A Comprehensive Comparison of Temporal Formula to Automaton Translation Algorithms

Chang, Jinn-Shu
Adviser: Tsay, Yih-Kuen
Graduate Institute of Information Management
National Taiwan University

November 20, 2009

THESIS ABSTRACT
Graduate Institute of Information Management
National Taiwan University

Student: Chang, Jinn-Shu Month/Year: November, 2009
Advisor: Tsay, Yih-Kuen

A Comprehensive Comparison of Temporal Formula to Automaton Translation Algorithms

Translation of a temporal formula into an automaton is a central issue in the automata-based approach to model checking. In the approach, model checking of a system $\mathcal{M}$ against a temporal specification f proceeds in three steps: (1) generate an automaton $\mathcal{A}_{\neg f}$ for the negation of f, (2) construct a product automaton $\mathcal{A}$ that is the intersection of $\mathcal{M}$ and $\mathcal{A}_{\neg f}$, and (3) check the emptiness of the product automaton $\mathcal{A}$. The time needed to complete the model checking task is proportional to the size of $\mathcal{A}$, which is the product of the sizes of $\mathcal{M}$ and $\mathcal{A}_{\neg f}$. For a given system, the size of $\mathcal{A}_{\neg f}$ determines the size of $\mathcal{A}$. Therefore, the smaller $\mathcal{A}_{\neg f}$ is, the faster the model checking task may be carried out.

In this thesis, we investigate an extensive collection of translation algorithms, including all of the well-known ones. We compare the state and the transition sizes of the automata generated from these algorithms. An algorithm that generates smaller automata should be more helpful when it is applied in model checking. The reason is that when one needs to certify that a system satisfies the desired property, the complete product automaton must be constructed. To perform the comparison, we implement not only the translation algorithms but also possible improvements in the GOAL tool. From the experimental results, we observe that none of the algorithms can always generate the smallest automaton for each of the temporal formulae considered. We therefore propose a portfolio for choosing suitable algorithms for different kinds of temporal formulae. We also design and implement an open repository called Büchi Store where one can look up the Büchi automaton for a given temporal formula.

Keywords: Büchi Automata, GOAL, Model Checking, ω-Automata, Temporal Logic, Verification.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software verification is a fundamental issue about the correctness of programs. A program should always accomplish the goal the programmer proposes and should not cause any unexpected side-effect, which is known as *bugs* in the program. Bugs in a program should be eliminated otherwise software might go wrong. One common solution is to review the code before publishing. Yet it might costs lots of time and human work. Hence some systematical methods are proposed to clean up the bugs in program.

1.1 Background

There are some methods to seize this goal, such as testing and simulating in the early years. In 1981, *model checking* is first introduced by E. M. Clarke and E. A. Emerson. Model checking is an automatic process which can check whether the given system satisfies the given property.

Model checking is a procedure to solve the fundamental problem whether a given system $\mathcal{M}$ and a given property p, $\mathcal{M} \vDash p$. It involves three main phases, which are modeling, specification, and verification. In modeling, the system $\mathcal{M}$ is usually given as a specification of the target program. It can also be formalized by a finite state machine, which can be represented by Kripke Structure. Kripke Structure contains nodes to describe each state of a program and transitions for the variation of a state to another state for statements in the program. However, a Kripke structure can be transformed into an equivalent ω-automaton. Hence, we will only use the ω-automaton $\mathcal{A}_{\mathcal{M}}$ to describe the system we want to verify. In specification, temporal logic is used for describing the desired property p. Temporal logic is a logic language which wildly used to describe the rules and the desired property with temporal operators in terms of time. In this step, one can translate the property described in temporal logic into an equivalent automaton $\mathcal{A}_p$. Usually, Büchi automata are chosen to represent such an automaton. Büchi automata are first represented by J. R. Büchi in the 1960's, which is the first relating to ω-automata [2]. It is also proved in [2, 20] that each temporal logic formula can be translated into a corresponding automaton. In verification, we solve the model checking problem $\mathcal{M} \vDash p$ by solving this equivalent containment problem whether $\mathcal{L}(\mathcal{A}_{\mathcal{M}}) \subseteq \mathcal{L}(\mathcal{A}_p)$. This containment problem can be rewritten as an emptiness check problem, $\mathcal{L}(\mathcal{A}_{\mathcal{M}}) \cap \mathcal{L}(\mathcal{A}_{\neg p}) = \emptyset$.

There is a modified method based on model checking which is called "*on-the-fly model checking*". The main idea of on-the-fly model checking is that the model checker does not have to generate the complete product automaton for the emptiness test in verification phase. If the language of an automaton is not empty, the path of the accepting run may not involve all states in this automaton. With this concept, even if the complete automaton haven't be constructed, emptiness check of the half-built automaton is sufficient enough to tell whether the language of the complete automaton is empty. Since the production procedure is based on depth first search, whenever it creates a transition which towards a state which is already generated, it must be either a back edge or a cross edge in the complete graph. If this transition is a back edge and the states in this loop contains an accepting state, the emptiness check can be terminated and the counterexample is found. In this way, the cost on average is decreased even though the complexity is still remain costly.

In automata-approach model checking, the connection between program, temporal logic, and automata is quite essential.

1.2 Motivation and Objectives

Since we try to check the emptiness of the product of two ω-automata, the state space of the product automaton is also important. However, the size of $\mathcal{A}_{\mathcal{M}}$ is always large because it is highly related to the complexity of the system. Even though simulation algorithms such as the one described in [23] can be used, downsizing the system $\mathcal{M}$ is still limited. Thus, the size of the corresponding automaton for a property is important. Intuitively, we assume that the smaller the automaton is, the smaller the product automaton in the model checking procedure will be. Hence, the size of the generated automaton of a temporal formula to automaton translation algorithm is usually be think as a major point of view when comparing between translation algorithms. However, we should also concern about the size of the half-built product automaton when we talk about on-the-fly model checking. Our goal in this thesis is to comprehensively compare translation algorithms in more general way. This comparison will bring out the result about which kinds of translation algorithms can generate better automata, which is smaller or more suitable for on-the-fly model checking than the others.

For this goal, rather than studying the core idea these algorithms, we also need a good assistant tool for us to obtain needed data. Therefore, GOAL, which is an interactive graphical tool for temporal logic and various kinds of ω-automata, is introduced. We implement all the algorithms discussed in this thesis in the GOAL tool.

1.3 Thesis Outline

The rest of this thesis is organized as follows:

- In Chapter 2, we will introduce many kinds of automata which we will use in this thesis and relevant research such as ω-automata, transition-based automata, and alternating automata. Furthermore, the brief introduction about temporal logic would be described.

- In Chapter 3, we will introduce some translation algorithms which we had studied and some related tools that is used for model checking or linear temporal logic translation.

- In Chapter 4, we will present some translation algorithms which are based on transition-based generalized Büchi automata in detail. We will also present an algorithm which is based on a different idea than the others.

- In Chapter 5, we will show the experiment environment settings and the results. These experiments will bring out the discussions and conclusions in Chapter 6.

- In Chapter 6, we will discuss the reason why some algorithms can perform better than the others and how should we choose a right algorithm to combine with model checking.
- In Chapter 7, we briefly summarize this thesis and conclude our contributions. We would also describe some work we need to do in the future.

Chapter 2

Preliminaries

Here in this chapter, we will introduce some pre-knowledge for automaton-based model checking. In the first section, we will describe many kinds of automata which will be used in this thesis, and some basic operations for those automata. In the second section, the characteristic of temporal logic will be introduced.

2.1 Automata on Infinite Words

Automata theory is considered as a good way to understand a program, which is important in formal verification. ω-automata can represent not only a given system but also a given property which is written in temporal formula. This work can be traced back about forty years ago in 1960's, when J. R. Büchi introduced his work, which using finite automata with infinite input words to obtain a decision procedure for a restricted second-order logic, the sequential calculus [2].

Some notations in the following should be brought out here. Usually, we use Σ to denote the set of alphabet, and Σ^ω to denote the set of infinite words over Σ. An infinite word then can be denoted as $w = w_0w_1w_2\ldots, w \in \Sigma^\omega$ and each $w_i \in \Sigma$.

An ω-automaton $\mathcal{A}$ is a 5-tuple $(\Sigma, \mathcal{Q}, \Delta, \mathcal{Q}_0, \mathcal{F})$ where

- Σ is the finite set of symbols, called **alphabet**,
- $\mathcal{Q}$ is the finite set of **states**,
- Δ, is the **transition relation**,
- $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is the **initial states**, and

- $\mathcal{F}$ is the **acceptance component**.

If $|\Delta(s, \alpha)| = 1$ for $s \in \mathcal{Q}$ and $\alpha \in \Sigma$, the automaton is *deterministic*, otherwise, it is *non-deterministic*.

The acceptance conditions of each kinds of automata may be based on states or transitions. An ω-automaton is called a *state-based* automaton if the acceptance condition is defined over states. A run ρ of a state-based automaton $\mathcal{A}$ on infinite word $w = w_0w_1w_2 \ldots \in \Sigma^\omega$ is a sequence of states $q_0, q_1, \ldots \in \mathcal{Q}^\omega$ where

$$q_0 \subseteq \mathcal{Q}_0, q_i \in \mathcal{Q} \text{ and } q_{i+1} \subseteq \Delta(q_i, w_i) \text{ for } 0 \leq i.$$

The set of states occurring infinitely often in $\rho = q_0, q_1, \ldots \in \mathcal{Q}^\omega$ is denoted as $\mathit{inf}(\rho)$, more precisely

$$\mathit{inf}(\rho) = \{q_i \in Q \mid \forall i \exists j > i, q_i = q_j\}.$$

Usually, we simply say "automata" for "state-based automata". On the other hand, an ω-automaton is called a transition-based automaton if the acceptance condition is defined over transition. A run ρ of a transition-based automaton $\mathcal{A}_\mathcal{T}$ on infinite word $w = w_0w_1w_2 \ldots \in \Sigma^\omega$ is a sequence of transitions $(q_0, w_0, q_1), (q_1, w_1, q_2), \ldots \in \Delta^\omega$ where

$$q_0 \subseteq \mathcal{Q}_0, q_i \in \mathcal{Q} \text{ and } q_{i+1} = \Delta(q_i, w_i) \text{ for } 0 \leq i.$$

The set of transitions occurring infinitely often in $(q_0, w_0, q_1), (q_1, w_1, q_2), \ldots \in \Delta^\omega$ is denoted as $\mathit{inf}(\rho)$, more precisely

$$\mathit{inf}(\rho) = \{(q_i, w_i, q_{i+1}) \mid \forall i \exists j > i, (q_j, w_j, q_{j+1}) \text{ where } q_i = q_j, w_i = w_j + 1, q_{i+1} = q_{j+1}\}.$$

Note that $\mathit{inf}(\rho)$ is a set of transitions for a transition-based automaton, while it is a set of states for a state-based automaton. Because an accepting run of an ω-automaton is differ from the accepting conditions, the definition of accepting run of an automaton will be introduced in section 2.2.

An *alternating automaton* $\mathcal{A}$ is an ω-automaton with $\Delta = \mathcal{Q} \times \Sigma$ to $\mathcal{B}^+(\mathcal{Q})$ where $\mathcal{B}^+(\mathcal{Q})$ is the set of positive boolean formulae over $\mathcal{Q}$. A simple example of $\mathcal{B}^+(\mathcal{Q})$ is $q_0 \vee (q_1 \wedge q_2)$. A run ρ of $\mathcal{A}$ on a word $w_0w_1 \ldots \in \Sigma^\omega$ is a labeled DAG (V, E, λ) such that:

- V is partitioned into $\bigcup_{i=0}^{\infty} V_i$ (infinite levels of nodes),

- $E \subseteq \bigcup_{i=0}^{\infty} V_i \times V_{i+1}$,
- λ: $V \to \mathcal{Q}$ is the labeling function,
- $\lambda(V_0) \in \mathcal{Q}_0$, and
- for all $x \in V_i$, there exists a Q satisfying $\Delta(\lambda(x), w_i)$ such that $Q = \lambda(E(x))$, where $E(x) = \{q \mid (x, q) \in E\}$.

2.2 Variants of ω-automata

There are several kinds of ω-automata in automaton theory. In this thesis, we will use some of them, which are Büchi automata, generalized Büchi automata, transition-based generalized Büchi automata, and co-Büchi very weak alternating automata. We will give the definition for each of them in the following.

2.2.1 Büchi Automata

Büchi automata are often used for automata-based model checking. An ω-automaton $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, \mathcal{Q}_0, \mathcal{F})$ is called Büchi automaton if the acceptance condition is defined as follows: $\mathcal{F} \subseteq \mathcal{Q}$ and a run ρ on a infinite word w is accepted by $\mathcal{A}$ if

$$inf(\rho) \cap \mathcal{F} \neq \varnothing.$$

In other words, there exists at least one state $q \in \mathcal{F}$ which is visited infinitely often on ρ. A word $w \in \Sigma^{\omega}$ is accepted by $\mathcal{A}$ if there is a corresponding accepting run ρ.

When we talk about Büchi automaton, two basic operations, union and intersection, for it should be mentioned.

Proposition 2.1. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two Büchi automata. There is a Büchi automaton $\mathcal{A}$ which accepts the union language, which means $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$. [4]*

Proof. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be defined as follows:
$\mathcal{A}_1 = (\Sigma_1, \mathcal{Q}_1, \Delta_1, \mathcal{Q}_{01}, \mathcal{F}_1)$ and $\mathcal{A}_2 = (\Sigma_2, \mathcal{Q}_2, \Delta_2, \mathcal{Q}_{02}, \mathcal{F}_2)$. Let $\mathcal{A}$ is a 5-tuple $(\Sigma, \mathcal{Q}, \Delta, \mathcal{Q}_0, \mathcal{F})$, where

1. $\Sigma = \Sigma_1 \cup \Sigma_2$,
2. $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$,
3. $\mathcal{Q}_0 = \mathcal{Q}_{01} \cup \mathcal{Q}_{02}$,
4. $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$, and
5. $\Delta(q,w) = \begin{cases} \Delta_1(q,w) & \text{if } q \in \mathcal{Q}_1 \\ \Delta_2(q,w) & \text{if } q \in \mathcal{Q}_2 \end{cases}$

In this constructive way, it is easy to see that $\mathcal{A}$ accepts and only accept any accepting word for $\mathcal{A}_1$ and $\mathcal{A}_2$. □

Proposition 2.2. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two Büchi automata. There is a Büchi automaton $\mathcal{A}$ which accepts the intersected language, which means $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$[4].*

Proof. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be defined as follows:
$\mathcal{A}_1 = (\Sigma_1, \mathcal{Q}_1, \Delta_1, \mathcal{Q}_{01}, \mathcal{F}_1)$ and $\mathcal{A}_2 = (\Sigma_2, \mathcal{Q}_2, \Delta_2, \mathcal{Q}_{02}, \mathcal{F}_2)$. Let $\mathcal{A}$ is a 5-tuple $(\Sigma, \mathcal{Q}, \Delta, \mathcal{Q}_0, \mathcal{F})$, where

1. $\Sigma = \Sigma_1 \cup \Sigma_2$,
2. $\mathcal{Q} = \mathcal{Q}_1 \times \mathcal{Q}_2 \times \{1,2\}$,
3. $q_0 = \mathcal{Q}_{01} \times \mathcal{Q}_{02} \times \{1\}$,
4. $\mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2 \times \{1\}$, and
5. $\Delta((q_1,q_2,i),w) = (q_1', q_2', j)$ where $q_1' = \Delta_1(q_1,w), q_2' = \Delta_2(q_2,w)$ and

$$\begin{cases} j = 1 & \text{if } q_1 \in \mathcal{F}_1 \text{ and } i = 2 \\ j = 2 & \text{if } q_2 \in \mathcal{F}_2 \text{ and } i = 1 \\ i = j & \text{false.} \end{cases}$$

□

The main idea of this construction is that if a run ρ is accepted, there exists two states in $inf(\rho)$ which are $((q_i, q_j, 1))$ and $((q_k, q_l, 2))$ where i, j, k, l are arbitrary number. Hence, by the construction, both $\mathcal{F}_1$ and $\mathcal{F}_2$ is visited infinitely often.

Proposition 2.3. *Let $\mathcal{A}$ be a Büchi automaton. Then there exists a Büchi automaton $\overline{\mathcal{A}}$ such that $L(\overline{\mathcal{A}}) = \Sigma^\omega - L(\mathcal{A})$ [2].*

2.2.2 Generalized Büchi Automata

An ω-automaton $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, \mathcal{Q}_0, \mathcal{F})$ is called Generalized Büchi automata iff the acceptance condition is defined as follows: $\mathcal{F} \subseteq 2^{\mathcal{Q}}$, e.g. $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k\}$ and for all $1 \le i \le k$, $\mathcal{F}_i \subseteq \mathcal{Q}$. A run ρ on an infinite word w is accepted by $\mathcal{A}$ iff

$$inf(\rho) \cap \mathcal{F}_i \neq \varnothing \text{ for every } \mathcal{F}_i \in \mathcal{F}.$$

In other words, there exists at least one state q for each $\mathcal{F}_i \in \mathcal{F}$ is visited infinitely often on ρ. A word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$ iff there is a corresponding accepting run ρ.

Proposition 2.4. *Let $\mathcal{A}_1$ be a generalized Büchi automaton. There is a Büchi automaton $\mathcal{A}$ which accepts the same language of $\mathcal{A}_1$, which means $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1)$.*

Proof. Let $\mathcal{A}_1 = (\Sigma_1, \mathcal{Q}_1, \Delta_1, \mathcal{Q}_{01}, \mathcal{F}_1)$, where $\mathcal{F}_1 = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k\}$. Let $\mathcal{A}$ is a 5-tuple $(\Sigma, \mathcal{Q}, \Delta, \mathcal{Q}_0, \mathcal{F})$, where

1. $\Sigma = \Sigma_1$,
2. $\mathcal{Q} = \mathcal{Q}_1 \times \{1..k\}$,
3. $\mathcal{Q}_0 = \{q_0\}$,
4. $\mathcal{F} = \mathcal{F}_1 \times \{1\}$,
5. $\Delta(q_0, w) = (q, 1)$ if there exists a $q_i \in \mathcal{Q}_{01}$, $\Delta_1(q_i, w) = q$, and
6. $\Delta((q', i), w) = (q, j)$ if $\Delta_1(q, w) = q'$ and $\begin{cases} j = i + 1 \pmod{n} & \text{if } q \in \mathcal{F}_i \\ j = i & \text{if } q \notin \mathcal{F}_i \end{cases}$

□

In order to record which acceptance set we are eager to visit, the third flag on state is needed. This idea is quite the same as the intersection operation of Büchi automata. Once a run ρ visits a state flagged with j, which means there is a state in $\mathcal{F}_j$ of $\mathcal{A}_1$ is visited. If the flag can always change from 1 to k infinitely often, every corresponding accepting set $\mathcal{F}_i \in \mathcal{F}$ is visited infinitely often. Hence, this run should be accepted by $\mathcal{A}$.

