Symbolic Algorithm for Generation Büchi Automata from LTL Formulas

Irina V. Shoshmina and Alexey B. Belyaev

Saint-Petersburg State Polytechnical University,
Distributed Computing and Networking Department, St. Petersburg, Russia
ishoshmina@dcn.ftk.spbstu.ru, belyaevab@gmail.com

Abstract. Model checking is a new techology developed to ensure the correctness of concurrent systems. In this paper we consider one of the algorithms included in this techology, an algorithm for constructing Büchi automaton from a given LTL formula. This algorithm uses an alternating automaton as an intermediate model while translating the LTL formula to a generalized Büchi automaton. We represent data structures and data manipulations with BDD to increase algorithm effectiveness. The algorithm is compared on time and resulting Büchi automaton size with well known LTL to Büchi realizations (SPIN, LTL2BA), and it shows its effectiveness for wide class of LTL formulas.

Keywords: concurrent system verification, model checking, LTL, BDD, alternating automaton, transition acceptance, Büchi automaton.

1 Introduction

The complexity of modern software constantly grows. As a consequence, the number of errors in programs grows too, especially in systems with parallel and distributed architecture.

It is well known that error detection in parallel, distributed, and multithreaded programs is not easy. Even when algorithms of each interacting process of parallel system are absolutely clear, it is difficult to understand the behavour of the entire system. While developing a parallel program a programmer should monitor the possible combinations of partially ordered events, which is much harder than to control completely ordered events in sequential programs. Parallel systems working correctly "almost always" may keep subtle errors over the years. Those errors may reveal in rare and critic situations. As a rule such errors cannot be found out by testing.

In recent years, a new approach to program verification, model checking, was developed. Model checking is a technology wich is the most effective for formal verification of parallel and distributed systems [7]. It provides a method to verify whether a given formula (usually a formula of temporal logic, in particular linear temporal logic — LTL) is true on a model of the system. A formula describes the desired requirements to the system behavior. Model checking algorithms carry

V. Malyshkin (Ed.): PaCT 2011, LNCS 6873, pp. 98–109, 2011.

out a full analasys of all possible system's runs. This method has great potential to increase the quality of distributed software systems.

One method of LTL model checking is based on a theoretical-automata approach. For this purpose, the LTL formula negation expressing a given system property is translated to a corespondent Büchi automaton. After that a parallel composition of this Büchi automaton and the system model represented by a Kripke structure is constructed. Our paper presents an effective algorithm of LTL formula to Büchi automaton translation.

2 Background

LTL formulas are built over a set AP of atomic propositions, logical connectives and temporal operators U, X. All LTL formulas are formed according to the following grammar:

$$\varphi ::= p|\neg\varphi|\varphi \wedge \varphi|\mathrm{X}\varphi|\varphi\mathrm{U}\varphi$$

where $p \in AP$. Deliverable temporal operators are $\mathrm{F}\varphi = True\mathrm{U}\varphi$ and $\mathrm{G}\varphi = \neg\mathrm{F}(\neg\varphi)$.

In general, the complexity of constructing Büchi automaton grows exponentially on the length of the LTL formula. Such complexity is not an obstacle for checking small LTL formulas. But in practice, there are cases when the size of LTL formulas are enormous. In particular it is the case when the property φ is verified according to some assumptions about the verified system behavior (so-called *fairness constraints*).

Here is an example of such property: "If infinitely often an alternator state will be consistent, than always by pressing the protection button the vessel power supply becomes active sometime in future". LTL formula which corresponds to this property is: $\mathrm{GF}p \rightarrow \mathrm{G}(q \rightarrow \mathrm{F}r)$, where p stands for "the alternator is in the consistent state", q stands for "the protection button is pressed", r stands for "the vessel power supply is active". It is obvious that a state of a vessel power supply system depends on many parametres (such as pressure in the diesel engine, diesel engine temperature etc.). So such kind of constraints are fairly easily expressed in LTL by adding additional subformulas, but the size of the whole LTL formula for this property is growing:

$$(\mathrm{GF}p_1 \wedge \ldots \wedge \mathrm{GF}p_k) \rightarrow \varphi$$

In SPIN, which is specially designed for verification of parallel algorithms and protocols, generation of Büchi automaton from such LTL formula for $n = 6$ takes about one hour and a half. And for $n = 9$ the translation algorithm used in SPIN could not construct Büchi automaton at all.

