula can be transformed into its disjunction normal form. For convenience, D(0) is used to denote the DNF of PLTL formula 0.

2.2 Alternating Biichi Automata

Within traditional automata theoretics, nondeterminism gives a computing device the power of existential choice. Its dual gives a computing device the power of universal choice. Motivated by this, alternating automata where both existential choice and universal choice are permitted are proposed [13, 14]. To introduce alternation transitions into automata, positive Boolean formulas built form set of states are useful. For a given set *X* of states, 1:)+(X) means the set of positive Boolean formulas over *X* (i.e., Boolean formulas built from elements in *X* using A and v). We say that Y c *X* satisfies a formula *0* E I *(X)* if the truth assignment that assigns *true* to the members of

Y and assigns *false* to the members of *X — Y* satisfies *O*. For example, the set {s1, s3} and *(Si,* s4} both satisfy the formula si A S3 V s3 A 54, while the set {s1, s4} does not. Accordingly, alternating Biichi automata are formally defined bellow [15].

Definition 3 An Alternating Biichi Automaton (ABA) is a tuple A = *(E, Q, qo, 6, F),* where *E* is a finite alphabet, *Q* is a finite set of states, qo E *Q* is the initial state, *F* E 2Q is a set of finial states sets, and *6 : Q x E —> 1.±(Q)* is a transition function that maps a state and an input letter to a positive boolean combination of states. ❑

Because of the universal choice in alternating transitions, a run of an alternating automaton is a tree rather than a sequence. A run of an ABA A on an infinite word w = so, si, s2, ... is an Q-labeled tree *r* such that *r(E) = qo,* and for each *i,* if lx1 = *r(x) = q,* and *6(q,* s_i*) = 6,* then x has *k* children x1, *xk,* for some *k* IQI, and tr(xi), *r(xk)}* satisfies *0.* A run tree is accepting iff all paths in the run tree satisfy the Biichi condition that there exist infinite many positions on the path being labeled with final state.

Considering that Generalized Biichi Automata (GBA) are often immediate results of the transformations from temporal logics to automata, we also present its alternating version.

Definition 4 A Generalized Alternating Bfichi automaton (GABA) is a tuple *GA = (E, Q,* q_o, *6, F = {F1, ...,* F,}), where *E* is a finite alphabet, *Q* is a finite set of states, q_o E *Q* is the initial state, *F* is a set of accepting sets 1F1, F_n} for *n* > 0 with *F,* c *Q,* ancIS : *Q* x —> I:±(Q) is a transition function that maps a state and an input letter to a positive boolean combination of states. ❑

A run of a GABA *GA* on an infinite word w = so, Si, S2, ... is an Q-labeled tree *r* such that *r(E)* = qo, and for each *i,* if Ix' = i, *r(x) = q,* and *6(q,* s_i*) = 0,* then *x* has *k* children xi, ..., *xk,* for some *k* IQI, and *{r(xi), r(xk)}* satisfies *19.* A run tree is accepting iff for each acceptance set F_i E *F,* there exists a state in *F,* that appears in each path of the run tree infinitely often. A word w is accepting if there exist a run tree on w which is acceptable in *GA.*

3 From PLTL Formulas to GABAs

In this section, we show how PLTL formulas can be equivalently transformed to GABAs by the usage of C-F normal forms.

3.1 CF-Normal Forms for PLTL

In this section, when concerning a PLTL formula, it indicates a PLTL formula in negation normal form. In what follows, CF-normal form (namely current and future normal form) is defined for PLTL formulas. Originally, this normal form was used in [16, 19] for dealing with Projection Temporal Logic (PTL).

Let *Atom()* be the set of atomic propositions occuring in *0, lAtom(0)1 = n,* **V** be the set of all possible valuations *Atom(0) —> {true, false},* and for each v E V, let fi_v be the formula a1 A ... A a_n where for each i, 1 < *i < n,* a_i a P_i if *v(P,) = true* and *a,* a if *v(Pi) = false,* where *P,* *E Atom().*

Definition 5 The CF-normal form of PLTL formula 0 can be defined by, 0 V */3* A
veV
$0^{11)}(^{4}0fv)$, where *f3*, is different from each other; and each *Of*, is an arbitrary PLTL formula which is rewritten into disjunction normal form by *D(Ofv).* ❑

In the normal form, each *$f3_v$* A OD($0f_v$), is called a basic component of the normal form. In each basic component, *fly* indicates the assignment of each proposition in *Atom()* at the current state, while D(0 f,,) denotes that at the next state D(Of_v) holds. Thus, CF-normal form splits a PLTL formula into current parts and future parts. The following theorem shows that any PLTL formula can be rewritten into its CF-normal form.