2.2.3 Transition-Based Generalized Büchi Automata

An ω-automaton $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, \mathcal{Q}_0, \mathcal{F})$ is called a transition-based generalized Büchi automaton if the acceptance condition is defined as follows: $\mathcal{F} \subseteq 2^{\Delta}$, which means the acceptance is a set of subset of transitions, e.g. $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n\}$ and for all $1 \le i \le n$, $\mathcal{F}_i \subseteq \Delta$. A run ρ on an infinite word w is accepted by $\mathcal{A}$ if

$$\mathit{inf}(\rho) \cap \mathcal{F}_i \neq \varnothing \text{ for every } \mathcal{F}_i \in \mathcal{F}.$$

In other words, there exists at least one transition t for every $\mathcal{F}_i \in \mathcal{F}$ appears infinitely often on the run ρ. A word $w \in \Sigma^{\omega}$ is accepted by $\mathcal{A}$ if there is a corresponding accepting run ρ.

2.2.4 Co-Büchi Very Weak Alternating Automata

An alternating automaton is called *very weak* (abbreviate as VWAA) if the following properties hold.

- There exists a partition of $\mathcal{Q}$ into disjoint sets $\mathcal{Q}_i$, such that either the accepting set $\mathcal{Q}_i \subseteq \mathcal{F}$ or $\mathcal{F} \cap \mathcal{Q}_i = \varnothing$.
- There exists a partial order $\le$ on the set of $\mathcal{Q}_i$ such that, for every $q \in \mathcal{Q}_i$, if $q' \in \mathcal{Q}_j$ occurs in $\Delta(q, w)$, then $\mathcal{Q}_j \le \mathcal{Q}_i$.

Thus, a run DAG of a VWAA will eventually "trapped" within a partition $\mathcal{Q}'$.

A VWAA with co-Büchi acceptance condition if the accepting set $\mathcal{F}$ is a subset of $\mathcal{Q}$. A run DAG ρ of co-Büchi VWAA is accepting if any infinite branch in ρ has only finite number of nodes labeled in $\mathcal{F}$.

2.3 Propositional Linear Temporal Logic (PTL)

Temporal logic is a description logic which is used to represent and reason about the specification of a system which is qualified in terms of time. Any logic which views time as a sequence of states is a temporal logic. It was first introduced by A. Prior in the 1960's, and developed further by A. Pnueli for computer usage. A. Pnueli pointed out that temporal logic is useful when people trying to verify and specify the software programs especially for concurrent, reactive, and non-terminating programs such as operating system [21].

Temporal logic is used to formalize the describing sequences of transitions between states in a reactive system, which can be represented as a Kripke structure [5]. A Kripke structure $\mathcal{M}$ can be defined as 4-tuple $(\mathcal{Q}, \mathcal{Q}_0, R, L)$ where $\mathcal{Q}$ is the set of state, $\mathcal{Q}_0$ is the set of initial state, R is the total transition relation between two states, and L is the labeling function which labels each state with a set of propositions if the propositions is true in the state. A path π of $\mathcal{M}$ from a state q is an infinite sequence of states $\pi = q_0, q_1, \dots$ such that $q_0 = q$ and $(q_i, q_{i+1}) \in R$ for all $i \geq 0$. Temporal formulae are then used to describe the properties about a state or a path, which would be called state formulae and path formulae. A state formula describes what property should be true at the current state while a path formula describes what property should be true along the specific path.

A formula written in temporal logic can specify the property of a program by the temporal operators. For example, we can use *always* operator to describe that some properties, sometimes called specifications, would *always* be true, which is usually considered as a safety property of a distributed system. Notice that temporal operators can also be combined with one another.

Propositional linear temporal logic is a restricted linear temporal logic which only allowing boolean variables.

State formulae, boolean operators, and temporal operators are contained in linear temporal logic [18]. The temporal operator can be separated into two parts, which are future operators and past operators. Here we would only focus on the future operators. Let π^i denotes the suffix of π staring at q_i in the sequence of path. The definition are as follows:

State Formulae

- For a state formula p,

$$M, \pi \models p \Leftrightarrow s \text{ is the first state of } \pi \text{ and } M, s \models p.$$

Boolean Operators

The following are the semantics of some boolean operations.

- **Negation**: $\neg p$,

$$M,\ \pi\ \models\ \neg\, p \Leftrightarrow M,\ \pi\ \not\models\ p.$$

- **Disjunction**: $p \vee q$,

$$M,\ \pi\ \models\ p \vee\ q \Leftrightarrow M,\ \pi\ \models\ p \text{ or } M,\ \pi\ \models\ q.$$

- **Conjunction**: $p \wedge q$,

$$M,\ \pi\ \models\ p \wedge\ q \Leftrightarrow M,\ \pi\ \models\ p \text{ and } M,\ \pi\ \models\ q.$$

There are some other operations which are not introduced here such as implication ($\rightarrow$) and equivalence ($\leftrightarrow$) can be defined by negation, disjunction, and conjunction for simplicity.

Future Operators

Here we are going to introduce the semantics of the future operators.

- **Next**: $\bigcirc p$, or sometimes be written as $\mathcal{X}$ p,

$$M, \pi \models \bigcirc p \Leftrightarrow M, \pi^1 \models p.$$

- **Eventually**: $\Diamond p$, or sometimes be written as $\mathcal{F}$ p,

$$M, \pi \models \Diamond p \Leftrightarrow \text{ for some } k \geq 0, M, \pi^k \models p.$$

- **Always**: $\Box p$, or sometimes be written as $\mathcal{G}$ p,

$$M, \pi \models \Box p \Leftrightarrow \text{ for all } i \geq 0, M, \pi^i \models p.$$

- **Until**: $p\,\mathcal{U}\,q$,

$$M, \pi \models p\,\mathcal{U}\,q \Leftrightarrow \text{ for some } k \geq 0, M, \pi^k \models q, \text{ and for all } 0 \leq i \leq k, M, \pi^i \models p.$$

- **Release**: $p \mathcal{R} q$,

$$M, \pi \vDash p \, \mathcal{U} \, q \Leftrightarrow \text{ for all } k \geq 0, \text{ if for every } i < k, M, \pi^i \nvDash p, \text{ then } M, \pi^k \vDash q.$$

- **Waits for**: $p \mathcal{W} q$,

$$M, \pi \vDash p \, \mathcal{W} \, q \Leftrightarrow M, \pi \vDash p \, \mathcal{U} \, q \text{ or } M, \pi \vDash \Box p.$$

Chapter 3

Related Work

One of the members in our group, W.-C. Chan, had given a comparison study of some algorithms in 2007 [3]. He compared five algorithms which were Tableau [18], Incremental Tableau [13], Temporal Tester [14], GPVW [9], and LTL2AUT [7].

He concluded that GPVW and LTL2AUT perform better than the others in terms of the state size. Both these algorithms have been improved as GPVW+ and LTL2AUT+. We will follow this conclusion and take more algorithms to have a more comprehensive comparison.

3.1 Translation Algorithms

Here we will briefly describe the algorithms we are going to focus on in this thesis. The section is ordered by the acceptance condition of the result automaton and the proposed year of each algorithm.

3.1.1 Couvreur's Algorithm

This work is presented in [6]. This algorithm presents a way to translate an LTL formula to a transition-based generalized Büchi automaton (TGBA). The first step in this algorithm is to expand the given formula to an expression which can obtain the information about the property which the current and the next states should satisfy and what kind of property had been satisfied in each step. With this expression, the automaton is constructed by translating each expression element into the corresponding transitions and states. Second, for every $\mathcal{U}$-formula ψ, it build a corresponding accepting set which contains every transition which makes ψ be satisfied. Then, the construction is completed.

The space complexity of this algorithm depends on the number of temporal operators in the given formula f.

3.1.2 LTL2BA

This algorithm translates an LTL formula into a Büchi automaton with three stages: (1) translating from LTL formulae to very week alternating co-Büchi automata (VWAA), (2) translating VWAA with co-Büchi condition into TGBA, and (3) translating TGBA into Büchi automata [8]. In the first stage, the states of the automaton is actually corresponding to the subformulae of the given formula. For each successor formula of the corresponding formula of the current state, if it is a conjunction formula, the algorithm will generate an and-branch transition and states, generate single state and transition otherwise. The accepting set will be a set of states which the corresponding formulae are until-formulae. The second stage translates the previous result into TGBA. For each until subformula f of the given formula, the construction will generate an accepting set which contains all the transitions whose successor state does not refer to any state in VWAA corresponding to f. The third stage translates the previous result TGBA into BA. It constructs the result Büchi automaton with an additional integer to keep track of which accepting condition is looking forward to. Each intermediate automaton is simplified on-the-fly, in order to save memory and time.

3.1.3 LTL2BUCHI

This is also a translation algorithm for a linear temporal formula to a Büchi automaton using transition-based generalized Büchi automata as intermediate automata. The main feature of this algorithm is that it records the information on transition rather on state, which allows it to merge states [10]. The core method of this algorithm is the same as LTL2AUT [7], but the ways collecting accepting set and merging two states are different. Merging states in this algorithm is more rigid since the information about previous state and accepting condition in a state may differ. This information should not be lost because the relation between states in transition-based automata is essential. Every state in this construction contains two list about accepting set, one records the $\mathcal{U}$-subformula in the given formula while the other contains those which are satisfied at the transition. These two lists are useful when computing the accepting set by building set of transitions for each $\mathcal{U}$-formula which is satisfied at this transition.

The paper also gives an algorithm for TGBA to Büchi automaton translation. It provides a construction of an automaton $\mathcal{A}'$ with Büchi acceptance condition. By computing an automaton $\mathcal{A}$ which is the intersection automaton of $\mathcal{A}'$ and the target TGBA $\mathcal{A}_{\mathcal{TGBA}}$, the corresponding automaton $\mathcal{A}$ is constructed, which means the language of $\mathcal{A}$ and $\mathcal{A}_{\mathcal{TGBA}}$ are equivalent.

3.1.4 GPVW+

GPVW is a simple on-the-fly algorithm proposed in [9]. It keeps the information of elementary formulae, $\mathcal{U}$-formulae, and the right-hand side formulae of $\mathcal{U}$-formulae in each state. The $\mathcal{U}$-formulae are kept for the accepting condition for generalized Büchi automaton. Once the right-hand side formula of an $\mathcal{U}$-formula f holds in the current state, the $\mathcal{U}$-formula is satisfied, which implies the accepting set corresponding to f should contains the current state. They also proposed a new way to detect the contradiction and redundancies for states.

3.1.5 LTL2AUT+

LTL2AUT+ improved GPVW by syntactically implication [7]. They deduce the information for a state by keeping the information of elementary formulae. The improvement helps merging states and detecting contradiction and redundancies.

3.1.6 MoDeLLa

The main idea of this method is that generated automata should be more deterministic rather than smaller because the final product in model checking may be smaller with the help of deterministic automata [22]. However, some LTL formulae cannot be translated into an equivalent deterministic Büchi automata, and even deciding whether the translation is possible belongs to EXPSpace and is PSpace-Hard [15]. In this way, this algorithm translates the given formula into a generalized Büchi automaton “as deterministic as possible” when computing the cover of the formula eventhough it may generate more states than other algorithms. First, the rewriting rule which would cause nondeter-

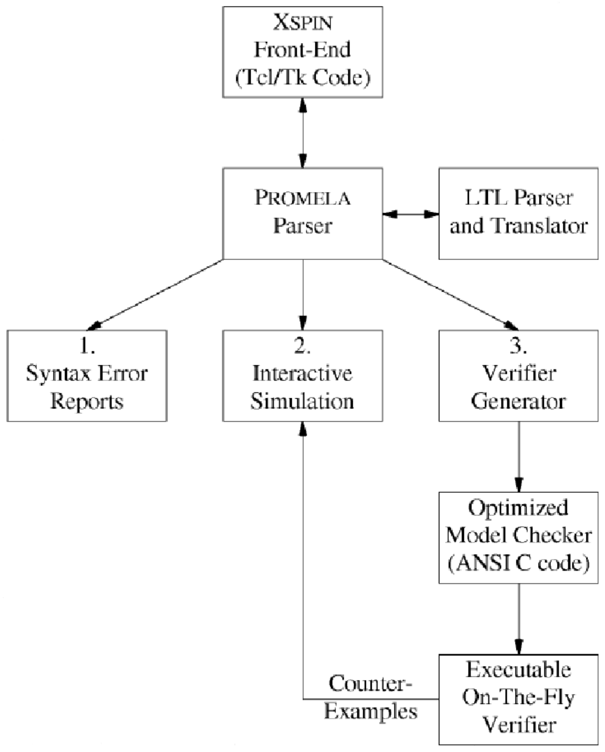

XSPIN Front-End (Tcl/Tk Code)
PROMELA Parser
LTL Parser and Translator
1. Syntax Error Reports
2. Interactive Simulation
3. Verifier Generator
Optimized Model Checker (ANSI C code)
Executable On-The-Fly Verifier
Counter-Examples

Figure 3.1: The structure of Spin simulation and verification

minism should be omitted, which would make the cover "less nondeterministic". Second, Shannon's expansion should be applied to the top level boolean propositions of the cover, which would make the cover "more deterministic". The last step is to merge the X-part of the cover. This merging action is not always safely applicable since the corresponding states would not be in the same accepting set. Therefore, this action can only be applied when it is guaranteed not to cause incorrectness. However, this work is only based on "as deterministic as possible". The result may be good enough for model checking usage.

3.2 Tools

There are some tools which are related to automata-based model checking. We will give a brief instruction for each of them in this section.

3.2.1 SPIN

Spin [11, 12] is a well-know model checker, can be used for the formal verification of asynchronous process systems. The tool was developed at Bell Labs, written by Gerard

J. Holzmann and others, starting in 1980. The basic structure is illustrated in Fig. 3.1.

XSpin is a graphical front-end for Spin. To verify a design, Spin accepts design specifications written in a high level language, called Promela (a PROcess MEta LAnguage), and it accepts correctness requirements expressed as Linear Temporal Logic (LTL) formulae. Promela parser can also fix syntax errors, and perform interactive simulation to roughly ensure that the design behaves as intended. Then, Spin generates an optimized on-the-fly verification program, which will be compiled and executed. Counterexamples to the correctness claims can be fed back into the interactive simulator, if detected.

The fundamental technique of Spin is logic model checking. M.Y. Vardi and P. Wolper extended model checking with an automata theoretic model [16]. The description of a concurrent system in Promela consists of one or more user-defined process templates, and Spin translates each into a finite automaton. The global system behavior can be obtained by computing an asynchronous interleaving product, resulting again represented by an automaton. Then, Spin takes the correctness claim in LTL formula and converts it into a Büchi automaton base on a simple on-the-fly construction [9]. The synchronous product of this claim and the automaton representing the global state space is again a Büchi automaton. The result of the validity of the claim is equivalent to the emptiness of this automaton. The correctness claims in Spin represents behaviors that are undesirable.

3.2.2 GOAL

GOAL (`http://goal.im.ntue.du.tw`) [24, 25] is a graphical interactive tool for user to define, manipulate and test temporal logics and ω-automata. The acronym GOAL is derived from "**G**raphical Tool for **O**mega-**A**utomata and **L**ogics". This tool is developed on JAVA and T.-K. Tsay is the leader of GOAL team at National Taiwan University. The graphical user interface of GOAL is extended from JFLAP.

The GOAL tool is used to be an educational assistant in the first place, helping users learning ω-automata theory and temporal logic. Recently, The GOAL tool had been proposed as a research tool because of the expanded collection of translation, simplification, and completementation algorithms. User can also write a program to access GOAL functions with command-line mode. The utility functions for some common tasks such as random formulae generation, and statistics collection are also provided.

GOAL is now provided the following functions:

- **Editing, Running, Testing, and Simplifying Büchi Automata**:
  One can easily point-and-click and drag-and-drop to build up a Büchi automaton. Once the automaton is created, he/she can easily run it by given input to see what kind of input language the automaton would accept or testing for emptiness. Not only that, any Büchi automaton can be simplified with the help of simplification algorithms which had been implemented. With simplification, user can get a smaller automaton which is equivalent to the original one, which would be much easier to understand.

- **Translating QPTL (and LTL) Formulae into Büchi Automata**:
  Numbers of translation algorithms have been implemented in GOAL. User can write a QPTL or LTL formula and translate it to a Büchi automaton via these algorithms. GOAL imposes a restriction that a quantifier must not fall in the scope of a temporal operator. This function would help user to get more understanding about the algorithm which he/she is interested in.

- **Boolean Operations on Büchi Automata**:
  The three standard boolean operations – union, intersection, and complementation are supported in GOAL.

- **Tests on QPTL Formulae**:
  Satisfiability and validity tests are supported. Even though the equivalent test between two QPTL formulae is not supported, one can use the mutual implication operator ($\leftrightarrow$) to accomplish the same feature.

- **Exporting Büchi Automata as Promela Code**:
  User can export the automaton in the PROMELA syntax on the screen or as a file. This feature makes it possible to use GOAL as a graphical specification definition frontend to an automata-theoretic model checker like SPIN.

- **The Automata Repository**:
  The repository in GOAL contains a collection of frequently used QPTL formulae

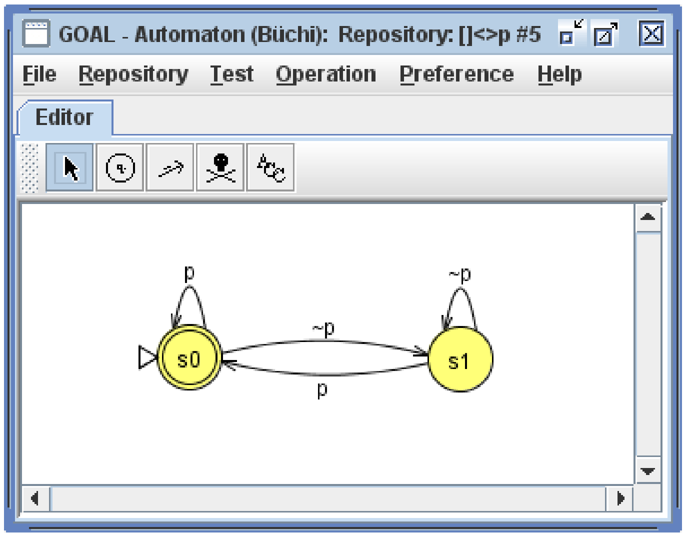


Figure 3.2: The editing environment of GOAL

and their corresponding equivalent automata. This is a very convenient way for learning the relation between Büchi automata and QPTL for beginners.

Chapter 4

Translation Algorithms

In this chapter, we will describe the translation algorithms in detail. Three of them use transition-based generalized Büchi automata (TGBA) as intermediate automata. The accepting sets of TGBA are transition sets rather than state sets, which helps reducing the state size of the automaton. Moreover, optimization for automaton can be introduced during each step. Hence, the translation algorithm with intermediate automaton can generates smaller automaton than others at most of the time.