The aim of this paper is to develop an effective algorithm for generation of Büchi automaton from an LTL formula. We use alternating automaton as intermediate model while translating the LTL formula to a generalized Büchi automaton. The main stages of our algorithm are (a) translation of the given LTL formula to an alternating automaton, (b) constructing a generalized Büchi

automaton with transition acceptance and (c) generation ordinary Büchi automaton from generalized one. We present states and transitions of all intermediate automata, as well as logical operations over them as Boolean functions using Binary Decision Diagrams. The algorithm is compared on time and resulting Büchi automaton size with well known LTL to Büchi realizations (SPIN, LTL2BA), and it shows its effectiveness for wide class of LTL formulas. The theoretical basis of using alternating automata for LTL formula representation is developed in [2,3].

3 Alternating Automata on Infinite Words

Unlike finite nondeterministic ordinary automata, alternating automata have existential and universal choices of a set of states in transition function. An existential choice (OR choice) implies a non-deterministic transition into one of the possible states. A universal choice (AND choice) means that a transition occurs simultaneously into all states corresponding to this choice.

Definition 1. *An alternating Büchi automaton is a tuple* $A = (Q, \Sigma, q^0, \delta, F)$, *where* $Q = \{q_0, q_1, \cdots, q_n\}$ *is a finite nonempty set of states,* Σ *is a finite nonempty input alphabet,* $q^0 \in Q$ *is an initial state,* $F \subseteq Q$ *is a set of accepting states, and* $\delta : Q \times \Sigma \to L(Q)$ *is a transition function,* $L(Q)$ *is a free distributive lattice generated by* Q.

$L(Q)$ has two binary operations: $\wedge$ for universal choice, and $\vee$ for existential choice. The operations satisfy the usual laws: commutativity, associativity, distributivity, idempotency, and merging.

$L(Q)$ may be represented by a set of positive Boolean formulas over S (without negation) $B^+(S)$ [1], if each state q_i is accociated with a propositional variable s_i:

$$s_i = \begin{cases} 1 \text{ , } q_i \in \tilde{Q}, \text{ where } \tilde{Q} \text{ is a given subset of } Q \\ 0 \text{ , else} \end{cases} \tag{1}$$

Henceforth we use propositional variables s instead of states q.

Positive Boolean formulas express universal and existential choices combinations unambiguously. If a transition $\delta(s, a)$ is nonempty then the automaton accepts a being in the state s. Transition function $\delta(s, a) = s_1 \vee (s_2 \wedge s_3)$ means that being in the state s the automaton accepts a word aw, if it accepts the word w from the state s_1 or from both s_2 and s_3.

Existential and universal choices of alternating automata transitions has the only interpretation in the disjunctive normal form over positive Boolean formulas (PDNF).

The disjunctive normal form over positive Boolean formulas (PDNF) is used for the only interpretation of existential and universal choices of alternating automata. In PDNF a term C is a conjunction of formulas $B^+(S) : C = \wedge_k s_k$, where no s_k occurs more than once. Each element $e \in B^+(S)$ has a unique

representation in a disjunctive normal form (up to the order of terms), $e = \vee_i C_i$, where no term C_i subsumes a C_j, $i \neq j$. It could be rewritten as $e = \vee_i \wedge_{k_i} s_{k_i}$.

A run of an alternating automaton is a tree rather than a sequence as it is for a nondeterministic Büchi automaton. Let β is an infinite branch of a run σ. A set of states occurring infinitely often in the branch β is $inf(\beta)$.