Theorem 1 Any PLTL formula *0* can be rewritten into its CF-normal form. 0

The proof provides an approach for rewriting a PLTL formula into its CF-normal form and can be found in the Appendix.

3.2 From PLTL Formulas to NFGs

Roughly speaking, Normal Form Graph (NFG) of a PLTL formula is constructed by repeatedly decomposing the PLTL formula and the new generated formulas to the current and next states according to C-F normal forms. The general idea for constructing NFGs is as bellow. To construct NFG of 0, initially, a root node Lo j is created. Then we rewrite 0 to its normal form. Let *0EpArn* A, 2) V *p* A A 0(0_3 *V* 54 A 05). As illustrated in Fig.l, five new nodes: LO1J, LO2J, 1_031 LO4J and L05] are created; and

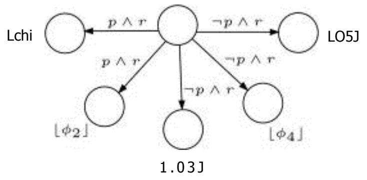

Fig.1. Constructing NFGs

the following relations between the new created nodes and the old ones are produced: (1) *Tao j, p* A = A L02], (2) T(L0J, A *r) = LO3J V* 1_04] A 1_05_1• Then to construct the whole NFG, L01_1, LO2J, LO3J, LP4J and LO5J need to be treated in a similar way. Note that when creating a node koj, the new node will be added if node koj does not already exist, otherwise only an edge back to the existing node koj is added. In the following, formal definition of NFG is presented.

Definition 6 (Normal Form Graph, NFG) For a PLTL formula 0, NFG of 0 is a directed graph, *G = (S, so,* r), where *S* denotes a non-empty finite set of nodes with so E *S* being the initial node; *T : S —> ±(S)* is a transition function. Further, the set *S* of nodes and the transition function *T* over the nodes are inductively defined as follows:

1. s_o = {LOA and *S* = {LOJ1
2. For each [(pi E *S* , if *co* has not been decomposed, rewrite *co* into its C-F normal form *(p* = V (1^{3}1$^{, A}$ OD(C°fv)) with D(cofv) ^{E} V A then *S* = *S U* aqviii1, and
   vcvj
   z(L(PJ,fiv) = V A Lnvii_l for each *i, j* and v. ❑
   j

A run of an NFG *G* on an infinite word *A = qo,* gi, q2, ... is an *(S)-labeled* tree *r* such that *r(E) = so,* and for each *i,* if Ix' = i, *r(x) = s,* and *8(s, q$_i$) = 0,* then *x* has *k* children x1, *xk,* for some *k < IS* I, and *{r(xl), ..., r(xk)}* satisfies *O.*

Further, based on the definition of NFGs, NFGs of PLTL formulas can be recursively constructed.

Theorem 2 Let *S* be the set of nodes in the NFG of an arbitrary PLTL formula *0.* We have IS I < 2 x 101. ❑

3.3 From NFGs to GABAs

Since NFGs are precisely constructed according to the equivalent transformation of PLTL formulas into the current and next parts. It is easy to achieve that: in the NFG of formula 0, for a run tree *r* on the infinite word w = qo, gi , ..., if each until formula occurs in any path of the run tree is fulfilled (i.e. for *PUQ,* a state where *Q* is satisfied can be finitely found), then w *0;* and for an infinite word w *0,* a run tree *r,* where each until formula occurs in any path of the run tree is fulfilled, on w can be found in the NFG of 0.

Now we show how to identify whether or not the until formulas are fulfilled in the paths of the NFGs by generalized Biichi condition. Given the NFG *G = (S, so, 7)* of PLTL formula *0,* GABA *GA = (E, Q, q$_0$, 6, F)* of *0* is obtained by: E = AIVE VI; *Q = S,* and go = *so; S = T; F = }Q \ {141U0211* I 1_01UO2I E *Q}.* In the transformation, the set of the states and the transitions are identical to the ones in NFGs. For each node in the form of L(fri UO2_1, an acceptance set is defined that consists all nodes excepting for UO2J. As a special case, when there exist no nodes in the form of VI U021 *F* = tQl.