The last one, MoDeLLa, proposes a different point of view. The algorithm doesn't put their strength on generating smaller automaton. Conversely, they propose that if the translation algorithm generates the automaton as determined as possible, which would generates more states, will helps reduce the size of the product automaton in model-checking procedure.

4.1 Couvreur's Algorithm

This algorithm translates the input temporal formula into a TGBA. The automaton construction is very similar to the one proposed in [9]. This algorithm is based on symbolic computation over a set of boolean variables. A boolean variable r_f of a temporal formula f will be expanded into three part, which are the alphabets should be true in current state, the $\mathcal{U}$-formulae have to be true from the next state on, and the formulae have to be true from the next state on. For an infinite word $\sigma = x_0\dot{x}_1\dot{x}_2\ldots$ over the alphabet 2^{AP}, r_f corresponds to $f \vDash \sigma$ and $a_{f\,\mathcal{U}g}$ corresponding to $(\sigma \vDash f\ \mathcal{U}\, g) \wedge \neg(\sigma \vDash g)$. There are several fundamental rules used for formula expansion, which are listed in Table 4.1 where p is a literal. With these fundamental rules, the variable r_f of the given LTL formula f

r_p	p
$r_{\neg p}$	$\neg p$
$r_{f \wedge g}$	$r_f \wedge r_g$
$r_{f \vee g}$	$r_f \vee r_g$
$r_{f\,\mathcal{U}\,g}$	$r_g \vee r_f \wedge r_{\bigcirc(f\,\mathcal{U}\,g)} \wedge a_{f\,\mathcal{U}\,g}$
$r_{f\,\mathcal{R}\,g}$	$r_f \wedge r_g \vee r_g \wedge r_{\bigcirc(f\,\mathcal{R}\,g)}$

Table 4.1: Formula expansion rules

can be expressed in an expression form which only uses the variables of these form: p, $\neg p$, a_g, and $r_{\bigcirc(g)}$, where the variables p are atomic proposition and g are subformulae of f. Proposition 1 is the application of this property to a set of temporal formula F.

Proposition 4.1. *Let F be a set of formulae. $\Delta(F) = \prod_{f \in F} r_f$ can be expanded to the form:*

$$\prod_{f \in F} r_f = \sum_{(X, NAcc, Next) \in L_F} \left(X \wedge \prod_{g \in NAcc} a_g \wedge \prod_{h \in Next} r_{\bigcirc(h)} \right)$$

with

$L_F \subseteq 2^{2^{AP}} \times \{g\,\mathcal{U}\,h \in Sub(f)\}$

$$\times \{\{g\,\mathcal{U}\,h \in Sub(f)\} \cup \{\{g\,\mathcal{R}\,h \in Sub(f)\} \cup \{g \in Sub(f) : \bigcirc g \in Sub(f)\}\}.$$

and $Sub(f)$ is the subformula set of f.

The automaton construction then starts by the expression form of input formula f. X will be considered as alphabet which should be true immediately in the current state, $NAcc$ is the $\mathcal{U}$-formulae which should hold from the next state on, and $Next$ is the formulae which should hold from the next state on. Each implication of this expansion will define a transition $(f, X, AccSet, g)$, where $AccSet = Acc \setminus NAcc$, Acc is the $\mathcal{U}$-formula set of $Sub(f)$, and $(X, NAcc, Next) \in L_f$ and $g \in Next$. The third tuple of a transition $Acc \setminus NAcc$ is used to record which $\mathcal{U}$-formulae are not concerned in the future. For each formula $g \in Acc$, the construction will create a transition set which collects all the transitions with the third tuple contains g. Hence, if a transition $(f, X, AccSet, g)$ in a run of the result TGBA, $AccSet$ gives the information of which $\mathcal{U}$-formulae are satisfied in the prefix of the path or are not concerned in the suffix. The $\mathcal{U}$-formulae which are not in $AccSet$ should be focused on in the suffix. The construction will then continue to expand all the formulae $g \in Next$ until all the reachable states are created. Theorem 4.2 formalizes the resulting automaton.

Theorem 4.2. *Let f be an LTL formula. Let $A_f = (\Sigma, Q, \to, q_0, Acc)$ be the transition generalized Büchi automaton where*

- *Σ is the set of atomic proposition of f,*
- *$Q = \{F \subseteq \{g\, \mathcal{U}\, h \in Sub(f)\} \cup \{g\, \mathcal{R}\, h \in Sub(f)\} \cup \{g \in Sub(f) : \bigcirc g \in Sub(f)\} \cup \{f\}\}$,*
- *$\to$: $(f, X, AccSet, g)$, where $AccSet = Acc \setminus NAcc$, and $(X, NAcc, Next) \in L_f$ and $g \in Next$,*
- *$q_0 = \{f\}$, and*
- *$Acc = \{g\, \mathcal{U}\, h \in Sub(f)\}$.*

An accepting run of the result TGBA will infinitely often go through the transitions for each accepting set in Acc which means each $\mathcal{U}$-subformulae is satisfied at some point of transition sequence. Hence the corresponding word for the run satisfies the input temporal formula. In another direction, a word which satisfies the temporal formula f will satisfies each $\mathcal{U}$-subformula at some point of a alphabet sequence, say it satisfies one $\mathcal{U}$-subformula $g\, \mathcal{U}\, h$ after reading the first ith alphabet $a_1, a_2, \ldots, a_i$. One of the corresponding runs will go through a transition $(g\, \mathcal{U}\, h, a_i, \{g\, \mathcal{U}\, h\}, h)$. In the suffix of the run, formula $g\, \mathcal{U}\, h$ will always be recorded on transitions because it is satisfied in the prefix of the path.

Here is an example of the construction for formula $f = \Box(p\, \mathcal{U}\, q)$

Example 4.3. *Construct an automaton for formula $f = \Box(p\, \mathcal{U}\, q)$. Let $g = (p\, \mathcal{U}\, q)$. We deduce that $Acc = \{g\}$. The expansion of formula f will be:*

$$\begin{aligned} r_f &= r_g \vee r_{\bigcirc(f)} = ((r_q) \vee (r_p \wedge a_g \wedge r_{\bigcirc(g)})) \wedge r_{\bigcirc(f)} \\ &= (r_q \wedge r_{\bigcirc(f)}) \vee (r_p \wedge a_g \wedge r_{\bigcirc(g \wedge f)}) \end{aligned}$$

This expression will produce two transitions:

$$\begin{gathered} (\{f\}, q, \{g\}, \{f\}) \\ (\{f\}, p, \varnothing, \{g, f\}) \end{gathered}$$

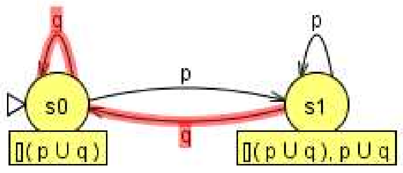


Figure 4.1: $TGBA_f$ generated by Couvreur's algorithm

The algorithm then produces the successors of state $\{g, f\}$:

$$\begin{aligned} r_{g\wedge f} &= r_g \wedge r_f = (r_q \vee (r_p \wedge a_g \wedge r_g)) \wedge ((r_q \wedge r_{\bigcirc(f)}) \vee (r_p \wedge a_g \wedge r_{\bigcirc(g\wedge f)})) \\ &= (r_q \wedge r_{\bigcirc(f)}) \vee (r_p \wedge r_q \wedge a_g \wedge r_{\bigcirc(g\wedge f)}) \vee (r_p \wedge a_g \wedge r_{\bigcirc(g\wedge f)}) \end{aligned}$$

Three more transitions will be produced:

$$\begin{array}{l} (\{g,f\},\{q\},\{g\},\{f\}) \\ (\{g,f\},\{p,q\},\varnothing,\{g,f\}) \\ (\{g,f\},\{p\},\varnothing,\{g,f\}) \end{array}$$

Since there are no more states created, the construction is completed and Fig. 4.1 gives the resulting automaton. Note that the red-labeled transitions are accepting transitions.

4.2 LTL2BA

This algorithm translate an LTL formula into a Büchi automaton with three stages: (1) translating from LTL formula to very week alternating co-Büchi automaton (VWAA), (2) translating VWAA with co-Büchi condition into TGBA, and (3) translating TGBA into Büchi automaton [8].

4.2.1 LTL to VWAA

The first stage generates the very weak co-Büchi alternating automaton from the LTL formula. Each state in $VWAA$ will actually corresponding to a subformula of the input formula f. In order to reach this goal, two operators are defined as follows.

Definition 4.4.
For $T_1, T_2 \in 2^{2^{\Sigma} \times Q}$ we define

$$T_1 \otimes T_2 = \{(\alpha_1 \cap \alpha_2, e_1 \wedge e_2) \mid (\alpha_1, e_1) \in T_1 \text{ and } (\alpha_2, e_2) \in T_2\}$$

For an LTL formula f we define $\overline{f}$ by: $\overline{f} = \{f\}$ if f is a temporal formula,

$$\overline{f_1 \wedge f_2} = \{e_1 \wedge e_2 \mid e_1 \in \overline{f_1} \text{ and } e_2 \in \overline{f_2}\} \text{ and } \overline{f_1 \vee f_2} = \overline{f_1} \cup \overline{f_2}.$$

The operator $\otimes$ computes the products of the alphabet and formula pair and $\overline{f}$ gives roughly the DNF of f. Here is the construction translating LTL formula into $VWAA$.

Theorem 4.5. *Let f be an LTL formula, and Prop is the proposition set for f. Let $VWAA_f = (\Sigma, Q, \delta, q_0, Acc)$ be the result VWAA where*

- $\Sigma = 2^{Prop}$,
- *Q is the set of temporal subformulae of f,*
- $q_0 = \overline{f}$,
- *Acc is the set of $\mathcal{U}$-subformulae of f, and*
- *δ is defined as follows (Δ is used to extend δ to all subformulae of f):*

$$\begin{cases} \delta(True) & = \{(\Sigma, True)\} \\ \delta(p) & = \{(\Sigma_p, True)\} \\ \delta(\neg p) & = \{(\Sigma_{\neg p}, True)\} \\ \delta(\bigcirc f) & = \{(\Sigma, e) \mid e \in \overline{f}\} \\ \delta(f_1 \,\mathcal{U}\, f_2) & = \Delta(f_2) \cup (\Delta(f_1) \otimes \{(\Sigma, f_1 \,\mathcal{U}\, f_2)\}) \\ \delta(f_1 \,\mathcal{R}\, f_2) & = \Delta(f_2) \otimes (\Delta(f_1) \cup \{(\Sigma, f_1 \,\mathcal{R}\, f_2)\}) \\ \Delta(f) & = \delta(f) \text{ if } f \text{ is a temporal formula} \\ \Delta(f_1 \vee f_2) & = \Delta(f_1) \cup \Delta(f_2) \\ \Delta(f_1 \wedge f_2) & = \Delta(f_1) \otimes \Delta(f_2) \end{cases}$$

Σ_p is the alphabet set contains proposition p. Note that if we treat "subformula of" as partial order of formulae, it is easy to see that $VWAA_f$ is very weak. The accepting set collects the states corresponding to $\mathcal{U}$-subformulae. A run tree which will be accepted by $VWAA_f$ will eventually reach a SCC which will not contains any $\mathcal{U}$-subformulae, which means all the $\mathcal{U}$-subformulae are satisfied.

Example 4.6. *Fig. 4.2 gives the result of the construction where $f = \Box(p \,\mathcal{U}\, q)$.*

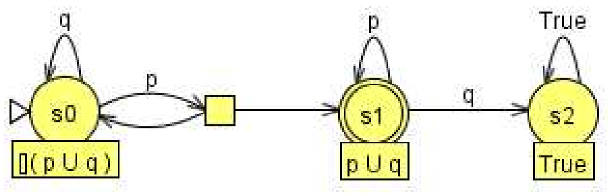


Figure 4.2: $VWAA_f$ generated by LTL2BA

4.2.2 VWAA to TGBA

An usual method to transform an alternating automaton is the algorithm proposed by Miyano and Hayashi[19]. Yet sometimes the algorithm generates a Büchi automaton which is too big in terms of state size. Thus, the algorithm generates a transition-based generalized Büchi automaton (TGBA) as intermediate automaton and translates the TGBA into a Büchi automaton in the third stage.

Here is the construction for building a TGBA from a co-Büchi $VWAA$.

Theorem 4.7. *Let $VWAA_f = (\Sigma, Q, \delta, q_0, Acc)$ be a co-Büchi $VWAA$. We define the $TGBA = (\Sigma, Q', \delta', q'_0, \mathcal{T})$ where*

- *$Q' = 2^Q$ is identified with conjunction of states,*
- *$q'_0 = \{q_0\}$,*
- *$\mathcal{T} = \{T_a \mid a \in Acc\}$ where*
  $T_a = \{(s, \alpha_1, s_1) \mid a \notin s_1$ or $\exists(\alpha_2, s_2) \in \delta(a), \alpha_1 \subseteq \alpha_2, s_2 \subseteq s_1,$ and $a \notin s_2\}$, and
- *$\delta''(q_1 \wedge \ldots \wedge q_n) = \otimes_{i=1}^{n} \delta(q_i)$,*
- *δ' is the set of $\leq$-minimal transitions of δ'' where a partial order relation $\leq$ of transitions $t_1 = (s, \alpha_1, s_1), t_2 = (s, \alpha_2, s_2)$ is defined by $t_1 \leq t_2$ if $\alpha_1 \subseteq \alpha_2$, $s_2 \subseteq s_1$, and $\forall T \in \mathcal{T}$, $t_2 \in T \Rightarrow t_1 \in T$.*

Since $VWAA_f$ is very weak, a state can only reach some states whose ordering is lower than or equals to it. This property gives an explanation of how the accepting set $\mathcal{T}$ is chosen. For each accepting set $T_a \in \mathcal{T}$, the transitions are collected either they

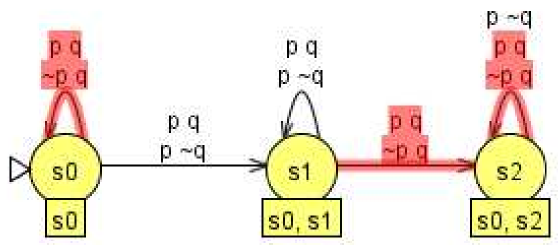


Figure 4.3: $TGBA_f$ generated by LTL2BA

don't transit to a state related to the formula a, which is an accepting state of $VWAA_f$ or they transit to a state corresponding to a state set s_1 contains a, but exists another transition to a state s_2 which doesn't contains a. It is very obvious if the transitions are collected because of the first condition. The transition leads the run tree to a state set without a, and because of the "very weak" property, the suffix of the run tree will never reach a again. For some special cases of $VWAA_f$ and a word ω, the state a is visited infinitely often in the run tree but finitely many times in each branch which is accepted by $VWAA_f$. The second condition above then is used to collect the transitions which will reach a infinitely often in these situations. Hence, the result TGBA will also accept the word ω. Fig. 4.3 shows the result TGBA of the example automaton after translating from $VWAA_f$. Notice that the accepting set only contains one accepting transition set and the red-labeled transitions are the accepting transitions.

4.2.3 TGBA to BA

The last stage is to transform the TGBA into a Büchi automaton. This algorithm is quite similar to the translation algorithm for generalized Büchi automaton to Büchi automaton, which we described in proposition 2.4.

Theorem 4.8. *Let $TGBA = (\Sigma, Q, \delta, q_0, \mathcal{T})$ be the TGBA from previous stage. Let $BA = (\Sigma, Q \times \{0, 1, \dots, r\}, \delta', q_0 \times \{0\}, Q \times \{r\})$ where*

- $\delta'((q, j)) = \{(\alpha, (q', j')) \mid (\alpha, q') \in \delta(q) \text{ and } j' = next(j, (q, \alpha, q'))\}$

$$with\ next(j,t) = \begin{cases} max\{j \le i \le r \mid \forall j \le k \le i, t \in T_k\} & if\ j \ne r \\ max\{0 \le i \le r \mid \forall 0 \le k \le i, t \in T_k\} & if\ j = r \end{cases}$$

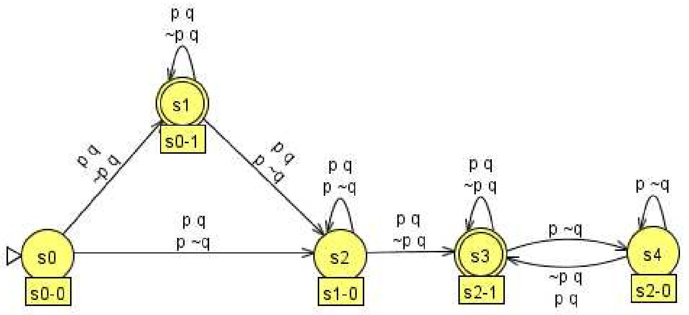


Figure 4.4: BA_f generated by LTL2BA

Usually, we check one accepting set at a time to keep track of at least one state/transition has been visited in every accepting set in GBA/TGBA. Here, the algorithm will seek as large index of the accepting set as it can to avoid creating unnecessary states for the result BA. Hence, the result BA will be small in terms of size.

Fig. 4.4 shows the result of the example automaton after translating from TGBA to BA.

4.3 LTL2BUCHI

This is also a translation algorithm for linear temporal formula to Büchi automata using transition-based generalized Büchi automata as intermediate automata[10]. It is actually an extended work of [9] and share a common framework with LTL2AUT [7]. Different from LTL2AUT, the information is recorded on the transition rather than state, which allows this algorithm to merge states while others cannot. The notes are labeled with sets of formulae, separated into several parts. The most important two parts are the formulae needed to be true immediately and the formulae have to be true from the next state on. We will first describe the data structure and present the algorithm afterwards.

4.3.1 Data Structure

The data structure of a node of this algorithm will have the following fields:

- **NodeId**: A unique node id. Id 0 is reserved for the initial state.
- **Incoming**: A set of node ID which records the incoming nodes of the current node.

- **ToBeDone**: A set of formulae that must hold at the current node and haven't been processed yet.
- **Old**: A set of already processed literals which must hold at the current node.
- **Next**: A set of formulae that should hold from the next state on.
- **Eventualities**: The set of promised and fulfilled eventuality obligations by the node. A promised obligation is an $\mathcal{U}$-formula that has been processed in the current node, and a fulfilled obligation is a formula processed in the current node that is the right-hand side argument of some $\mathcal{U}$-formulae processed in the current node.
- **Accepting**: The accepting sets to which the node belongs.
- **EquivClass**: The id of the equivalence class to which the node belongs.

The field *ToBeDone* of the node s will be denoted as "$s.ToBeDone$", and similarly for all the fields.

4.3.2 Algorithm

The algorithm for translating a formula f starts by creating an initial node *INIT* with *NodeId* = *EquivClass* = 0, *Next* = $\{f\}$, and with all other fields empty. A set of nodes will be collected when the algorithm is completed. Fig. 4.5 shows the main expansion algorithm.