Definition 2 (Büchi acceptance condition). *An infinite branch β of a run σ is accepting, if $inf(\beta) \cap F \neq 0$. A run σ accepts an infinite word w, if any its infinite branch is accepting.*

4 Symbolic Realization of LTL to Büchi Automaton Translation

The main idea of the symbolic approach to algorithms processing finite data structures consists in using Boolean characteristic functions representing finite sets. A characteristic function defined on a subset $\tilde{A} \subseteq A$ of a finite set A is a Boolean function which indicates membership of an element in a subset $\tilde{A}$. All operations over finite sets are corresponding to Boolean operations over characteristic functions.

In our algorithm subsets of automaton states, transition functions, transition labels, labels of accepting states are all specified by Boolean characteristic functions. Alternating automaton definition is suitable for symbolic apporach. Alternating automaton states are encoded according to (1). Transition functions describing state sets in which transitions are carried out are presented by Boolean functions. Symbolic algorithms of main LTL to Büchi translation stages are stated in the next sections in detail.

In our algorithm we use Binary Decision Diagrams (BDD) as an effective form of Boolean functions representation. BDD is a directed acyclic binary graph without redundancy in its structure. More details about BDDs may be found, for example, in [7].

5 Generation an Alternating Automaton from LTL Formula

We construct an alternating automaton Büchi with input alphabet 2^{AP} for a given LTL formula φ over a set AP of atomic propositions, which accepts exactly all infinite words satisfying the formula and only them. Our algorithm is based on the theoretical background given in [2].

Theorem 1. *[2]. Given a LTL formula φ, one can build an alternating Büchi automaton $A_\varphi = (S, \Sigma, s_0, \delta, F)$, where $\Sigma = 2^{AP}$ and $|S|$ is in $O(|\varphi|)$, such that $L_\omega(A_\varphi)$ is exactly the set of computations satisfying the formula φ.*

All states of an alternating automaton A_φ are labeled by subformulas of the given LTL formula φ. Next states are obtained supplying transition rules recursively for every state. Transitions are labeled with elements of 2^{AP}.

As far as alternating automaton transitions are defined in positive Boolean functions, we transform a LTL formula into a canonical form, so-called *negation normal form* (NNF), where all negations are adjacent to atomic propositions: $\neg(\varphi \wedge \psi) = \neg\varphi \vee \neg\psi$, $\neg(\mathrm{X}\varphi) = \mathrm{X}\neg\varphi$, $\neg(\varphi \mathrm{U} \psi) = \neg\varphi \mathrm{R} \neg\psi$. The grammar of LTL formulas in NNF is the following:

$$\varphi ::= p | \neg p | \varphi \vee \varphi | \varphi \wedge \varphi | \mathrm{X}\varphi | \varphi \mathrm{U} \varphi | \varphi \mathrm{R} \varphi$$

During syntactic analysis any subformula φ_i of the temporal formula φ is corresponded to a A_φ state s_i. So there is a function $\tau : \varphi \to S$ labeling states with subformulas such that $\tau(\varphi_i) = s_i$.

Example 1. Let's consider a part of the property given above: $\varphi = \mathrm{G}(q \to \mathrm{F}r)$. The NNF of this formula is the following: $\varphi = false\ \mathrm{R}\ (\neg q \vee (true\ \mathrm{U}\ r))$. As the result of syntactic analysis the following subformulas would be extracted: $\varphi_1 = r$, $\varphi_2 = true \mathrm{U} \varphi_1$, $\varphi_3 = \neg q$, $\varphi_4 = \varphi_3 \vee \varphi_2$, $\varphi_5 = \varphi = false \mathrm{R} \varphi_4$, and constructed propositional variables $s_1, \ldots, s_5$ corresponding to states of an alternating automaton B_φ.