3.4 Satisfiability and Complexity

Based on the transformation from PLTL formulas to GABAs, a decision procedure for checking the satisfiability of a PLTL formula *0* is achieved by first constructing the GABA *GA* of *0,* and then checking the emptiness of *GA.* Further, by Theorem 2, the complexity of the transformation from PLTL formulas to GABAs is linear to the length of the formulas. And it has been known that the complexity for checking the emptyness of alternating Biichi automata is exponential to the size of the automata [15]. Thus, our decision procedure for checking the satisfiability of LTL formula is exponential which matches the lower bound result [20].

However,compared to the existing transformations from PLTL to automata, the new transformation is much more simple and intuitive. Moreover, since the transformation is mainly achieved by gradually decomposing the formulas to the current and the next states, a complete on-the-fly model checking approach can be obtained.

4 Conclusions

A new linear transformation from PLTL formulas to automata is given in this paper. The new transformation is much more simple and intuitive than the existing ones. In the near future, a supporting tool will be developed based on the new proposed methods. Also, we will take several case studies to examine our approach and tool.

References

1. Moshe Y. Vardi. *Automata-theoretic model checking revisited.* VMCAI 07. LNCS 4349. Springer, New York, pp 137C150.
2. M.Y. Vardi and P. Wolper. *Reasoning about infinite computations.* Information and Computation, 115(1):1C37, November 1994.
3. P.Wolper, M.Y. Vardi, and A.P. Sistla. *Reasoning about infinite computation paths.* In Proc. 24th IEEE Symp. on Foundations of Computer Science, pages 185C194, Tucson, 1983.
4. R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. *Simple on-the-fly automatic verification of linear temporal logic.* In P. Dembiski and M. Sredniawa, editors, Protocol Specification, Testing, and Verification, pages 3C18. Chapman & Hall, August 1995.
5. E.M. Clarke, 0. Grumberg, and K. Hamaguchi. *Another look at LTL model checking.* CAV94, pages 415-427, June 1994, Springer-Verlag.
6. M.Y. Vardi. *Nontraditional applications of automata theory.* International Symp. on Theoretical Aspects of Computer Software, LNCS 789, pages 575-597. Springer-Verlag, 1994.
7. N. Daniele, F Guinchiglia, and M.Y. Vardi. *Improved automata generation for linear temporal logic.* CAV99, LNCS 1633, pages 249C260. Springer-Verlag, 1999.
8. K. Etessami and G.J. Holzmann *Optimizing Bi,ichi automata.* In Proc. 11th Intl Conf. on Concurrency Theory, LNCS 1877, pages 153C167. Springer- Verlag, 2000.
9. C. Fritz. *Constructing Bi,ichi automata from linear temporal logic using simulation relations for alternating bchi automata.* ICIAA03, LNCS 2759, pages 35C48. Springer-Verlag, 2003.
10. C. Fritz. *Concepts of automata construction from LTL.* In Proc. ICLPAIR 2005, LNCS 3835, pages 728-742. Springer-Verlag, 2005.
11. F Somenzi and R. Bloem. *Efficient Biichi automata from LTL formulae.* CAVOO, volume 1855 of Lecture Notes in Computer Science, pages 248C263. Springer-Verlag, 2000.
12. X. Thirioux. *Simple and efficient translation from LTL formulas to Bi,ichi automata.* Electr. Notes Theor. Comput. Sci., 66(2), 2002.
13. J.A. Brzozowski and E. Leiss. *Finite automata, and sequential networks.* Theoretical Computer Science, 10:19C35, 1980.
14. A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. *Alternation.* Journal of the Association for Computing Machinery, 28(1):114C133, 1981.
15. Moshe Y. Vardi. *Alternating Automata and Program Verification.* In Computer Science Today. LNCS 1000, Springer-Verlag, pp.471-485, 1995.
16. Z.Duan, C.Tian and L.Zhang. A *Decision Procedure for Propositional Projection Temporal Logic with Infinite Models,* Acta Informatica, 45(1):43-78, 2008.
17. A.Pnueli. *The temporal logic of programs.* In Proc. 18th IEEE Symp. Found. of Comp. Sci.,pages 46-57,1977
18. Allen E.Emerson. *Temporal and modal logic,* in: Jan van Leeuwen, ed., Handbook of Theoretical Computer Science, volume B: Formal Methods and Semantics, 995-1072,1990.
19. Z.Duan. *An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming.* PhD thesis, University of Newcastle Upon Tyne, May 1996.
20. A.P.Sistla and E.M.Clarke. *The complexity of propositional linear temporal logics.* Journal of the ACM, Volume 32, Issue 3, Pages: 733 - 749, 1985.