Algorithm from LTL to TGBA For different type of formulae, the split function returns the split node with different formulae set to hold. The splitting function and rules is illustrated in Table 4.2 and Fig. 4.6.

The result $TGBA = (\Sigma, Q, \delta, q_0, T)$, where

- $\Sigma = 2^{literals}$,
- Q is $node\_set$,

```
input  : Set of nodes node_set
output: Set of nodes node_set
if this.ToBeDone is empty then                          // node has been fully computed
    compute_acc(this);
    if This equals to any other node $\in$ nodes_set then
        merge them;
    else                                       // processed node to be added to node set
        create new_node with ToBeDone := this.Next;
        expand(nodes_set);
    end
else                                  // still some formulae should be true, keep processing
    choose a formula next_formulae from this.ToBeDone to process;
    update_fulfilled_obligations(this, next_formula);
    if !(next_formulae contradicts this node or it is redundant) then
        if next_formula is a '$\mathcal{U}$','$\mathcal{R}$' or '$\vee$' formula then
            Node2 = this.split(next_formula);
            if next_formula is a '$\mathcal{U}$' formula then
                update_promised_obligations(this, next_formula);
            end
            return Node2.expand(this.expand(nodes_set));
        else if next_formula is a '$g \wedge h$' then
            ToBoDone = ToBoDone $\cup$ ($\{g, h\} \setminus$ Old);
            return this.expand(nodes_set);
        else if next_formula is a '$\bigcirc g$' then
            Next = Next $\cup$ $\{\varphi\}$;
            return this.expand(nodes_set);
        else                                                  // next formula is a literal
            Old = Old $\cup$ {next_formula};
            return this.expand(nodes_set);
        end
    end
end
```

Figure 4.5: The expansion algorithm for the node set

form	New1(form)	Next1(form)	New2(form)
$g\,\mathcal{U}\,h$	$\{g\}$	$\{g\,\mathcal{U}\,h\}$	$\{h\}$
$g\,\mathcal{R}\,h$	$\{h\}$	$\{g\,\mathcal{R}\,h\}$	$\{g, h\}$
$g \vee h$	$\{g\}$	$\varnothing$	$\{h\}$
$g \wedge h$	$\varnothing$	$\varnothing$	$\{g, h\}$

Table 4.2: Definitions of New and Next functions for non-literals

```
input  : formula form
output: A new split node
create Node2 with new ID.;
Node2.ToBeDone := this.ToBeDone $\cup$ (New2(form) $\setminus$ Old);
this.ToBeDone := this.ToBeDone $\cup$ (New1(form) $\setminus$ Old);
this.Next := Next $\cup$ Next1(form);
return Node2;
```

Figure 4.6: The splitting function

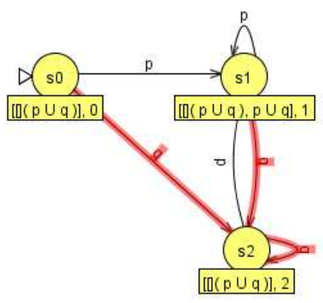


Figure 4.7: $TGBA_f$ generated by LTL2BUCHI

- $\delta = \{(nd_i, label_i, nd_j) \mid nd_i \in nd_j.Incoming \wedge label_i = nd_j.Old\}$,
- $q_0 = INIT \in Q$, and
- $T = \{T_i \mid T_i \subseteq \delta\}$, which are defined by $nd_j.Acceping$.

For each $\mathcal{U}$-subformula g of a input formula f, if the transition promised g holds and it actually fulfilled the promise, add g to the *succ_node.Accepting*. Fig. 4.7 illustrates the result TGBA generated by LTL2BUCHI for formula $\square(p\,\mathcal{U}\,q)$.

Degeneralization The translation algorithms are well known in the literature. Here they present a different method for degeneralize the TGBA from the previous step. First, A degeneralizer Büchi automaton will be created. The number of states depends on the number of accepting sets T in TGBA. A BA will be obtained by a TGBA by computing its synchronous product with the appropriate degeneralizer. A joint transition (t_1, t_2) of a degeneralizer with a TGBA is enabled if t_2 belongs to the accepting set that the predicate on t_1 requires. The accepting states of the products are the ones where the degeneralizer is in an accepting state. Fig. 4.8 illustrates the example of degeneralizers for the TGBA with the size of accepting set is one (left) or two (right). There is a priority relation between the successor transitions for each state in a degeneralizer. The priority of the transition set is based on how many accepting sets can the transition from TGBA fulfilled. The transition in a degeneralizer which can transit to most accepting sets has the highest priority and so on. If the transition didn't fulfill any of the accepting sets, the "else" transition will be chosen. The implicit meaning of the method is to focus on

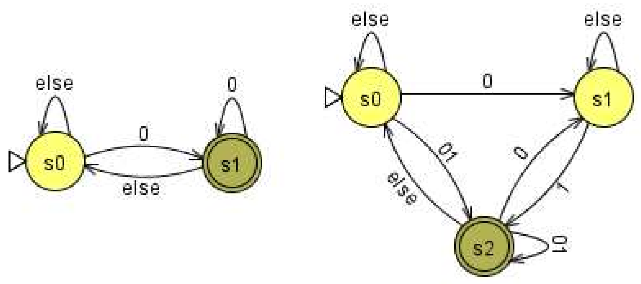


Figure 4.8: The examples of degeneralizers

how many accepting sets are fulfilled by the current transition. Once an accepting set is fulfilled, the construction will focus on the fulfillment status for the following accepting sets. If all the accepting sets are fulfilled infinite often, the result BA will contains a corresponding run which visits some states in the accepting set infinite often. The algorithm for generating the degeneralizer is described in Fig. 4.9. The result BA by this degeneralization algorithm for the TGBA in Fig. 4.7 is illustrated in Fig. 4.10.

4.4 MoDeLLa

For a system M, the standard technique for LTL model checking consists on translating temporal formula f into BA A_f and then checking the emptiness of the product $M \times A_f$. The size of the product $M \times A_f$ is the product of the size of M and A_f in the worst case, which brings up the idea of minimizing the size of A_f will help reducing the size of the final product. Fig. 4.11 shows the product of a system M with a non-deterministic and a deterministic automaton.

Yet in this algorithm, they come up with a new idea. Instead of reducing the size of A_f, trying to create a BA A'_f which makes the product automaton $M \times A'_f$ smaller without concentrating on the size of A'_f itself. Note the fact that if a state s in $M \times A_f$ is given by the combination of the state $s' \in M$ and $s'' \in A_f$, and if the successors of s'' is deterministic, then each successor state of s' can be combined consistently with exactly one successor of s'', they propose an algorithm to reducing the presence of non-deterministic decision states in A_f as much as possible.

```
input  : the size of accepting sets $size$
output: A degeneralizer BA
$nnodes := size + 1$;
$last := size$;
for $i = 0$ KwTo $nnodes$ do
    create automaton state $s_i$;
end
for $i = 0$ KwTo $last$ do
    for $j = last$ KwTo $i$ do
        create transition $trans$ from $s_i$ to $s_j$;
        for $k = i$ KwTo $j$ do
            add label $k$ to $trans$;
        end
    end
    create looping transition for $s_i$ labeled with else;
end
create looping transition $trans$ for $s_{last}$;
for $i = 0$ KwTo $last$ do
    add label $i$ to $trans$;
end
for $i = last - 1$ KwTo $0$ do
    if $i == 0$ then
        create transition from $s_{last}$ to $s_i$ labeled else;
    else
        create transition $trans$ from $S_{last}$ to $s_i$;
        for $j = 0$ KwTo $i$ do
            add label $j$ to $trans$;
        end
    end
end
```

Figure 4.9: The degeneralizer generating algorithm

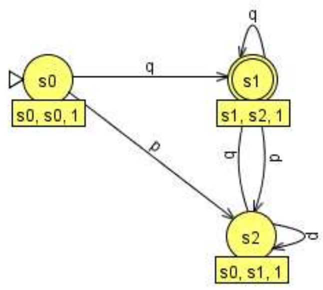


Figure 4.10: BA_f generated by the degeneralization algorithm

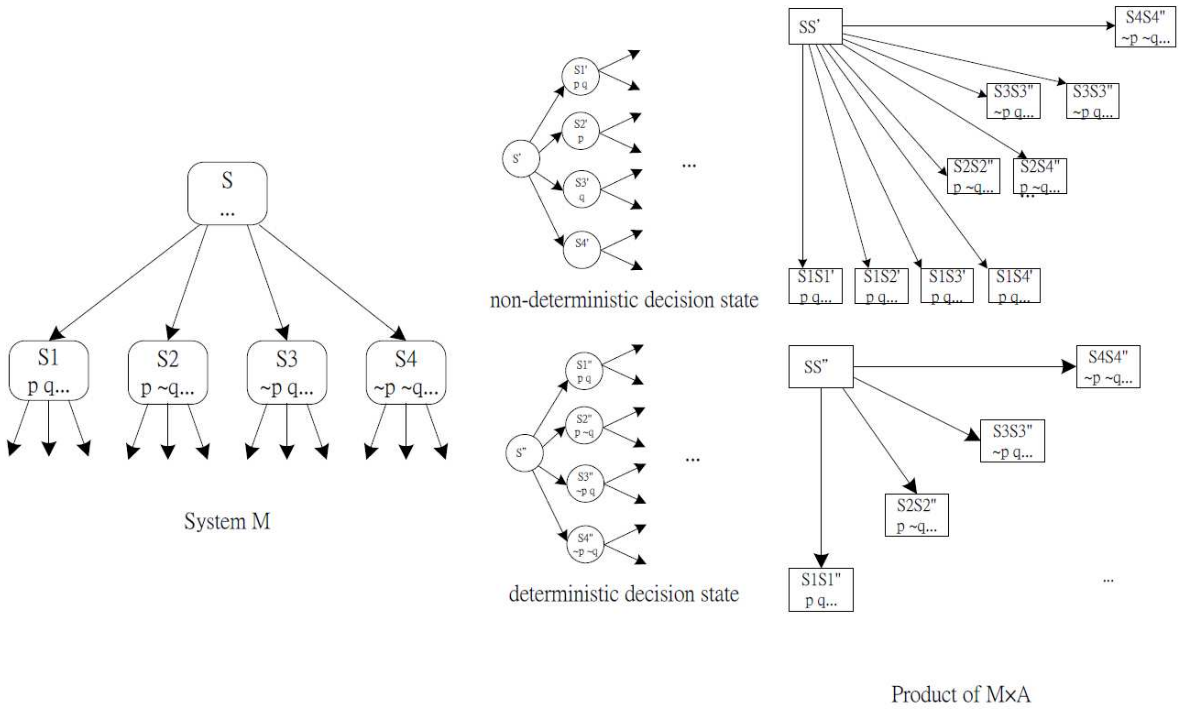


Figure 4.11: The difference of the product of system M with a deterministic or a non-deterministic automaton

4.4.1 Determining the covers

Proposition 4.9. *Let $\{f_k\}_k$ be a set of LTL formulae in negation normal form and $f = \wedge_k f_k$. Let $C = \{\{\Theta_{ij}\}_j\}_i$ which can be written as $\{\alpha_i \cup X_i\}_i$, where*

$$\begin{aligned} \alpha_i &= \{\Theta_{ij} \in \{\Theta_{ij}\}_j \mid \Theta_{ij} \text{ is a proposition literal}\} \text{ and} \\ X_i &= \{\Theta_{ij} \in \{\Theta_{ij}\}_j \mid \Theta_{ij} \text{ is a } \bigcirc\text{-formula}\} \end{aligned}$$

are the set of propositional literals and $\bigcirc$-formulae in $\{\Theta_{ij}\}_j$ respectively. Hence, f can be written in this form

$$f \leftrightarrow \bigvee (\alpha_i \wedge X_i)$$

A cover $C = \{\alpha_i \cup X_i\}_i$ is a deterministic cover if and only if all α_i's are pairwise mutually inconsistent, non-deterministic otherwise. Each element $(\alpha_i \wedge X_i)$ in a cover will represent a state in the result GBA, where α_i is the label of the state, and X_i is the next part of the state.

There are two steps for computing deterministic covers, which are *semantic branching*, and *branching postponement*. We will describe each in detail in the following.

Semantic Branching

Usually, the latter step (called syntactic branching) of DNF is achieved by applying recursively to the top level formulae the rewriting rule

$$f' \wedge (f_1 \vee f_2) \Rightarrow (f' \wedge f_1) \vee (f' \wedge f_2)$$

However, a major weakness of syntactic branching is that it generates subbranches which are not mutually inconsistent. In order to avoid this fact, they apply the Shannon expansion to the top level boolean propositions.

$$f \Rightarrow (p \wedge (f[\{p\}])) \vee (\neg p \wedge (f[\{\neg p\}]))$$

A formula f will be split into two parts where a proposition p holds in the first part and $\neg p$ holds in the second. This step is called *semantic branching* because it "semantically" splits on the truth values of top level propositions. Then we can obtain an expression in the form

$$\bigvee_i (\alpha_i \wedge f[\alpha_i])$$

After applying the technique semantic branching, all α_i's are pairwise mutually inconsistent and $f[\alpha_i]$ is a boolean combination of $\bigcirc$-formulae. If all $f[\alpha_i]$ are conjunctions of $\bigcirc$-formulae, then we have obtained a deterministic cover. Otherwise, the only possible sources of non-determinism (if any) are due to the next-part components $f[\alpha_i]$. Each non-deterministic disjunct represent a set of states S_i which have the same label of α_i but different next-part. For two states in different state set S_i's are mutually inconsistent.

Branching Postponement

If the correctness of the encoding will not be affected, each formula $f[\alpha_i]$ can be rewritten into single $\bigcirc$-formula by applying *branching postponement*:

$$\begin{aligned} \bigcirc f_1 \wedge \bigcirc f_2 &\Rightarrow \bigcirc(f_1 \wedge f_2), \\ \bigcirc f_1 \vee \bigcirc f_2 &\Rightarrow \bigcirc(f_1 \vee f_2). \end{aligned}$$

This step is called branching postponement because it allows for postponing the or-branching to the expansion of the next part. With branching postponement, the result expansion becomes deterministic.

However, branching postponement may cause some incorrectness. For example, for two states $s_1 = (\alpha, f_1)$, $s_2 = (\alpha, f_2)$, it may be the case that s_1 is in a fair set F_1 but s_2 is not, and the state corresponding to $(\alpha, f_1 \vee f_2)$ is not in F_1. The fairness set F_1 may be loosed, which cause the language may also be changed. The rewritten rule should be applied only to the formula for which is guaranteed that the rule does not cause incorrectness.

4.4.2 Algorithm

The standard schema of temporal formula to BA algorithm in MoDeLLa is described in Fig. 4.12. It differs from the others in two steps, which are the computation of covers fair sets and the computation of acceptance condition.

Computation of covers The function of computing covers of each formula is in the following steps:

1. Apply the tableau rules.

2. Apply the *semantic branching* step.

```
input  : temporal formula $f$
output: BA
$\Sigma := \{p \mid p \in f\}$;
$D := 2^{\Sigma}$;
$C(f) := expand(f)$;
$Q_0 := C(f)$;
$Q := C(f)$;                                                      // compute $Q$
$Queue.put(C(f))$;
while $Queue$ is not empty do
    $(\alpha \wedge X) := Queue.get()$ $C(next(X)) := expand(next(X))$;
    $Q := Q \cup C(next(X))$;
end
$T := \varnothing$;                                                   // compute $T$
foreach $(\alpha, X) \in Q$ do
    foreach $(\alpha', X') \in C(next(X))$ do
        $T := T \cup ((\alpha, X), (\alpha', X'))$;
    end
end
foreach $(\alpha, X) \in Q$ do
    $L(\alpha, X) := \{p \in D \mid p \wedge \alpha \not\models \bot\}$ ;            // compute $L$
end
$F := compute_f airset(f, Q)$;                                       // compute $F$
return $(Q, Q_0, T, L, D, F)$;
```

Figure 4.12: The regular schema of temporal logic to BA algorithm

3. Rewrite the formula into DNF form.

4. If postponement is safe, apply *branching postponement*.

The functions here tries to make the cover "more deterministic" as possible. During the computation of covers, the set of sub states of each state $s = (\alpha, X)$, $Sub(s)$, is a set of states where $Sub(s) = \{(\alpha', X') \mid \alpha = \alpha'\}$ is also collected. The set will be used in the computation of fair sets.

Computation of fair sets Let U_f be the set of $\mathcal{U}$-subformulae of f, the usual set of accepting condition is:

$$\begin{aligned} F^* &:= \{F^*_{g\,\mathcal{U}\,h} \mid g\,\mathcal{U}\,h \in U_f\}, \\ F^*_{g\,\mathcal{U}\,h} &:= \{s \in Q \mid s \not\models g\,\mathcal{U}\,h \text{ or } s \models h\}. \end{aligned}$$

The definition is extended here for MoDeLLa construction as follows:

$$\begin{aligned} F &:= \{F_H \mid H \in 2^{U_f}\}, \\ F_H &:= \{s \in Q \mid \text{ there exists } g\,\mathcal{U}\,h \in H \text{ s.t.} \\ &\quad\ \text{for each } s^* \in Subs(s), s^* \not\models g\,\mathcal{U}\,h \text{ or} \\ &\quad\ \text{for each } s^* \in Subs(s), s^* \models h\} \end{aligned}$$

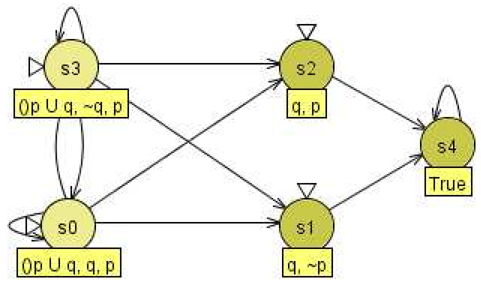


Figure 4.13: GBA_f generated by MoDeLLa

Note that the branching postponement is not safe for a state s if there exists $F_H \in F$ such that $s \notin F_H$ and there exist $g\,\mathcal{U}\,h \in H$, $s^* \in Subs(s)$ such that $s^* \in F_{g\,\mathcal{U}\,h}$. Fig. 4.12 shows the result of the construction of temporal formula $f = p\,\mathcal{U}\,q$. The gray-colored states are the accepting states.