Let $\Sigma = 2^{AP}$. Transitions rules of automaton A_φ are defined for logical connectives and temporal operators. For example, a state corresponding to a proposition $p \in AP$ accepts a symbol $a \in \Sigma$ if p is included in this set:

$$\delta\big(\tau(p), a\big) = true, \text{if } p \in a \tag{2}$$

$$\delta\big(\tau(p), a\big) = false, \text{if } p \notin a \tag{3}$$

The transiton from a state with a proposition negation is defined similarly.

A transition function for a state labeled with a disjunction of formulas φ and ψ define a set of states in which transitions on a from φ or from ψ are defined:

$$\delta\big(\tau(\varphi \vee \psi), a\big) = \delta\big(\tau(\varphi), a\big) \vee \delta\big(\tau(\psi), a\big) \tag{4}$$

A state corresponding to a temporal formula $\varphi \mathrm{U} \psi$ accepts a if a is accepted by a set of states in which a transition from ψ exists or, otherwise, a transition from a set of states φ and $\varphi \mathrm{U} \psi$ exists, because in this case Until obligation is not realized:

$$\delta\big(\tau(\varphi \mathrm{U} \psi), a\big) = \delta\big(\tau(\psi), a\big) \vee \big(\delta\big(\tau(\varphi), a\big) \wedge \tau(\varphi \mathrm{U} \psi)\big) \tag{5}$$

This definition corresponds to a recursive formula for $Until$: $\varphi \mathrm{U} \psi = \psi \vee \varphi \wedge X(\varphi \mathrm{U} \psi)$.

Similarly, transition functions for $\varphi \wedge \psi$, $\varphi \mathrm{R} \psi$, $\mathrm{X}\varphi$ are defined as follows:

$$\delta\big(\tau(\varphi \wedge \psi), a\big) = \delta\big(\tau(\varphi), a\big) \wedge \delta\big(\tau(\psi), a\big) \tag{6}$$

$$\delta\big(\tau(\varphi \mathrm{R} \psi), a\big) = \delta\big(\tau(\psi), a\big) \wedge (\delta\big(\tau(\varphi), a\big) \vee \tau(\varphi \mathrm{R} \psi)) \tag{7}$$

$$\delta\big(\tau(\mathrm{X}\varphi), a\big) = \tau(\varphi) \tag{8}$$

Joining all transitions from a state s over Σ we get a function $\delta(s) = \vee_{a \in \Sigma}$ $\delta(s, a)$. If $s = \tau(p)$ then $\delta(s) = \delta(\tau(p)) = p$. The change of rules (4-8) is obvious,

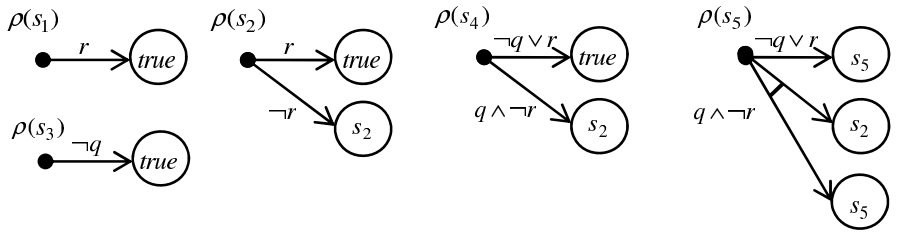

Fig. 1. Transition functions for B_φ constructed from $\mathrm{G}(q \rightarrow \mathrm{F}r)$

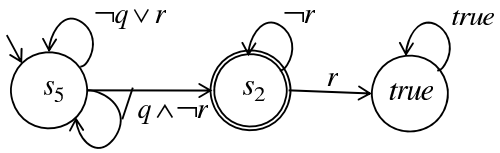

Fig. 2. The alternating automaton of LTL formula $\mathrm{G}(q \rightarrow \mathrm{F}r)$

for example, $\delta\big(\tau(\varphi \mathrm{U} \psi)\big) = \delta\big(\tau(\psi)\big) \vee \big(\delta\big(\tau(\varphi)\big) \wedge \tau(\varphi \mathrm{U} \psi)\big)$. Transition functions obtained from the rules (2-8) for B_φ states (Example 1) are given on the Fig.1.