Chapter 5

Experimental Results

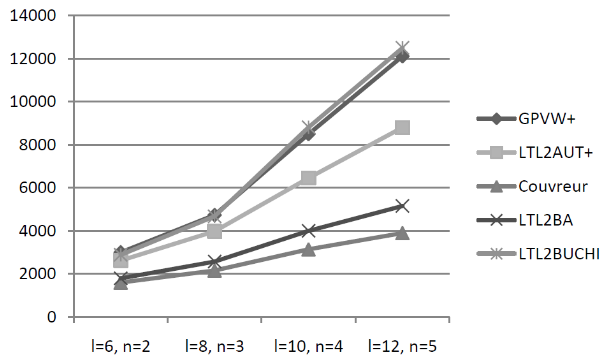
5.1 Settings of the Experiments

All of the algorithms described in Chapter 4 have been implemented in GOAL. We use the third stage of LTL2BA, which is an algorithm transform TGBA into BA, to translate the TGBA generated by Couvreur's algorithm since Couvreur's algorithm only generates TGBA as the result. We also notice the translation from TGBA to BA proposed by LTL2BA is different from the one proposed by LTL2BUCHI. From a simple comparison, we discover that the translation algorithm proposed by LTL2BA performs better. One of the reasons is that LTL2BA applies the on-the-fly simplification during the generation. Hence, we apply this construction to LTL2BUCHI and labeled as LTL2BUCHI+. Moreover, we imply the standard algorithms for label-on-state GBA to label-on-state GBA or BA for MoDeLLa. Optimization described in those works are also implemented.

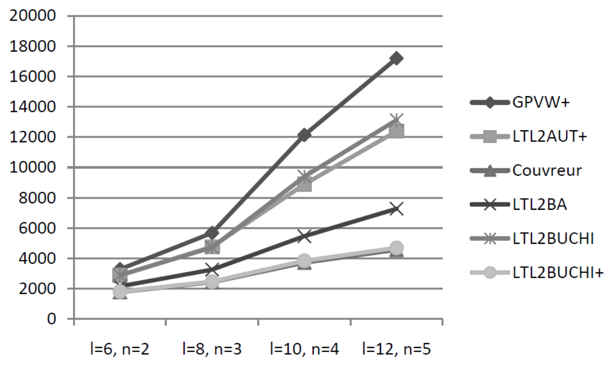
5.2 Results

We are going to present three sets of experimental results. The first one is the performance on state size for random generated formulae. The smaller size of automaton for input property always helps the performance of model checking procedure. Fig 5.1 shows the result of the experiment.

The second one is focus on the performance on state size and transition size for frequently used LTL formulae. There are two different kinds of categories for these formulae, one is proposed by Manna and Pnueli [17], another one is Spec Patterns [1]. We collect some frequently used formulae for both categories and design the following



(a) The state size of the result GBA/TGBA



(b) The state size of the result BA

Figure 5.1: The state size of different algorithms translating from 500 formulae. Note that l is the length of the formula and n is the number of propositions in the formula.

experiment. We compare the performance of the algorithms to see which of the algorithm can give a better result for specific categories. The comparison for the formulae categorized in Manna and Pnueli's temporal hierarchy is showed in Table 5.1 to Table 5.4. The comparison for the formulae categorized in Spec Patterns is showed in Table 5.5 to Table 5.10.

No	Formulae	GPVW+		DGV+		MoDeLLa	
		st.	tran.	st.	tran.	st.	tran.
Safety							
1	$\Box p$	2	2	2	2	2	2
2	$p \to \Box q$	4	14	4	14	8	24
3	$\Box(p \to q)$	3	12	3	12	4	12
4	$\Box p \wedge \Box q$	2	2	2	2	3	3
5	$\Box p \vee \Box q$	3	8	3	8	9	20
6	$\neg p \, \mathcal{W} \, q$	4	16	4	16	6	24
7	$\Box(p \to q) \wedge \Box(q \to r)$	4	24	4	24	5	20
8	$\Box(p \to \Box q) \wedge \Box(r \to \Box s)$	10	116	5	52	65	480
Guarantee							
9	$p \to \Diamond q$	5	26	5	26	15	79
10	$\Diamond(p \to r)$	5	28	5	28	9	51
11	$\Diamond p \vee \Diamond q$	6	36	6	36	25	139
12	$\Diamond(p \vee q)$	5	28	5	28	9	51
13	$\Diamond p \wedge \Diamond q$	10	58	10	58	23	141
14	$\Box p \to \Diamond q$	6	36	6	36	25	139
15	$\Diamond(p \to q) \wedge \Diamond(q \to r)$	16	220	16	220	78	1136
16	$p \to \Diamond(q \to r)$	6	68	6	68	31	351
Obligation							
17	$\Box p \vee \Diamond q$	5	24	5	24	18	84
18	$\Diamond p \to \Diamond q$	5	24	5	24	18	84
19	$(\Box p \vee \Diamond q) \vee (\Box r \vee \Diamond s)$	8	176	8	176	126	2316
20	$(\Box p \vee \Diamond q) \wedge (\Box r \vee \Diamond s)$	17	320	17	320	194	3824
21	$\Diamond r \to (p \, \mathcal{U} \, r)$	5	20	5	20	14	50
22	$\Box p \wedge \Diamond q$	4	10	4	10	9	23
23	$p \, \mathcal{U} \, q$	4	16	4	16	6	24
24	$(p \to \Box q) \vee (p \to \Diamond r)$	6	60	6	60	36	330
Recurence (Responce)							
25	$\Box \Diamond p$	4	14	3	9	7	21
26	$\Box(p \to \Diamond q)$	6	50	4	30	20	128
27	$\Box \Diamond p \vee \Box \Diamond q$	7	56	5	36	37	228
28	$\Box \Diamond(p \vee q)$	4	32	4	32	15	105
29	$\Box \Diamond p \wedge \Box \Diamond q$	10	130	5	45	37	333
30	$\Box(q \to \Box(p \to \Diamond s))$	12	292	5	84	45	602
31	$\Box \Diamond(p \wedge q \to r)$	11	292	5	100	31	465
32	$\Diamond(p \to \Box \Diamond q)$	9	68	6	44	26	180
Persistence							
33	$\Diamond \Box p$	4	8	3	7	5	11
34	$\Box(p \to \Diamond \Box q)$	9	61	4	24	25	144
35	$\Diamond \Box p \wedge \Diamond \Box q$	10	34	5	25	26	122
36	$\Diamond \Box(p \wedge q)$	4	12	3	11	7	27
37	$\Diamond \Box p \vee \Diamond \Box q$	7	32	5	28	29	132
38	$\neg \Box \Diamond p$	4	8	3	7	5	11
39	$\Diamond(p \, \mathcal{W} \, q)$	7	36	5	28	14	76
40	$\Diamond(p \to \Box q)$	6	28	5	26	12	60
Reactivity							
41	$\Box \Diamond p \vee \Diamond \Box q$	7	44	5	32	33	180
42	$\Box \Diamond p \to \Box \Diamond q$	7	44	5	32	33	180
43	$(\Box \Diamond p \vee \Diamond \Box q) \wedge (\Box \Diamond r \vee \Diamond \Box s)$	31	848	17	544	581	16064
44	$\Box(p \to \Diamond q) \vee \Box(p \to \Diamond \Box r)$	14	222	7	108	117	1462
45	$\Diamond \Box p \to \Diamond \Box q$	6	34	5	32	33	180
46	$(\Diamond \Box p \to \Diamond \Box q) \wedge \Box(p \to \Diamond \Box r)$	38	567	12	180	295	4674
47	$(\Box \Diamond p \to \Box \Diamond q) \wedge \Box(r \to \Diamond s)$	31	1440	13	536	345	11596
48	$\Box(r \to \Diamond s) \wedge (\Box \Diamond r \vee \Diamond \Box s)$	23	227	11	107	89	759

Table 5.1: A comparison of the generated GBA of GPVW+, LTL2AUT+, and MoDeLLa

No	Formulae	Couvreur		LTL2BA		LTL2Buchi	
		st.	tran.	st.	tran.	st.	tran.
Safety							
1	$\Box p$	1	1	1	1	2	2
2	$p \to \Box q$	3	10	3	10	4	14
3	$\Box(p \to q)$	1	3	1	3	3	12
4	$\Box p \wedge \Box q$	2	2	1	1	2	2
5	$\Box p \vee \Box q$	3	8	3	8	3	8
6	$\neg p\ \mathcal{W}\ q$	2	8	2	8	4	16
7	$\Box(p \to q) \wedge \Box(q \to r)$	2	8	1	4	4	24
8	$\Box(p \to \Box q) \wedge \Box(r \to \Box s)$	5	52	4	36	10	116
Guarantee							
9	$p \to \Diamond q$	3	17	3	17	5	26
10	$\Diamond(p \to r)$	2	11	2	11	5	28
11	$\Diamond p \vee \Diamond q$	4	27	4	27	6	36
12	$\Diamond(p \vee q)$	2	11	2	11	5	28
13	$\Diamond p \wedge \Diamond q$	5	34	7	46	10	58
14	$\Box p \to \Diamond q$	4	27	4	27	6	36
15	$\Diamond(p \to q) \wedge \Diamond(q \to r)$	5	84	7	112	16	220
16	$p \to \Diamond(q \to r)$	3	37	3	37	6	68
Obligation							
17	$\Box p \vee \Diamond q$	4	20	4	20	5	24
18	$\Diamond p \to \Diamond q$	4	20	4	20	5	24
19	$(\Box p \vee \Diamond q) \vee (\Box r \vee \Diamond s)$	6	140	6	140	8	176
20	$(\Box p \vee \Diamond q) \wedge (\Box r \vee \Diamond s)$	10	208	14	272	17	320
21	$\Diamond r \to (p\ \mathcal{U}\ r)$	4	16	4	16	5	20
22	$\Box p \wedge \Diamond q$	3	8	4	10	4	10
23	$p\ \mathcal{U}\ q$	2	8	2	8	4	16
24	$(p \to \Box q) \vee (p \to \Diamond r)$	4	42	4	42	6	60
Recurence (Responce)							
25	$\Box\Diamond p$	1	2	3	8	4	14
26	$\Box(p \to \Diamond q)$	2	13	4	28	7	63
27	$\Box\Diamond p \vee \Box\Diamond q$	3	16	7	44	7	56
28	$\Box\Diamond(p \vee q)$	1	4	3	18	6	60
29	$\Box\Diamond p \wedge \Box\Diamond q$	2	8	5	36	10	130
30	$\Box(q \to \Box(p \to \Diamond s))$	3	44	5	74	15	489
31	$\Box\Diamond(p \wedge q \to r)$	1	8	3	38	11	292
32	$\Diamond(p \to \Box\Diamond q)$	3	18	5	32	7	54
Persistence							
33	$\Diamond\Box p$	2	4	2	4	3	7
34	$\Box(p \to \Diamond\Box q)$	4	24	3	16	8	60
35	$\Diamond\Box p \wedge \Diamond\Box q$	5	25	4	16	5	25
36	$\Diamond\Box(p \wedge q)$	2	6	2	6	3	11
37	$\Diamond\Box p \vee \Diamond\Box q$	5	28	5	28	5	28
38	$\neg\Box\Diamond p$	2	4	2	4	3	7
39	$\Diamond(p\ \mathcal{W}\ q)$	3	16	3	16	5	28
40	$\Diamond(p \to \Box q)$	3	14	3	14	5	26
Reactivity							
41	$\Box\Diamond p \vee \Diamond\Box q$	4	22	6	36	6	42
42	$\Box\Diamond p \to \Box\Diamond q$	4	22	6	36	6	42
43	$(\Box\Diamond p \vee \Diamond\Box q) \wedge (\Box\Diamond r \vee \Diamond\Box s)$	10	244	22	608	26	1044
44	$\Box(p \to \Diamond q) \vee \Box(p \to \Diamond\Box r)$	7	104	8	118	14	246
45	$\Diamond\Box p \to \Diamond\Box q$	4	11	6	36	6	42
46	$(\Diamond\Box p \to \Diamond\Box q) \wedge \Box(p \to \Diamond\Box r)$	13	184	16	236	26	504
47	$(\Box\Diamond p \to \Box\Diamond q) \wedge \Box(r \to \Diamond s)$	7	226	15	520	31	1806
48	$\Box(r \to \Diamond s) \wedge (\Box\Diamond r \vee \Diamond\Box s)$	9	76	14	121	23	328

Table 5.2: A comparison of the generated TGBA of Couvreur, LTL2BA, LTL2BUCHI

No	Formulae	GPVW+		DGV+		MoDeLLa		Couvreur		LTL2BA		LTL2Buchi		LTL2Buchi+		SPIN	
		st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.
Safety																	
1	$\Box p$	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1
2	$p \rightarrow \Box q$	4	14	4	14	8	24	3	10	3	10	4	14	3	10	3	10
3	$\Box(p \rightarrow q)$	3	12	3	12	4	12	2	6	2	6	3	12	2	6	1	3
4	$\Box p \wedge \Box q$	2	2	2	2	3	3	2	2	2	2	2	2	2	2	1	1
5	$\Box p \vee \Box q$	3	8	3	8	9	20	3	8	3	8	3	8	3	8	3	8
6	$\neg p\ \mathcal{W}\ q$	4	16	4	16	6	24	3	12	3	12	4	16	3	12	4	16
Guarantee																	
7	$\Box(p \rightarrow q) \wedge \Box(q \rightarrow r)$	4	24	4	24	5	20	2	8	2	8	4	24	2	8	1	4
8	$\Box(p \rightarrow \Box q) \wedge \Box(r \rightarrow \Box s)$	10	116	5	52	65	480	5	52	5	52	10	116	5	52	4	72
9	$p \rightarrow \Diamond q$	5	26	5	26	15	79	3	17	3	17	5	26	3	17	3	17
10	$\Diamond(p \rightarrow r)$	5	28	5	28	9	51	2	11	2	11	5	28	2	11	2	11
11	$\Diamond p \vee \Diamond q$	9	50	9	50	33	179	4	27	4	27	6	36	4	27	2	11
12	$\Diamond(p \vee q)$	5	28	5	28	9	51	2	11	2	11	5	28	2	11	2	11
13	$\Diamond p \wedge \Diamond q$	11	62	11	62	31	181	4	25	9	58	10	58	4	25	4	25
14	$\Box p \rightarrow \Diamond q$	9	50	9	50	33	179	4	27	4	27	6	36	4	27	2	11
15	$\Diamond(p \rightarrow q) \wedge \Diamond(q \rightarrow r)$	20	260	17	228	94	1312	4	60	9	140	16	220	4	60	4	60
16	$p \rightarrow \Diamond(q \rightarrow r)$	6	68	6	68	31	351	3	37	3	37	6	68	3	37	3	37
Obligation																	
17	$\Box p \vee \Diamond q$	5	24	5	24	18	84	4	20	4	20	5	24	4	20	4	20
18	$\Diamond p \rightarrow \Diamond q$	5	24	5	24	18	84	4	20	4	20	5	24	4	20	4	20
19	$(\Box p \vee \Diamond q) \vee (\Box r \vee \Diamond s)$	13	248	13	248	184	3116	6	140	6	140	8	176	6	140	6	140
20	$(\Box p \vee \Diamond q) \wedge (\Box r \vee \Diamond s)$	23	376	23	376	272	4784	10	208	16	320	17	320	10	208	10	208
21	$\Diamond r \rightarrow (p\ \mathcal{U}\ r)$	5	20	5	20	14	50	4	16	4	16	5	20	4	16	4	32
22	$\Box p \wedge \Diamond q$	4	10	4	10	9	23	2	5	4	10	4	10	2	5	2	5
23	$p\ \mathcal{U}\ q$	4	16	4	16	6	24	2	8	2	8	4	16	2	8	2	8
24	$(p \rightarrow \Box q) \vee (p \rightarrow \Diamond r)$	6	60	6	60	36	330	4	42	4	42	6	60	4	42	4	42

Table 5.3: A comparison of the result BA of all six algorithms and SPIN (part a)

No	Formulae	GPVW+		DGV+		MoDeLLa		Couvreur		LTL2BA		LTL2Buchi		LTL2Buchi+		SPIN	
		st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.
Recurence (Responce)																	
25	$\Box\Diamond p$	4	14	3	9	7	21	2	4	5	13	4	14	4	14	2	5
26	$\Box(p \rightarrow \Diamond q)$	6	50	4	30	20	128	3	20	6	41	7	63	4	30	4	28
27	$\Box\Diamond p \vee \Box\Diamond q$	13	100	9	60	73	444	5	24	9	52	7	56	7	56	5	32
28	$\Box\Diamond(p \vee q)$	4	32	4	32	15	105	2	8	5	29	6	60	4	34	2	11
29	$\Box\Diamond p \wedge \Box\Diamond q$	19	251	9	81	91	819	3	12	9	60	13	174	8	82	3	17
30	$\Box(q \rightarrow \Box(p \rightarrow \Diamond s))$	12	292	5	84	45	602	4	62	7	104	15	489	5	86	6	100
31	$\Box\Diamond(p \wedge q \rightarrow r)$	11	292	5	100	31	465	2	16	5	61	11	292	4	74	2	23
32	$\Diamond(p \rightarrow \Box\Diamond q)$	17	124	11	76	46	296	4	22	8	54	7	54	5	38	4	26
Persistence																	
33	$\Diamond\Box p$	4	8	3	7	5	11	2	4	2	4	3	7	2	4	2	4
34	$\Box(p \rightarrow \Diamond\Box q)$	9	61	4	24	25	144	5	32	4	24	8	60	4	24	4	24
35	$\Diamond\Box p \wedge \Diamond\Box q$	11	35	7	29	28	124	4	16	4	16	5	25	4	16	4	16
36	$\Diamond\Box(p \wedge q)$	4	12	3	11	7	27	2	6	2	6	3	11	2	6	2	6
37	$\Diamond\Box p \vee \Diamond\Box q$	11	44	8	38	43	176	5	28	5	28	5	28	5	28	3	12
38	$\neg\Box\Diamond p$	4	8	3	7	5	11	2	4	2	4	3	7	2	4	2	4
39	$\Diamond(p\ \mathcal{W}\, q)$	7	36	5	28	14	76	3	16	3	16	5	28	3	16	4	20
40	$\Diamond(p \rightarrow \Box q)$	6	28	5	26	12	60	3	14	3	14	5	26	3	14	3	14
Reactivity																	
41	$\Box\Diamond p \vee \Diamond\Box q$	13	76	9	52	61	324	5	26	7	40	6	42	6	42	5	30
42	$\Box\Diamond p \rightarrow \Box\Diamond q$	13	76	9	52	55	296	5	26	7	40	6	42	6	42	5	30
43	$(\Box\Diamond p \vee \Diamond\Box q) \wedge (\Box\Diamond r \vee \Diamond\Box s)$	77	2064	50	1300	4225	117776	14	292	29	732	31	1284	22	732	15	388
44	$\Box(p \rightarrow \Diamond q) \vee \Box(p \rightarrow \Diamond\Box r)$	24	360	13	184	229	2806	7	104	9	130	14	246	7	108	9	134
45	$\Diamond\Box p \rightarrow \Diamond\Box q$	9	48	8	46	61	324	3	7	7	40	6	42	6	42	5	30
46	$(\Diamond\Box p \rightarrow \Diamond\Box q) \wedge \Box(p \rightarrow \Diamond\Box r)$	88	1288	31	400	1373	21186	16	208	20	276	29	568	12	174	16	218
47	$(\Box\Diamond p \rightarrow \Box\Diamond q) \wedge \Box(r \rightarrow \Diamond s)$	75	3622	34	1284	1275	43568	9	278	22	708	36	2198	13	526	19	630
48	$\Box(r \rightarrow \Diamond s) \wedge (\Box\Diamond r \vee \Diamond\Box s)$	47	441	23	213	396	3310	11	95	18	149	29	434	13	146	13	112

Table 5.4: A comparison of the result BA of all six algorithms and SPIN (part b)

No	Formulae	GPVW+		DGV+		MoDeLLa	
		st.	tran.	st.	tran.	st.	tran.
Existence (P becomes true)							
1	$\Diamond(P)$	4	10	4	10	5	13
2	$\neg R\,\mathcal{W}\,(P \wedge \neg R)$	4	14	4	14	5	17
3	$\Box(\neg Q) \vee \Diamond(Q \wedge \Diamond P))$	8	42	8	42	26	141
4	$\Box(Q \wedge \neg R \rightarrow (\neg R\,\mathcal{W}\,(P \wedge \neg R)))$	8	103	5	62	18	158
5	$\Box(Q \wedge \neg R \rightarrow (\neg R\,\mathcal{U}\,(P \wedge \neg R)))$	7	88	5	62	18	158
Bounded Existence (P becomes true twice)							
6	$(\neg P\,\mathcal{W}\,(P\,\mathcal{W}\,(\neg P\,\mathcal{W}\,(P\,\mathcal{W}\,\Box\neg P))))$	6	20	6	20	454	4840
7	$\Diamond R \rightarrow ((\neg P \wedge \neg R)\,\mathcal{U}\,(R \vee ((P \wedge \neg R)\,\mathcal{U}$ $(R \vee ((\neg P \wedge \neg R)\,\mathcal{U}\,(R \vee ((P \wedge \neg R)\,\mathcal{U}\,(R \vee (\neg P\,\mathcal{U}\,R)))))))))$	33	204	9	50	44	204
8	$\Diamond Q \rightarrow (\neg Q\,\mathcal{U}\,(Q \wedge (\neg P\,\mathcal{W}\,(P\,\mathcal{W}\,(\neg P\,\mathcal{W}\,(P\,\mathcal{W}\,\Box\neg P))))))$	13	78	13	78	977	21223
9	$\Box((Q \wedge \Diamond R) \rightarrow ((\neg P \wedge \neg R)\,\mathcal{U}\,(R \vee ((P \wedge \neg R)\,\mathcal{U}$ $(R \vee ((\neg P \wedge \neg R)\,\mathcal{U}\,(R \vee ((P \wedge \neg R)\,\mathcal{U}\,(R \vee (\neg P\,\mathcal{U}\,R))))))))))$	157	5361	9	136	421	6318
10	$\Box(Q \rightarrow ((\neg P \wedge \neg R)\,\mathcal{U}\,(R \vee ((P \wedge \neg R)\,\mathcal{U}\,(R \vee ((\neg P \wedge \neg R)\,\mathcal{U}$ $(R \vee ((P \wedge \neg R)\,\mathcal{U}\,(R \vee (\neg P\,\mathcal{W}\,R) \vee \Box P)))))))))$	115	4203	9	152	351	6308
Universality (P is always true)							
11	$\Box(P)$	2	2	2	2	2	2
12	$\Diamond R \rightarrow (P\,\mathcal{U}\,R)$	5	20	5	20	14	50
13	$\Box(Q \rightarrow \Box(P))$	4	14	3	10	7	20
14	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (P\,\mathcal{U}\,R))$	12	170	5	60	33	294
15	$\Box(Q \wedge \neg R \rightarrow (P\,\mathcal{W}\,R))$	7	90	4	44	23	252
Precedence (S precedes P)							
16	$\neg P\,\mathcal{W}\,S$	4	16	4	16	7	32
17	$\Diamond R \rightarrow (\neg P\,\mathcal{U}\,(S \vee R))$	6	56	6	56	30	230
18	$\Box\neg Q \vee \Diamond(Q \wedge (\neg P\,\mathcal{W}\,S))$	8	72	8	72	46	449
19	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (\neg P\,\mathcal{U}\,(S \vee R)))$	17	644	6	192	79	1642
20	$\Box(Q \wedge \neg R \rightarrow (\neg P\,\mathcal{W}\,(S \vee R)))$	11	416	5	152	55	1484
Response (S responds to P)							
21	$\Box(P \rightarrow \Diamond S)$	6	50	4	30	14	90
22	$\Diamond R \rightarrow (P \rightarrow (\neg R\,\mathcal{U}\,(S \wedge \neg R)))\,\mathcal{U}\,R$	9	98	7	72	42	349
23	$\Box(Q \rightarrow \Box(P \rightarrow \Diamond S))$	12	292	5	84	45	602
24	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (P \rightarrow (\neg R\,\mathcal{U}\,(S \wedge \neg R)))\,\mathcal{U}\,R)$	28	1182	7	208	151	3530
25	$\Box(Q \wedge \neg R \rightarrow ((P \rightarrow (\neg R\,\mathcal{U}\,(S \wedge \neg R)))\,\mathcal{W}\,R))$	15	574	6	176	77	1962

Table 5.5: A comparison of the generated GBA of GPVW+, LTL2AUT+, and MoDeLLa (part a)

No	Formulae	GPVW+		DGV+		MoDeLLa	
		st.	tran.	st.	tran.	st.	tran.
Precedence Chain (S, T precedes P)							
26	$\Diamond P \rightarrow (\neg P\, \mathcal{U}\, (S \wedge \neg P \wedge X(\neg P\, \mathcal{U} T)))$	7	52	7	52	30	194
27	$\Diamond R \rightarrow (\neg P\, \mathcal{U}\, (R \vee (S \wedge \neg P \wedge X(\neg P\, \mathcal{U} T))))$	8	136	8	136	74	1132
28	$(\Box\neg Q) \vee (\neg Q\, \mathcal{U}\, (Q \wedge \Diamond P \rightarrow (\neg P\, \mathcal{U}\, (S \wedge \neg P \wedge X(\neg P\, \mathcal{U} T)))))$	13	224	10	188	126	1948
29	$\Box((Q \wedge \Diamond R) \rightarrow (\neg P\, \mathcal{U}\, (R \vee (S \wedge \neg P \wedge X(\neg P\, \mathcal{U} T)))))$	40	2346	15	876	383	16276
30	$\Box(Q \rightarrow (\Diamond P \rightarrow (\neg P\, \mathcal{U}\, (R \vee (S \wedge \neg P \wedge X(\neg P\, \mathcal{U} T))))))$	46	3576	15	916	471	26052
Precedence Chain (P precedes S, T)							
31	$(\Diamond(S \wedge X \Diamond T)) \rightarrow ((\neg S)\, \mathcal{U} P)$	8	74	7	64	46	332
32	$\Diamond R \rightarrow ((\neg(S \wedge (\neg R) \wedge X(\neg R\, \mathcal{U}\, (T \wedge \neg R))))\, \mathcal{U}\, (R \vee P))$	27	1104	17	652	156	4224
33	$(\Box\neg Q) \vee ((\neg Q)\, \mathcal{U}\, (Q \wedge ((\Diamond(S \wedge X \Diamond T)) \rightarrow ((\neg S)\, \mathcal{U} P))))$	14	248	13	224	122	1844
34	$\Box((Q \wedge \Diamond R) \rightarrow$ $((\neg(S \wedge (\neg R) \wedge X(\neg R\, \mathcal{U}\, (T \wedge \neg R))))\, \mathcal{U}\, (R \vee P)))$	62	9384	20	2312	731	51848
35	$\Box(Q \rightarrow (\neg(S \wedge (\neg R) \wedge X(\neg R\, \mathcal{U}\, (T \wedge \neg R)))\, \mathcal{U}\, (R \vee P) \vee \Box(\neg(S \wedge X \Diamond T))))$	-	-	28	3120	-	-
Response Chain (P responds S, T)							
36	$\Box(S \wedge X \Diamond T \rightarrow X(\Diamond(T \wedge \Diamond P)))$	40	779	28	516	143	2384
37	$\Diamond R \rightarrow (S \wedge X(\neg R\, \mathcal{U} T) \rightarrow X(\neg R\, \mathcal{U}\, (T \wedge \Diamond P)))\, \mathcal{U} R$	61	1804	67	2357	435	14020
38	$\Box(Q \rightarrow \Box(S \wedge X \Diamond T \rightarrow X(\neg T\, \mathcal{U}\, (T \wedge \Diamond P))))$	80	3870	29	1030	451	14984
39	$\Box((Q \wedge S \Diamond R) \rightarrow$ $(S \wedge X(\neg R\, \mathcal{U} T) \rightarrow X(\neg R\, \mathcal{U}\, (T \wedge \Diamond P)))\, \mathcal{U} R)$	-	-	75	6998	-	-
40	$\Box(Q \rightarrow (S \wedge X(\neg R\, \mathcal{U} T) \rightarrow X(\neg R\, \mathcal{U}\, (T \wedge \Diamond P)))\, \mathcal{U}$ $(R \vee \Box(S \wedge X(\neg R\, \mathcal{U} T) \rightarrow X(\neg R\, \mathcal{U}\, (T \wedge \Diamond P)))))$	-	-	117	15234	-	-
Response Chain (S, T responds P)							
41	$\Box(P \rightarrow \Diamond(S \wedge X \Diamond T))$	16	342	9	162	69	1176
42	$\Diamond R \rightarrow (P \rightarrow (\neg R\, \mathcal{U}\, (S \wedge \neg R \wedge X(\neg R\, \mathcal{U} T))))\, \mathcal{U} R$	20	484	13	280	132	2464
43	$\Box(Q \rightarrow \Box(P \rightarrow (S \wedge X \Diamond T)))$	14	432	7	152	57	984
44	$\Box((Q \wedge \Diamond R) \rightarrow (P \rightarrow (\neg R\, \mathcal{U}\, (S \wedge \neg R \wedge X(\neg R\, \mathcal{U} T))))\, \mathcal{U} R)$	68	6252	13	744	589	31392
45	$\Box(Q \rightarrow (P \rightarrow (\neg R\, \mathcal{U}\, (S \wedge \neg R \wedge X(\neg R\, \mathcal{U} T))))\, \mathcal{U}$ $(R \vee \Box(P \rightarrow (S \wedge X \Diamond T))))$	-	-	35	1962	-	-
Constrained Chain Patterns (S, T without Z responds to P)							
46	$\Box(P \rightarrow \Diamond(S \wedge \neg Z \wedge X((\neg Z)\, \mathcal{U} T)))$	16	480	9	236	99	2384
47	$\Diamond R \rightarrow$ $(P \rightarrow (\neg R\, \mathcal{U}\, (S \wedge \neg R \wedge \neg Z \wedge X((\neg R \wedge \neg Z)\, \mathcal{U} T))))\, \mathcal{U} R$	20	764	13	472	220	6744
48	$\Box(Q \rightarrow \Box(P \rightarrow (S \wedge \neg Z \wedge X((\neg Z)\, \mathcal{U} T))))$	14	560	7	212	105	2632
49	$\Box((Q \wedge \Diamond R) \rightarrow$ $(P \rightarrow (\neg R\, \mathcal{U}\, (S \wedge \neg R \wedge \neg Z \wedge X((\neg R \wedge \neg Z)\, \mathcal{U} T))))\, \mathcal{U} R)$	68	9240	13	1288	-	-
50	$\Box(Q \rightarrow (P \rightarrow (\neg R\, \mathcal{U}\, (S \wedge \neg R \wedge \neg Z \wedge X((\neg R \wedge \neg Z)\, \mathcal{U} T))))\, \mathcal{U}$ $(R \vee \Box(P \rightarrow (S \wedge \neg Z \wedge X((\neg Z)\, \mathcal{U} T)))))$	-	-	35	2734	-	-

Table 5.6: A comparison of the generated GBA of GPVW+, LTL2AUT+, and MoDeLLa (part b)

No	Formulae	Couvreur		LTL2BA		LTL2BUCHI	
		st.	tran.	st.	tran.	st.	tran.
Existence (P becomes true)							
1	$\Diamond(P)$	2	5	2	5	4	10
2	$\neg R\,\mathcal{W}\,(P \wedge \neg R)$	2	7	2	7	4	14
3	$\Box(\neg Q) \vee \Diamond(Q \wedge \Diamond P))$	5	28	5	28	8	42
4	$\Box(Q \wedge \neg R \rightarrow (\neg R\,\mathcal{W}\,(P \wedge \neg R)))$	2	17	4	36	8	103
5	$\Box(Q \wedge \neg R \rightarrow (\neg R\,\mathcal{U}\,(P \wedge \neg R)))$	2	17	4	36	8	103
Bounded Existence (P becomes true twice)							
6	$(\neg P\,\mathcal{W}\,(P\,\mathcal{W}\,(\neg P\,\mathcal{W}\,(P\,\mathcal{W}\,\Box\neg P))))$	5	15	5	15	6	20
7	$\Diamond R \rightarrow ((\neg P \wedge \neg R)\,\mathcal{U}\,(R \vee ((P \wedge \neg R)\,\mathcal{U}$ $(R \vee ((\neg P \wedge \neg R)\,\mathcal{U}\,(R \vee ((P \wedge \neg R)\,\mathcal{U}\,(R \vee (\neg P\,\mathcal{U}\,R))))))))$	8	46	8	46	9	50
8	$\Diamond Q \rightarrow (\neg Q\,\mathcal{U}\,(Q \wedge (\neg P\,\mathcal{W}\,(P\,\mathcal{W}\,(\neg P\,\mathcal{W}\,(P\,\mathcal{W}\,\Box\neg P))))))$	8	48	8	48	13	78
9	$\Box((Q \wedge \Diamond R) \rightarrow ((\neg P \wedge \neg R)\,\mathcal{U}\,(R \vee ((P \wedge \neg R)\,\mathcal{U}$ $(R \vee ((\neg P \wedge \neg R)\,\mathcal{U}\,(R \vee ((P \wedge \neg R)\,\mathcal{U}\,(R \vee (\neg P\,\mathcal{U}\,R)))))))))$	22	716	24	322	62	1616
10	$\Box(Q \rightarrow ((\neg P \wedge \neg R)\,\mathcal{U}\,(R \vee ((P \wedge \neg R)\,\mathcal{U}\,(R \vee ((\neg P \wedge \neg R)\,\mathcal{U}$ $(R \vee ((P \wedge \neg R)\,\mathcal{U}\,(R \vee (\neg P\,\mathcal{W}\,R) \vee \Box P)))))))))$	22	456	15	244	-	-
Universality (P is always true)							
11	$\Box(P)$	1	1	1	1	2	2
12	$\Diamond R \rightarrow (P\,\mathcal{U}\,R)$	4	16	4	16	5	20
13	$\Box(Q \rightarrow \Box(P))$	2	6	2	6	4	14
14	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (P\,\mathcal{U}\,R))$	4	46	8	66	10	112
15	$\Box(Q \wedge \neg R \rightarrow (P\,\mathcal{W}\,R))$	2	18	4	38	6	72
Precedence (S precedes P)							
16	$\neg P\,\mathcal{W}\,S$	2	8	2	8	4	16
17	$\Diamond R \rightarrow (\neg P\,\mathcal{U}\,(S \vee R))$	4	36	4	36	6	56
18	$\Box\neg Q \vee \Diamond(Q \wedge (\neg P\,\mathcal{W}\,S))$	5	48	5	48	8	72
19	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (\neg P\,\mathcal{U}\,(S \vee R)))$	4	104	8	164	19	702
20	$\Box(Q \wedge \neg R \rightarrow (\neg P\,\mathcal{W}\,(S \vee R)))$	2	42	4	90	11	416
Response (S responds to P)							
21	$\Box(P \rightarrow \Diamond S)$	2	13	4	28	6	54
22	$\Diamond R \rightarrow (P \rightarrow (\neg R\,\mathcal{U}\,(S \wedge \neg R)))\,\mathcal{U}\,R$	5	48	7	69	10	113
23	$\Box(Q \rightarrow \Box(P \rightarrow \Diamond S))$	3	44	5	74	14	448
24	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (P \rightarrow (\neg R\,\mathcal{U}\,(S \wedge \neg R)))\,\mathcal{U}\,R)$	9	282	12	316	33	1584
25	$\Box(Q \wedge \neg R \rightarrow ((P \rightarrow (\neg R\,\mathcal{U}\,(S \wedge \neg R)))\,\mathcal{W}\,R))$	6	148	6	144	18	781

Table 5.7: A comparison of the generated TGBA of Couvreur, LTL2BA, and LTL2BUCHI (part a)