As a result of specifying the transition functions described above we construct a very weak alternating automata (VWAA) [6]. This VWAA contains states labeled by those subformulas of φ which includes temporal operators, and those which labeled with *true*, and φ.

Definition 3. *An alternating automaton is called very weak alternating automaton, if a requirement of the partial order on S is added to the Def. 1.*

As a consequence of Def. 3 there are no cycles formed by transitions between different states in VWAA.

Definition 4 (co-Büchi acceptance condition). *A run of A_φ σ is accepting if any infinite branch in σ has only a finite number of nodes labeled by states from F (in terms of Def. 2 if $inf(\beta) \cap F = 0$).*

For VWAA Büchi and co-Büchi acceptance conditions are equivalent up to redefinition of F. The final states for the co-Büchi condition are labeled with formulas with the temporal operator *Until*. The resulting VWAA B_φ (Example 1) is shown on the Fig. 2, where the accepting co-Büchi state is labeled by a bold line.

6 From Alternating Automata to Generalized Büchi Automata with Transition-Based Acceptance

The next step of the algorithm is constructing a generalized Büchi automaton from the derived alternating one.

Definition 5. *A generalized Büchi automaton is a tuple* $GBA = (Z, \Sigma, z^0, \delta', T)$, *where:*

- Z *is a finite set of states,*
- Σ *is an finite alphabet,*
- $z^0 \in Z$ *is an initial state,*
- $\delta' : Z \to \Sigma \times Z$ *is a transition function,*
- $T = \{T_1, \cdots, T_r\}$ *is a set of accepting transitions.*

At this step we developed a symbolic algorithm based on the algorithm proposed and proved in [3]. Construction of GBA from an alternating automaton is similar to the classical algorithm for constructing a finite deterministic automaton from a finite nondeterministic one.

Each GBA state z is a product of VWAA states, so $Z \subseteq 2^S$. Its transition function is a product of corresponding VWAA transitions. The initial state of the GBA coincides with the initial VWAA state: $z^0 = s^0$.

Since F is the set of VWAA co-Büchi accepting states then a set of GBA accepting transitions groups are defined as following:

$$T = \{T_k \mid f_k \in F, 1 \leq k \leq r\}, \tag{9}$$

where

$$\begin{aligned} T_k = \{(z, a, z') \in Z \times \Sigma \times Z \mid f_k \notin z' \\ \vee \exists (b, z'') \in \delta(f_k) : (a \subseteq b \wedge f_k \notin z'' \wedge z'' \subseteq z')\}. \end{aligned} \tag{10}$$

The symbolic algorithm constructing the set Δ of GBA transitions is presented on Fig. 3. Consider some algorithm steps in detail.

Products of VWAA states forming GBA states are obtained from unique conjunctions entering into the disjunctive normal form of transition functions, starting from the initial state. To obtain these conjunctions from ordinary Boolean functions we add few operations.

A solution of a Boolean function $\delta(s)$ is a vector:

$$\xi = \big(\xi(p_1), \ldots, \xi(p_m); \xi(s_1), \ldots, \xi(s_n)\big),$$

where $\xi(x)$ is an assignment of a Boolean variable x, $p_i \in AP$, $s_i \in S$.

Consider a set of solutions $\zeta(\delta)$ of a transition function $\delta(s)$, such that:

$$\zeta(\delta; x) = \begin{cases} 0 & , \forall \xi \in \zeta : \xi(x) = 0, \\ 1 & , \forall \xi \in \zeta : \xi(x) = 1, \\ -1 & , \exists \xi_1 : \exists \xi_2 : \xi_1(x) = 1 \text{ and } \xi_2(x) = 0, \xi_1 \neq \xi_2 \end{cases} \tag{11}$$

Using sets of solutions $\zeta(\delta)$ a GBA transition function $\delta'(s)$ is calculated as:

$$\wedge_{i, \zeta(\delta, s_i)=1} \, \delta(s_i). \tag{12}$$

Constructed transitions functions (12) may contain redundant transitions. For example, for a transition function like $s_1 \vee s_2$ the disjunctive normal form over

ordinary Boolean functions would be $s_1 \wedge \neg s_2 \vee \neg s_1 \wedge s_2 \vee s_1 \wedge s_2$. However, the use of a disjunctive normal form over positive Boolean functions assumes that there is a transition in one of states s_1 or s_2 only. After removing redundant transitions on steps 19-21 (Fig. 3) the *reverse* function is used to obtain GBA transitions finally:

$$reverse\big(\zeta\big) = \bigwedge_{i=1}^{m} \rho\big(\zeta(\delta; p_i)\big) \wedge \bigwedge_{i=1}^{n} \rho\big(\zeta(\delta; s_i)\big),$$

where

$$\rho\big(\zeta(\delta; x)\big) = \begin{cases} \neg x \ , \zeta(\delta; x) = 0, \\ x \quad , \zeta(\delta; x) = 1, \\ 1 \quad , \zeta(\delta; x) = -1 \end{cases}$$

```
buildGBA(δ(z^0))      // δ(z^0) = δ(s^0) — transition function of initial state of GBA
1     δ_0 ← δ(z^0)
2     Δ' ← {δ}
3     foreach ζ(δ_0) :
4           δ_1 ← ∧_{i,ζ(δ_0;s_i)=1} δ(s_i)
5           Δ' ← Δ' ∪ {δ_1}
6     old_size ← 0
7     new_size ← |Δ'|
8     while new_size ≠ old_size
9         old_size ← new_size
10          foreach δ_0 ∈ Δ'
11                similar steps 3-5
14          new_size ← |Δ'|
15    Δ ← {}
16    foreach δ_0 ∈ Δ' // remove redundant transitions
17          δ_1 ← false
18          foreach ζ(δ_0) :
19                for i ← 1 to n
20                      if ζ(δ_0; s_i) = −1 then
21                            ζ(δ_0; s_i) ← 0
22                δ_1 ← δ_1 ∨ reverse(ζ(δ_0))
23          Δ ← Δ ∪ {δ_1}
```

Fig. 3. Algorithm for constructing the generalized Büchi automaton from a given VWAA

Example 2. The initial state of the alternating automaton B_φ is s_5 (Fig. 2), so the initial state of the corresponding GBA is $z_0 = s_5$. According to $\delta(s_5)$ there are a transition to s_5 and a transition with universal choice to s_2 and s_5, so $z_1 = s_5 \wedge s_2$. There are no new states occurring from the conjunction of transitions s_2 and s_5. The GBA constructed by our symbolic algorithm is shown in Fig. 4.

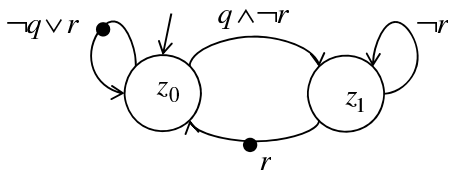

Fig. 4. The generalized Büchi automaton with accepting transitions for the formula $\mathrm{G}(q \rightarrow \mathrm{F}r)$

According to the conditions (9-10) accepting transition labeles are defined as:

$$\lambda_k = \neg f_k \vee \exists f_k : (\delta(f_k) \wedge \neg f_k) \tag{13}$$

We use these labels to mark degeneralizer transitions in the algorithm of degeneralizer construction at the next step.