No	Formulae	Couvreur		LTL2BA		LTL2BUCHI	
		st.	tran.	st.	tran.	st.	tran.
Precedence Chain (S, T precedes P)							
26	$\Diamond P \rightarrow (\neg P\,\mathcal{U}\,(S \wedge \neg P \wedge X(\neg P\,\mathcal{U}\,T)))$	5	36	5	36	7	52
27	$\Diamond R \rightarrow (\neg P\,\mathcal{U}\,(R \vee (S \wedge \neg P \wedge X(\neg P\,\mathcal{U}\,T))))$	5	88	5	88	8	136
28	$(\Box\neg Q) \vee (\neg Q\,\mathcal{U}\,(Q \wedge \Diamond P \rightarrow (\neg P\,\mathcal{U}\,(S \wedge \neg P \wedge X(\neg P\,\mathcal{U}\,T)))))$	7	140	7	140	10	188
29	$\Box((Q \wedge \Diamond R) \rightarrow (\neg P\,\mathcal{U}\,(R \vee (S \wedge \neg P \wedge X(\neg P\,\mathcal{U}\,T)))))$	11	522	16	716	52	3553
30	$\Box(Q \rightarrow (\Diamond P \rightarrow (\neg P\,\mathcal{U}\,(R \vee (S \wedge \neg P \wedge X(\neg P\,\mathcal{U}\,T))))))$	8	470	16	1010	64	5776
Precedence Chain (P precedes S, T)							
31	$(\Diamond(S \wedge X\Diamond T)) \rightarrow ((\neg S)\,\mathcal{U}\,P)$	5	52	5	52	7	64
32	$\Diamond R \rightarrow ((\neg(S \wedge (\neg R) \wedge X(\neg R\,\mathcal{U}\,(T \wedge \neg R))))\,\mathcal{U}\,(R \vee P))$	6	170	8	256	22	844
33	$(\Box\neg Q) \vee ((\neg Q)\,\mathcal{U}\,(Q \wedge ((\Diamond(S \wedge X\Diamond T)) \rightarrow ((\neg S)\,\mathcal{U}\,P))))$	7	136	7	136	13	224
34	$\Box((Q \wedge \Diamond R) \rightarrow$ $((\neg(S \wedge (\neg R) \wedge X(\neg R\,\mathcal{U}\,(T \wedge \neg R))))\,\mathcal{U}\,(R \vee P)))$	12	1158	14	950	76	15736
35	$\Box(Q \rightarrow (\Diamond P \rightarrow (\neg P\,\mathcal{U}\,(R \vee (S \wedge \neg P \wedge X(\neg P\,\mathcal{U}\,T))))))$	12	1334	21	2582	185	49210
Response Chain (P responds S, T)							
36	$(\Diamond(S \wedge X\Diamond T)) \rightarrow ((\neg S)\,\mathcal{U}\,P)$	8	142	16	291	31	529
37	$\Diamond R \rightarrow ((\neg(S \wedge (\neg R) \wedge X(\neg R\,\mathcal{U}\,(T \wedge \neg R))))\,\mathcal{U}\,(R \vee P))$	14	539	22	772	59	2037
38	$\Box(Q \rightarrow \Box(S \wedge X\Diamond T \rightarrow X(\neg T\,\mathcal{U}\,(T \wedge \Diamond P))))$	9	372	17	710	80	4705
39	$\Box((Q \wedge S\Diamond R) \rightarrow$ $(S \wedge X(\neg R\,\mathcal{U}\,T) \rightarrow X(\neg R\,\mathcal{U}\,(T \wedge \Diamond P)))\,\mathcal{U}\,R)$	44	5292	42	3762	244	36832
40	$\Box(Q \rightarrow (S \wedge X(\neg R\,\mathcal{U}\,T) \rightarrow X(\neg R\,\mathcal{U}\,(T \wedge \Diamond P)))\,\mathcal{U}$ $(R \vee \Box(S \wedge X(\neg R\,\mathcal{U}\,T) \rightarrow X(\neg R\,\mathcal{U}\,(T \wedge \Diamond P)))))$	56	11222	41	6492	-	-
Response Chain (S, T responds P)							
41	$\Box(P \rightarrow \Diamond(S \wedge X\Diamond T))$	4	68	8	142	19	469
42	$\Diamond R \rightarrow (P \rightarrow (\neg R\,\mathcal{U}\,(S \wedge \neg R \wedge X(\neg R\,\mathcal{U}\,T))))\,\mathcal{U}\,R$	9	186	11	238	23	611
43	$\Box(Q \rightarrow \Box(P \rightarrow (S \wedge X\Diamond T)))$	3	56	5	92	17	622
44	$\Box((Q \wedge \Diamond R) \rightarrow (P \rightarrow (\neg R\,\mathcal{U}\,(S \wedge \neg R \wedge X(\neg R\,\mathcal{U}\,T))))\,\mathcal{U}\,R)$	17	1232	20	1228	88	10672
45	$\Box(Q \rightarrow (P \rightarrow (\neg R\,\mathcal{U}\,(S \wedge \neg R \wedge X(\neg R\,\mathcal{U}\,T))))\,\mathcal{U}$ $(R \vee \Box(P \rightarrow (S \wedge X\Diamond T))))$	29	2968	31	2614	347	90607
Constrained Chain Patterns (S, T without Z responds to P)							
46	$\Box(P \rightarrow \Diamond(S \wedge \neg Z \wedge X((\neg Z)\,\mathcal{U}\,t)))$	4	98	8	202	19	625
47	$\Diamond R \rightarrow$ $(P \rightarrow (\neg R\,\mathcal{U}\,(S \wedge \neg R \wedge \neg Z \wedge X((\neg R \wedge \neg Z)\,\mathcal{U}\,T))))\,\mathcal{U}\,R$	8	298	11	390	23	909
48	$\Box(Q \rightarrow \Box(P \rightarrow (S \wedge \neg Z \wedge X((\neg Z)\,\mathcal{U}\,T))))$	3	84	5	132	17	768
49	$\Box((Q \wedge \Diamond R) \rightarrow$ $(P \rightarrow (\neg R\,\mathcal{U}\,(S \wedge \neg R \wedge \neg Z \wedge X((\neg R \wedge \neg Z)\,\mathcal{U}\,T))))\,\mathcal{U}\,R)$	20	2688	20	1892	86	14036
50	$\Box(Q \rightarrow (P \rightarrow (\neg R\,\mathcal{U}\,(S \wedge \neg R \wedge \neg Z \wedge X((\neg R \wedge \neg Z)\,\mathcal{U}\,T))))\,\mathcal{U}$ $(R \vee \Box(P \rightarrow (S \wedge \neg Z \wedge X((\neg Z)\,\mathcal{U}\,T)))))$	33	3700	31	3506	347	103079

Table 5.8: A comparison of the generated TGBA of Couvreur, LTL2BA, and LTL2BUCHI (part b)

No	Formulae	GPVW+		DGV+		MoDeLLa		Couvreur		LTL2BA		LTL2Buchi		LTL2Buchi+		SPIN	
		st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.
Existence (P becomes true)																	
1	$\Diamond(P)$	4	10	4	10	5	13	2	5	2	5	4	10	2	5	2	4
2	$\neg R\ \mathcal{W}(P \wedge \neg R)$	4	14	4	14	5	17	3	10	3	10	4	14	3	10	4	14
3	$\Box(\neg Q) \vee \Diamond(Q \wedge \Diamond P))$	13	65	13	65	32	157	6	34	5	28	8	42	5	28	5	28
4	$\Box(Q \wedge \neg R \rightarrow (\neg R\ \mathcal{W}(P \wedge \neg R)))$	8	103	5	62	18	158	3	28	5	47	8	103	3	28	8	86
5	$\Box(Q \wedge \neg R \rightarrow (\neg R\ \mathcal{U}(P \wedge \neg R)))$	7	88	5	62	18	158	3	28	6	53	8	103	4	38	4	36
Bounded Existence (P becomes true twice)																	
6	$(\neg P\ \mathcal{W}(P\ \mathcal{W}(\neg P\ \mathcal{W}(P\ \mathcal{W}\Box\neg P))))$	6	20	6	20	16	40	6	20	6	20	6	20	6	20	6	34
7	$\Diamond R \rightarrow ((\neg P \wedge \neg R)\ \mathcal{U}(R \vee ((P \wedge \neg R)\ \mathcal{U}$ $(R \vee ((\neg P \wedge \neg R)\ \mathcal{U}$ $(R \vee ((P \wedge \neg R)\ \mathcal{U}(R \vee (\neg P\ \mathcal{U} R))))))))$	79	425	34	158	786	3376	9	51	11	62	9	50	8	46	8	46
8	$\Diamond Q \rightarrow (\neg Q\ \mathcal{U}(Q \wedge (\neg P\ \mathcal{W}(P\ \mathcal{W}(\neg P\ \mathcal{W}(P\ \mathcal{W}\Box\neg P))))))$	13	78	13	78	49	245	8	48	8	48	13	78	8	48	9	54
9	$\Box((Q \wedge \Diamond R) \rightarrow$ $((\neg P \wedge \neg R)\ \mathcal{U}(R \vee ((P \wedge \neg R)\ \mathcal{U}$ $(R \vee ((\neg P \wedge \neg R)\ \mathcal{U}(R \vee ((P \wedge \neg R)\ \mathcal{U}$ $(R \vee (\neg P\ \mathcal{U} R)))))))))$	670	22810	41	584	12145	182950	28	820	19	244	42	1097	11	158	-	-
10	$\Box(Q \rightarrow ((\neg P \wedge \neg R)\ \mathcal{U}(R \vee ((P \wedge \neg R)\ \mathcal{U}$ $(R \vee ((\neg P \wedge \neg R)\ \mathcal{U}(R \vee ((P \wedge \neg R)\ \mathcal{U}$ $(R \vee (\neg P\ \mathcal{W} R) \vee \Box P)))))))))$	350	12685	33	536	4055	66654	35	820	18	300	-	-	-	-	-	-
Universality (P is always true)																	
11	$\Box(P)$	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1
12	$\Diamond R \rightarrow (P\ \mathcal{U} R)$	5	20	5	20	14	50	4	16	4	16	5	20	4	16	4	16
13	$\Box(Q \rightarrow \Box(P))$	4	14	3	10	7	20	3	10	3	10	4	14	3	10	3	10
14	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (P\ \mathcal{U} R))$	12	170	5	60	33	294	5	60	8	76	8	96	5	52	7	66
15	$\Box(Q \wedge \neg R \rightarrow (P\ \mathcal{W} R))$	7	90	4	44	19	174	3	28	5	48	6	72	3	28	8	86
Precedence (S precedes P)																	
16	$\neg P\ \mathcal{W} S$	4	16	4	16	6	24	3	12	3	12	4	16	3	12	4	16
17	$\Diamond R \rightarrow (\neg P\ \mathcal{U}(S \vee R))$	6	56	6	56	30	230	4	36	4	36	6	56	4	36	4	38
18	$\Box\neg Q \vee \Diamond(Q \wedge (\neg P\ \mathcal{W} S))$	8	72	8	72	42	376	5	48	5	48	8	72	5	48	6	56
19	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (\neg P\ \mathcal{U}(S \vee R)))$	17	644	6	192	79	1642	5	134	11	230	19	702	7	160	8	164
20	$\Box(Q \wedge \neg R \rightarrow (\neg P\ \mathcal{W}(S \vee R)))$	11	416	5	152	43	914	3	64	5	112	11	416	3	64	8	196
Response (S responds to P)																	
21	$\Box(P \rightarrow \Diamond S)$	6	50	4	30	14	90	3	20	6	41	6	54	4	30	4	28
22	$\Diamond R \rightarrow (P \rightarrow (\neg R\ \mathcal{U}(S \wedge \neg R)))\ \mathcal{U} R$	17	178	13	126	52	397	6	54	8	82	10	113	5	48	5	48
23	$\Box(Q \rightarrow \Box(P \rightarrow \Diamond S))$	12	292	5	84	45	602	4	62	7	104	14	448	5	86	6	100
24	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (P \rightarrow (\neg R\ \mathcal{U}(S \wedge \neg R)))\ \mathcal{U} R)$	54	2288	13	372	397	9258	13	420	10	240	20	837	6	156	9	254
25	$\Box(Q \wedge \neg R \rightarrow ((P \rightarrow (\neg R\ \mathcal{U}(S \wedge \neg R)))\ \mathcal{W} R))$	15	574	6	176	69	1542	7	178	8	186	18	781	5	122	15	398

Table 5.9: A comparison of the result BA of all six algorithms and SPIN (part a)

No	Formulae	GPVW+		DGV+		MoDeLLa		Couvreur		LTL2BA		LTL2Buchi		LTL2Buchi+		Spin	
		st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.	st.	tran.
Precedence Chain (S, T precedes P)																	
26	$\Diamond P \rightarrow (\neg P\,\mathcal{U}\,(S \land \neg P \land X(\neg P\,\mathcal{U}\,T)))$	13	94	11	78	48	306	6	44	5	36	7	52	6	44	4	66
27	$\Diamond R \rightarrow (\neg P\,\mathcal{U}\,(R \lor (S \land \neg P \land X(\neg P\,\mathcal{U}\,T))))$	15	244	13	212	108	1548	6	104	5	88	8	136	6	104	4	104
28	$(\Box\neg Q) \lor (\neg Q\,\mathcal{U}\,(Q \land \Diamond P \rightarrow$ $(\neg P\,\mathcal{U}\,(S \land \neg P \land X(\neg P\,\mathcal{U}\,T)))))$	30	468	26	444	476	6812	8	156	8	152	10	188	8	156	9	192
29	$\Box((Q \land \Diamond R) \rightarrow$ $(\neg P\,\mathcal{U}\,(R \lor (S \land \neg P \land X(\neg P\,\mathcal{U}\,T)))))$	74	4332	29	1680	949	40304	17	828	25	1102	60	4099	17	834	20	918
30	$\Box(Q \rightarrow (\Diamond P \rightarrow$ $(\neg P\,\mathcal{U}\,(R \lor (S \land \neg P \land X(\neg P\,\mathcal{U}\,T))))))$	85	6636	29	1760	1169	64528	13	780	29	1856	72	6652	17	1196	25	1690
Precedence Chain (P precedes S, T)																	
31	$(\Diamond(S \land X\Diamond T)) \rightarrow ((\neg S)\,\mathcal{U}\,P)$	8	74	7	64	46	332	5	52	5	52	7	64	5	52	6	64
32	$\Diamond R \rightarrow ((\neg(S \land (\neg R) \land$ $X(\neg R\,\mathcal{U}\,(T \land \neg R))))\,\mathcal{U}\,(R \lor P))$	27	1104	17	652	156	4224	6	170	10	334	22	844	6	170	8	260
33	$(\Box\neg Q) \lor ((\neg Q)\,\mathcal{U}\,(Q \land ((\Diamond(S \land X\Diamond T)) \rightarrow$ $((\neg S)\,\mathcal{U}\,P))))$	22	364	18	288	220	3028	7	136	8	152	13	224	7	136	8	160
34	$\Box((Q \land \Diamond R) \rightarrow$ $((\neg(S \land (\neg R) \land X(\neg R\,\mathcal{U}\,(T \land \neg R))))\,\mathcal{U}\,(R \lor P)))$	62	9384	20	2312	731	51848	13	1258	19	1294	76	15736	13	1022	17	1172
35	$\Box(Q \rightarrow (\neg(S \land (\neg R) \land X$ $(\neg R\,\mathcal{U}\,(T \land \neg R)))\,\mathcal{U}\,(R \lor P) \lor \Box(\neg(S \land X\Diamond T))))$	192	39688	28	3120	1335	122224	13	1466	28	3430	185	49210	19	2670	24	2872
Response Chain (P responds S, T)																	
36	$\Box(S \land X\Diamond T \rightarrow X(\Diamond(T \land \Diamond P)))$	57	1113	40	758	352	5792	8	143	21	392	27	512	12	252	14	250
37	$\Diamond R \rightarrow$ $(S \land X(\neg R\,\mathcal{U}\,T) \rightarrow X(\neg R\,\mathcal{U}\,(T \land \Diamond P)))\,\mathcal{U}\,R$	97	2816	157	5569	2207	69940	16	581	31	1204	72	2811	22	859	22	762
38	$\Box(Q \rightarrow \Box(S \land X\Diamond T \rightarrow X(\neg T\,\mathcal{U}\,(T \land \Diamond P))))$	115	5592	43	1522	1177	39104	12	448	19	700	66	3747	13	520	16	530
39	$\Box((Q \land \Diamond R) \rightarrow$ $(S \land X(\neg R\,\mathcal{U}\,T) \rightarrow X(\neg R\,\mathcal{U}\,(T \land \Diamond P)))\,\mathcal{U}\,R)$	435	45932	213	19798	13995	1135336	61	8102	38	3222	193	32408	25	2550	80	8614
40	$\Box(Q \rightarrow (S \land X(\neg R\,\mathcal{U}\,T) \rightarrow X(\neg R\,\mathcal{U}\,(T \land \Diamond P)))\,\mathcal{U}$ $(R \lor \Box(S \land X(\neg R\,\mathcal{U}\,T) \rightarrow X(\neg R\,\mathcal{U}\,(T \land \Diamond P)))))$	1236	304813	343	43988	13727	1115872	82	16778	57	8256	-	-	-	-	-	-
Response Chain (S, T responds P)																	
41	$\Box(P \rightarrow \Diamond(S \land X\Diamond T))$	29	626	17	308	169	2880	9	152	13	226	23	579	9	176	10	188
42	$\Diamond R \rightarrow (P \rightarrow (\neg R\,\mathcal{U}\,(S \land \neg R \land X(\neg R\,\mathcal{U}\,T))))\,\mathcal{U}\,R$	28	624	27	548	288	4940	10	202	18	388	23	611	7	146	7	134
43	$\Box(Q \rightarrow \Box(P \rightarrow (S \land X\Diamond T)))$	14	432	7	152	57	984	5	96	7	132	17	622	5	104	6	120
44	$\Box((Q \land \Diamond R) \rightarrow (P \rightarrow$ $(\neg R\,\mathcal{U}\,(S \land \neg R \land X(\neg R\,\mathcal{U}\,T))))\,\mathcal{U}\,R)$	149	13576	31	1732	3525	188992	19	1344	14	720	49	4909	8	440	43	2807
45	$\Box(Q \rightarrow (P \rightarrow (\neg R\,\mathcal{U}\,(S \land \neg R \land X(\neg R\,\mathcal{U}\,T))))\,\mathcal{U}$ $(R \lor \Box(P \rightarrow (S \land X\Diamond T))))$	-	-	126	6844	-	-	71	8204	60	4630	621	171092	-	-	-	-
Constrained Chain Patterns (S, T without Z responds to P)																	
46	$\Box(P \rightarrow \Diamond(S \land \neg Z \land X((\neg Z)\,\mathcal{U}\,T)))$	29	872	17	444	245	5908	7	172	13	320	23	771	9	240	10	258
47	$\Diamond R \rightarrow (P \rightarrow$ $(\neg R\,\mathcal{U}\,(S \land \neg R \land \neg Z \land X((\neg R \land \neg Z)\,\mathcal{U}\,T))))\,\mathcal{U}\,R$	24	860	17	568	366	9592	8	298	12	442	23	909	7	250	7	250
48	$\Box(Q \rightarrow \Box(P \rightarrow (S \land \neg Z \land X((\neg Z)\,\mathcal{U}\,T))))$	14	560	7	212	105	2632	5	144	7	192	17	768	5	152	6	172
49	$\Box((Q \land \Diamond R) \rightarrow (P \rightarrow$ $(\neg R\,\mathcal{U}\,(S \land \neg R \land \neg Z \land X((\neg R \land \neg Z)\,\mathcal{U}\,T))))\,\mathcal{U}\,R)$	149	20184	31	3004	5497	471608	46	6716	14	1216	47	6559	8	764	13	1348
50	$\Box(Q \rightarrow (P \rightarrow (\neg R\,\mathcal{U}$ $(S \land \neg R \land \neg Z \land X((\neg R \land \neg Z)\,\mathcal{U}\,T))))\,\mathcal{U}$ $(R \lor \Box(P \rightarrow (S \land \neg Z \land X((\neg Z)\,\mathcal{U}\,T)))))$	-	-	128	9506	5569	482168	75	8536	52	5190	621	190572	-	-	-	-

Table 5.10: A comparison of the result BA of all six algorithms and Spin (part b)

	$\mathcal{A}_{\neg f}$		$\mathcal{M} \cap \mathcal{A}_{\neg f}$		Memory	
Desired property	st.	tran.	st.	tran.	usage (MB)	$\mathcal{A}_{\neg f}$ generated by
1. $\square(p \rightarrow \Diamond q)$	2	7	75681	464848	5.626	Couvreur, LTL2BUCHI+, Spin
	4	14	83653	501986	5.919	GPVW+, LTL2AUT+, LTL2BA, LTL2BUCHI
	8	31	107558	648778	6.993	MoDeLLa
2. $\square(c \rightarrow ((c\,\mathcal{U}\,d)\,\mathcal{U}\,e))$	4	32	108504	829956	6.993	Couvreur, LTL2BA, LTL2BUCHI+
	5	39	135113	1040443	8.165	Spin
	10	78	143893	1096969	8.458	GPVW+, LTL2AUT+, LTL2BUCHI
	26	215	141741	1043380	8.360	MoDeLLa
3. $\square(((((a\,\mathcal{U}\,b)\,\mathcal{U}\,c)\,\mathcal{U}\,d)\,\mathcal{U}\,e)\,\mathcal{U}\,f)$	7	480	257282	3901908	13.243	Couvreur, LTL2BA, LTL2BUCHI+
	11	721	449778	6466946	21.348	Spin
	23	1469	530772	6707414	24.766	LTL2AUT+, LTL2BUCHI
	29	1822	719969	9259967	32.676	GPVW+
	418	28863	N/A	N/A	N/A	MoDeLLa

Table 5.11: The experiment result for token ring with size 6. There are 76665 states and 460929 transitions in $\mathcal{M}$.