7 From Generalized Büchi Automata with Accepting Transitions to Büchi Automata

A Büchi automaton is built as a product of GBA with accepting transitions and an automaton-template, so-called a *degeneralizer* [5]. The number of degeneralizer states depends linearly on the number of GBA accepting labels. The structure of the degeneralizer guarantees that the word is accepting by automata composition if and only if transitions from every GBA accepting group are used. The degeneralizer is used to transfer accepting labels from transitions to states so that there was only one group of accepting labels in the final Büchi automaton. We use the algorithm of degeneralizer construction given in [5] and proved in [3].

The Büchi automaton for the formula $\mathrm{G}(q \rightarrow \mathrm{F}r)$ coincides with GBA where the state z_0 is accepting (Fig. 4).

8 Results and Related Works

In spite of the fact that in the worst case the number of states of a Büchi automaton is growing exponentially from the LTL formula length, in many cases algorithms based on alternating automaton lead to a very compact Büchi automaton.

For example, the LTL to Büchi algorithm based on atoms (sets of LTL subformulas) and obligations is desrcibed in [7]. For the formula $(p \vee q)\mathrm{U}(p \wedge q)$ the resulting Büchi automaton generated according to the algorithmin [7] has 6 states, while for our algorithm it has only 2 states (Fig. 5).

The idea of using the alternating automata as an intermediate step of building Büchi automata for LTL formula is presented in some another

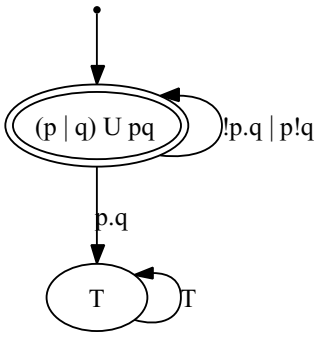

Fig. 5. The Büchi automaton constucted by symbolic algorithm for LTL formula $(p \vee q)\mathrm{U}(p \wedge q)$

algorithms [3,4]. The LTL2BA algorithm [3] use explicit presentation of states and transitions of VWAA, GBA and Büchi automata in memory. It uses various rules to minimize the automata by merging the equivalent states and removing the redundant transitions on-the-fly on each step. Also it provides a posteriori Büchi automata simplification. In our symbolic algorithm these simplifications are reached automatically thank to the BDD representation of Boolean characteristic functions. The paper [4] presents a generalized definition of alternating automata with fin- and inf-accepting conditions, which requires new rules for on-the-fly simplification.

Our realization is written in C++ using BuDDy v2.4 library [11] used for operations over BDD. We compare our realization with the Büchi generator of Spin model checker [8], one of the most popular software for LTL verification, developed by Bell Labs, and the LTL2BA program, which realizes an explicit version of the algorithm based on alternating automata described in [3]. Those both programs are written in C too. All tests were done on Intel Core 2 Duo CPU (2.33 GHz) with 2 GB of RAM.

Consider a formula $\mathrm{G}(q \rightarrow \mathrm{F}r)$, which means that a request q always leads a response r in the future. This kind of formulas refining with fairness conditions are often encountered in practice:

$$\Phi_n = \neg\big((\mathrm{GF}p_1 \wedge ... \wedge \mathrm{GF}p_n) \rightarrow \mathrm{G}(q \rightarrow \mathrm{F}r)\big).$$

The experiments results are presented in Table 1. It is practically impossible to use Spin to generate Büchi automata for this kind of formulas with more than four fairness conditions. Moreover, the number of states and transitions of Büchi automata generated by Spin is significantly greater than automata generated with algorithms based on alternating automata. The LTL2BA program reaches unreasonable time of work after nine fairness conditions.

Table 1. Time results for the formula Φ_n for $1 \leq n \leq 11$, time is in *sec*

n	Spin	LTL2BA	Symbolic
1	0.05	< 0.01	< 0.01
2	0.19	< 0.01	< 0.01
3	4	< 0.01	< 0.01
4	155	< 0.01	< 0.01
5	4607	0.05	0.03
6	5232	0.57	0.11
7	8113	4	0.33
8	11212	45	2
9	+	375	11
10	+	4500	16
11	+	> 36000	36

The same results has been shown by tests with the following kind of formulas (Table 2):

$$\Psi_n = \neg\big(p_1 \mathrm{U}(p_2 \mathrm{U}(...\mathrm{U}p_n)...\big).$$

Table 2. Time results in *sec* on the formula Ψ_n for $2 \leq n \leq 10$

n	Spin	LTL2BA	Symbolic
2	0.02	< 0.01	< 0.01
3	0.03	< 0.01	< 0.01
4	0.17	< 0.01	< 0.01
5	1.23	< 0.01	< 0.01
6	38	0.02	< 0.01
7	127	1.15	0.02
8	+	150	0.03
9	+	> 3600	0.17
10	+	+	0.63

The Büchi generator of the Spin model checker fails out of memory on formula Ψ_8.

The paper [3] presents the results of comparison LTL2BA program with algorithms LTL2AUT [9] and Wring [10], which improve Spin's algorithm. Spin and LTL2AUT use the optimization ideas: to generate states by demand only and use state labels rather than transition labels. In addition Wring simplifies formulas before translation using NNF. LTL2BA has shown the best results in described comparison. According to our experiments (Tables 1 and 2) the symbolic algorithm works more than in a thousand times faster than LTL2BA for some n.

9 Conclusion

In recent years, the model checking as the method for improving the quality of parallel and distributed programs is actively developed to find more effective algorithms and to apply it to real practical software systems. In this paper we consider one of the algorithms included in the LTL model checking.

The theoretical basis for using alternating automata for translation of linear time logic (LTL) formulas in Büchi automata is given in [1,2,3]. The advantage of this approach stems from the fact that in most cases the resulting Büchi automaton is rather compact. However, the translation algorithm itself poses exponential requirements on processing time and memory.

We used binary decision diagrams (BDD) to represent all data structures and all operations performed on them as binary functions. This representation led to significantly reduced complexity of the process of Büchi automaton construction for some types of LTL formulas.

References

1. Muller, D., Schupp, P.E.: Alternating Automata on Infinite Objects, Determinacy and Rabin's Theorem. In: Proc. Automata on Infinite Words – École de Printemps d'Informatique Théorique (EPIT 1984). LNCS, vol. 192, pp. 99–107 (1985)
2. Vardi, M.: Nontraditional Applications of Automata Theory. In: Hagiya, M., Mitchell, J.C. (eds.) TACS 1994. LNCS, vol. 789, pp. 575–597. Springer, Heidelberg (1994)
3. Gastin, P., Oddoux, D.: Fast LTL to Büchi Automata Translation. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 53–65. Springer, Heidelberg (2001)
4. Tauriainen, H.: Automata and Linear Temporal Logic: Translations with Trace-Based Acceptance. PhD.Thesis, Helsinki University of Technology (2006)
5. Giannakopoulou, D., Lerda, F.: From States to Transitions: Improving Translation of LTL Formulae to Büchi Automata. In: Peled, D.A., Vardi, M.Y. (eds.) FORTE 2002. LNCS, vol. 2529, pp. 308–326. Springer, Heidelberg (2002)
6. Rohde, G.S.: Alternating Automata and the Temporal Logic of Ordinals. PhD. Thesis in Mathematics, University of Illinois at Urbana-Champaign (1997)
7. Karpov, Y.G.: Model Checking: Verification of Parallel and Distributed Program Systems, p. 560. SPb:BHV-Petersburg (2010) (in Russian)
8. Holzmann, G.: Spin Model Checker. The Primer and Reference Manual, p. 608. Addison-Wesley, Reading (2003)
9. Daniele, M., Giunchiglia, F., Vardi, M.: Improved automata generation for linear temporal logic. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 249–260. Springer, Heidelberg (1999)
10. Somenzi, F., Bloem, R.: Efficient Büchi Automata from LTL Formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, Springer, Heidelberg (2000)
11. The BuDDy Project, `http://sourceforge.net/projects/buddy/`