The third experiment is focus on the performance when the generated automaton is used in model checking procedure. We implement a token ring protocol with PROMELA code, which is showed in Fig. 5.2. Three desired properties will be used to verify the protocol:

- if some node is waiting to enter the critical section, eventually it will enter the critical section.
- if the token is held by a node p_i, it will be passed to p_{i+1} and then p_{i+2} respectively (partial proceeding).
- the token is held first by p_0, then p_1, p_2, and so on (total proceeding).

We rewrite these properties into temporal formulae, and translate the negation of them with the six algorithms we had described and also SPIN. With the generated property automata, we can verify the protocol with automaton-based model checking.

The experiment result for token ring protocol with size equals to 6 is showed in Table 5.11. The result for size equals to 7 is showed in Table 5.12.

```
#define size 7
#define p (wait[_pid] == 1)
#define q (critusr == _pid)
#fefine a (token == 0)
#fefine b (token == 1)
#fefine c (token == 2)
#fefine d (token == 3)
#fefine e (token == 4)
#fefine f (token == 5)
#fefine g (token == 6)
byte token;
byte wait[size];
byte ncrit;
byte critusr;

active [size] proctype user() {

again:
if :: skip;
:: wait[_pid] = 1;
fi;
if  :: token == _pid ->
if
:: wait[_pid] == 1 ->
wait[_pid] = 0;
ncrit++;
critusr = _pid;
assert(ncrit == 1); /* critical section */
ncrit--;
critusr = 0;
assert(ncrit == 0); /* critical section */
if :: token = (token+1) % size;
fi;
token = (token+1) % size
::  token = (token+1) % size
fi;

fi;
goto again
}
```

Figure 5.2: The PROMELA code of token ring protocol

	$\mathcal{A}_{\neg f}$		$\mathcal{M} \cap \mathcal{A}_{\neg f}$		Memory	
Desired property	st.	tran.	st.	tran.	usage (MB)	$\mathcal{A}_{\neg f}$ generated by
1. $\square(p \rightarrow \diamond q)$	2	7	344700	2472567	18.223	Couvreur, LTL2BUCHI+, SPIN
	4	14	396667	2790912	20.567	GPVW+, LTL2AUT+, LTL2BA, LTL2BUCHI
	8	31	453728	3174346	23.204	MoDeLLa
2. $\square(c \rightarrow ((c\,\mathcal{U}\,d)\,\mathcal{U}\,e))$	4	32	449743	3409257	23.008	Couvreur, LTL2BA, LTL2BUCHI+
	5	39	545968	4742975	27.403	SPIN
	10	78	624381	5228076	31.016	GPVW+, LTL2AUT+, LTL2BUCHI
	26	215	657326	5903425	32.579	MoDeLLa
3. $\square((((((a\,\mathcal{U}\,b)\,\mathcal{U}\,c)\,\mathcal{U}\,d)\,\mathcal{U}\,e)\,\mathcal{U}\,f)\,\mathcal{U}\,g)$	8	1088	1291721	25622379	61.583	Couvreur, LTL2BA, LTL2BUCHI+
	13	1697	2331436	43684255	116.751	SPIN
	30	3773	3174908	49864012	155.325	LTL2AUT+, LTL2BUCHI
	37	4606	4214817	67849050	202.981	GPVW+
	962	131455	N/A	N/A	N/A	MoDeLLa

Table 5.12: The experiment result for token ring with size 7. There are 330476 states and 2278785 transitions in $\mathcal{M}$.

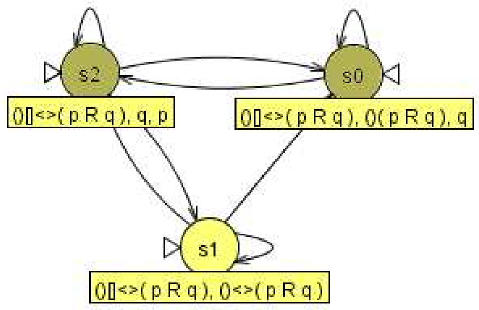
Chapter 6

Discussion and Implication

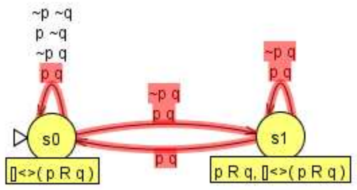
In most of the cases, algorithms with TGBA as intermediate automaton perform better than others. One of the advantages comes from the transition-based acceptance condition. A state of GBA will be labeled a set of formulae/propositions which hold on this state and also a set of formulae which should hold from the next state on. Two states labeled with same next-part formulae but differ from propositions is true at present will be considered as different states, which cannot be merged. Furthermore, it may be the case that the accepting set contains one of them instead both of them. Remind that an accepting set of TGBA is a set of transitions. A state of a TGBA will be labeled a set of formulae which holds on this state only. One state can have two different transitions consuming different alphabet and are contained in different accepting sets but reach the same successor state. This situation always happens when the formula of the state is a $\Box\Diamond$-formula. States then can be merged without considering the accepting set. For example, for a formula $\Box\Diamond(p\ \mathcal{R}\, q)$, the result GBA from LTL2AUT and TGBA from Couvreur's algorithm is illustrated in Fig. 6.1. The label of the GBA are the proposition have to be true on the current state and the formulae set which should be true from the next state on denoted as "`()f`." The label of the TGBA are the next-part of the current state denoted as "`f`." s_1 and s_2 in GBA can be simulated by the state s_0 in TGBA even though s_2 is in the accepting set while s_1 is not.

Another advantage is we can use on-the-fly simulation during each stage of the translation. For example, LTL2BA simplifies the automata in each step using the rules as follows:

- If a transition t_1 implies a transition t_2, then t_2 can be removed,



(a) GBA generated by LTL2AUT



(b) TGBA generated by Couvreur's algorithm

Figure 6.1: The comparison of LTL2AUT and Couvreur's algorithm

- If two sates s_1 and s_2 are equivalent, then they can be merged.

The simplification rules help the algorithm generate smaller automaton in each step.

In the comparison of these algorithms with TGBA as intermediate automaton, we also notice some interesting phenomena. In most cases, Couvreur's algorithm gives the smallest automaton than the other two. Moreover, Couvreur's algorithm performs better than others dealing with recurrence formulae in Manna and Pnueli's hierarchy. It is not surprising because they optimize the expansion function for $\Box\Diamond$-formulae. However, Couvreur's algorithm encounters some dilemmas. A big difference between Couvreur's algorithm and LTL2BUCHI+ is that Couvreur's algorithm shrink the result automaton by merging the states with the same next-part and the literals which should hold on current state will be record on the in-transitions. There might be the case that two transitions with different alphabet point to the same state in the result automaton of Couvreur's algorithm. The accepting condition is also be handled this way. Hence, a accepting set corresponding to an $\mathcal{U}$-formula will also be minimized. But in LTL2BUCHI, states stores the information of both the literals and the next-part of the current state. In-transition of a state in the result automaton will always label with same alphabet, so does the acceptance condition. Couvreur's algorithm thus, without doubt, generate a smaller TGBA than LTL2BUCHI. Yet in the transforming step of TGBA to BA, the construction simplify the result BA on-the-fly. The information of satisfaction of an $\mathcal{U}$-formula in the result of Couvreur's algorithm is not as complete as LTL2BUCHI. A trace may need to generate more states to reach the lasso of the accepting run in the result TGBA generated by Couvreur's algorithm. Moreover, LTL2BA can not beat LTL2BUCHI for this kinds of formulae with similar reason. The first stage and second stage of LTL2BA reduce the state size and the accepting sets, which cause the lost of information. Unfortunately, the relation between this situation and the input formula is not vivid.

From these comparison results, we notice that Spin may not be able to generate the smallest automaton for each formula in most of the cases. If one uses the smallest automaton generated by other algorithms rather than Spin, time and space can be saved. We also noticed that MoDeLLa performs better that GPVW in a particular case when the size of the token ring is 6. Thus, it is possible to produce smaller product automaton if the property automaton is "more deterministic" than the others. However, it doesn't help in most of the cases.

6.1 Portfolio

From the experimental results in Chapter 5, none of the algorithms is dominant. Hence, we propose a portfolio for choosing better-performance algorithm when combining with model checking procedure.

Some model checking procedures use GBA/TGBA as the property automata instead of BA. For these procedures, LTL2AUT+ and Couvreur's algorithm gives the smallest GBA/TGBA in most of the time. For smallest BA, the situation is much trickier. If one needs the smallest automaton, always choose an algorithm with TGBA as intermediate automaton. If the formula can be classified in recurrence of Manna and Pnueli's hierarchy, Couvreur's algorithm can always give the smallest automaton. Otherwise, We cannot tell which of the three will generate the smallest automaton. However, by our experimental experiences, LTL2BUCHI always takes longer time and spend more space than the rest of the two. Hence, for one who has time and space considerations, use Couvreur and LTL2BA might be a better choice. On the other hand, for one who can always pre-produce the automaton for the property with large-memory environment can use LTL2BUCHI+ more often since it might beat both of the two on some conditions and often make tight on the other cases.

A screen shot of Büchi Store is illustrated in Fig. 6.2.

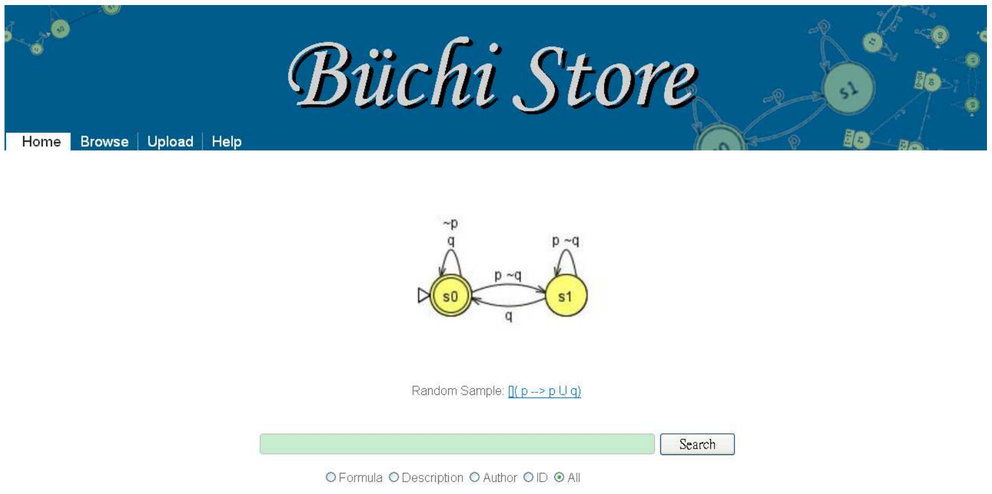


Figure 6.2: The screen shot of Büchi Store

6.2 Büchi Store

From section 6.1, we conclude that there is no algorithm which can always generate the smallest BA for different kinds of formulae. When people need the smallest BA for a formula, the only thing they can do is to translate the formula by each algorithms and pick up the best one. This solution is not that practical because it wastes a lot of time and space generating some automata which are useless.

The conclusion inspires us a new idea for "pre-translating" the temporal formula to BA, which is "*table look-up.*" The number of the formulae which are frequently used is limited. If we can have a repository for each of them to the corresponding smallest automaton, people can reach the corresponding BA for their need very quickly without applying translation algorithms.

Thus, we build up a web-based open repository "Büchi Store." It stores numerous of temporal formulae and their corresponding Büchi automata. People can contribute the best BA in their understanding and get the smallest BA for each formula collected in Büchi Store. With people's contributions, Büchi Store will become more and more helpful for research, industry, and education. Büchi Store not only collects Büchi automata, but also classifies the temporal formulae in different manners. People can get the BA in different classes depends on which property they are trying to verify to the target system.

Chapter 7

Conclusion

In this thesis, we attempted to find out which translation algorithms generates a smaller automaton from the given specification formula as the size of the specification automaton determines the efficiency of model checking. We gave a comprehensive review of a substantial number of translation algorithms. We compared six of these algorithms as well as the one implemented in Spin. Algorithms that translate the formula with intermediate TGBA tend to generate smaller automaton. However, none of the algorithms can always generate the smallest automata for various of temporal formulae. From the experimental results, we also fund that the idea of making the result automaton "more deterministic" does not seem to help. When the size of the specification automaton becomes much bigger than others, the automaton may not be able to improve the performance of model checking. We proposed a portfolio for choosing these translation algorithms to generate specification automata. For an even more convenient approach, we built an open Büchi automata repository, the Büchi Store where one can look up the Büchi automaton for a given temporal formula.

7.1 Contributions

Our contributions can be summarized as follows:

- *Comparison of translation algorithms*

  We compared six of the translation algorithms and designed three experiments for the comparison. From these comparison results, we noticed that some of the algorithms may perform better than others most of the time. Via the comparison of the translation algorithms, we got more understanding about the algorithms. We also proposed the portfolio for translation algorithms with the help of comparison results.

- *Expansion of translation algorithms in GOAL*
  Originally, GOAL had nine translation algorithms. The collection has been extended to twelve now. Couvreur's algorithm, LTL2BUCHI, and MoDeLLa had been implemented. LTL2BA had been remodeled for presenting each stage of the translation. Couvreur's algorithm and LTL2BUCHI are also improved. We apply the third stage of LTL2BA to Couvreur's algorithm in order to translate the result TGBA of Couvreur's algorithm into a Büchi Automaton. LTL2BUCHI is also improved by applying the same algorithms for better performance.

- *Extension of the user interface in GOAL*
  In the past, GOAL has the ability to present state-based regular automata. Since we wanted to present each step in Couvreur's algorithm, LTL2BA, and LTL2BUCHI, the user interface of transition-based automata and alternating automaton are developed. One can create and edit an automaton with transition-based acceptance conditions and alternating automaton with different acceptance conditions. One can also generate the TGBA or co-Büchi VWAA in GOAL via the translation algorithms .

- *Building an open repository – the Büchi Store*
  With better understanding of the translation algorithms, we noticed the fact that none of the algorithm is dominant. Sometimes we have to translate the input formula with each algorithms in order to choose the best result, yet it is impractical. Hence, we built an open repository for BA, called Büchi Store. The user can contribute the best BA in his/her understanding and get the smallest BA for each formula collected in Büchi Store. With users' contributions, the Büchi Store will become more and more helpful for research, practice, and education.

7.2 Future Work

The following works are very interesting and worth to be focused on.

- *Extending the portfolio for different classifications of formulae*

  One of the extension work is to develop a sequence of steps of procedure to classify the formulae into the classes in [17] and the Spec Patterns. With more understanding of the specification formula, the portfolio will be more complete.

- *Improving the translation algorithms*

  All of the algorithms we described in this thesis put their strength on the temporal formula with only future operators. The algorithms will be more powerful if it is improved for past operators. The structural simulation and fairness set pruning technique for TGBA haven't been mentioned before. If the translation algorithms can be improved in these directions, a smaller automaton can be generated.

- *Extending the Büchi Store*

  Currently, the functionality of Büchi Store is not completed yet. The classification algorithm is needed for classifying the temporal formulae. Some user interface abilities, like on-line automaton redraw, on-line automaton translation and verifying are not capable in the current version. The correctness check for large automaton is limited by the computing power. These work are absolutely useful for the next generation of Büchi Store.

Bibliography

[1] H. Alavi, G. Avrunin, J. Corbert, L. Dillon, M. Dwyer, and C. Pasareanu. Spec Patterns. http://patterns.projects.cis.ksu.edu/.

[2] J.R. Büchi. On a decision method in restricted second-order arithmetic. In *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Standford University Press, 1962.

[3] W.-C. Chan. A comparative study of algorithms for linear temporal logic to Büchi automata translation, 2007.

[4] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and System Science*, pages 8:117–141, 1974.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[6] J.-M. Couvreur. On-the-fly verification of linear temporal logic. In *World Congress on Formal Methods, LNCS 1708*, pages 253–271. Springer, 1999.

[7] M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In *Proceedings of the 11th International Conference on Computer-Aided Verification (CAV 1999), LNCS 1633*, pages 249–260, 1999.

[8] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translations. In *Proceedings of the 13th International Conference on Computer-Aided Verification (CAV 2001), LNCS 2102*, pages 53–65. Springer, 2001.

[9] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification*, pages 3–18. Chapman & Hall, 1995.

[10] D. Giannakopoulou and F. Lerda. From states to transitions: Improving translation of LTL formulae to büchi automata. In *Formal Techniques for Networked and Distributed Sytems, LNCS 2529*, pages 308–326. Springer-Verlag, 2002.

[11] G. J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[12] G.J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.

[13] Y. Kesten, Z. Manna, H. McGuire, and A. Pnueli. A decision algorithm for full propositional temporal logic. In *Proceedings of the 5th International Conference on Computer-Aided Verification (CAV 1993), LNCS 697*, pages 97–109. Springer, 1993.

[14] Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Information and Computation*, 163(1):203–243, 2000.

[15] O. Kupferman and M. Y. Vardi. Freedom, weakness, and determinism: From linear-time to branching-time. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS 1998)*, pages 81–92. Springer, 1998.

[16] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[17] Z. Manna and A. Pnueli. A hierarchy of temporal properties. Technical Report STAN-CS-87-1186, Stanford University, Department of Computer Science, 1987.

[18] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

[19] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[20] D. E. Muller. Infinite sequences and finite machines. In *Proceedings of the 4th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1963)*, pages 3–16, 1963.

[21] A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[22] R. Sebastiani and S. Tonetta. "more deterministic" vs. "smaller" Büchi automata for efficient LTL model checking. In *Correct Hardware Design and Verification Methods*, pages 126–140, 2003.

[23] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Proceedings of the 12th International Conference on Computer-Aided Verification (CAV 2000), LNCS 1855*, pages 248–263. Springer, 2000.

[24] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, W.-C. Chan, and C.-J. Luo. GOAL extended: Towards a research tool for omega automata and temporal logic. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008), LNCS 4963*, pages 346–350. Springer, 2008.

[25] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, K.-N. Wu, W.-C. Chan C.-J. Luo, and J.-S. Chang. Tool support for learning büchi automata and linear temporal logic. *Formal Aspects of Computing*, 21(3):259–275, 2009.