

國立臺灣大學管理學院資訊管理學研究所

碩士論文

Department of Information Management

College of Management

National Taiwan University

Master Thesis

時序性質之轉換、分類及其應用

Transformation and Classification of Temporal Properties

with Applications



Yi-Wen Chang

指導教授：蔡益坤 博士

Advisor: Yih-Kuen Tsay, Ph.D.

中華民國 99 年 7 月

July, 2010



國立台灣大學資訊管理學研究所碩士論文

指導教授：蔡益坤 博士

時序性質之轉換、分類及其應用

**Transformation and Classification of Temporal Properties
with Applications**



研究生：常怡文 撰

中華民國九十九年七月

時序性質之轉換、分類及其應用

**Transformation and Classification of Temporal Properties
with Applications**



本論文係提交國立臺灣大學
資訊管理學研究所作為完成碩士學位
所需條件之一部分

研究生：常怡文 撰

中華民國九十九年七月

國立臺灣大學碩士學位論文
口試委員會審定書

時序性質之轉換、分類及其應用

本論文係常怡文君(學號 R97725004)在國立臺灣大學
資訊管理學系、所完成之碩士學位論文，於民國 99 年 6 月
24 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

蔡益坤

江介宏

郭副好

所長：

曹明禮 (公)

致謝辭

在資管所兩年，長了年紀，長了智慧，更增加了許多生活經驗。

首先要感謝我的指導老師，蔡益坤老師。老師除了指導我研究領域上的專業知識，更為我們示範了做研究應有的態度，凡事實事求是、精益求精。同時老師也會和我們分享生活經驗，每次的實驗室聚餐都是最輕鬆聊天的日子。

我還要感謝我的家人，雖然無法完全了解我的研究內容，但是提供我舒適的家，並且包容我時常晚歸。

接著，我要感謝我的男友，張晉碩。晉碩在我研究遇到困難時總是耐心與我討論，在我最無助時陪在我身旁、不斷鼓勵我，伴著我讓這篇論文從無到有。閒暇時，還安排了出遊計畫，讓有些枯燥的研究生生活多了些樂趣。

謝謝郁方、明憲兩位厲害的博士班學長，讀 paper 遇到不懂的問題總是可以請教他們，得到一些有用的提示或想法；而明憲的 coding 習慣對我來說也是一個良好的示範。

睿元、昇峰、智斌是和我在同一間實驗室的同學們，大家一起學習、一起和論文奮鬥。另外也謝謝睿元處理了許多實驗室的大小事。依珊、啟傑、正一和任峰、奕翔、辰旻這些實驗室的學長姐、學弟妹們，以及研究所的各位同學們，也伴隨我度過了這兩年的研究生生活。祝福大家未來不論在學業或者事業上都能夠發揮所長，在各自的人生下一階段都能有很好的發展。



常怡文 謹識

予臺灣大學資訊管理研究所 民國九十九年七月

論文摘要

作者：常怡文

九十九年七月

指導教授：蔡益坤 博士

時序性質之轉換、分類及其應用

以自動機為基礎的自動化模型驗證問題為：給定一個系統 M 以及一個時序邏輯特性 f ，判斷 A_M 和 $A_{\neg f}$ 的交集所能接受的語言是否為空集合；其中 A_M 為表示系統 M 的自動機， $A_{\neg f}$ 為表示性質 f 的否定的自動機。原則上，較小的自動機 $A_{\neg f}$ 可以加速模型驗證的過程。最近，一個叫做叫做「Büchi Store」的開放 Büchi 自動機知識倉儲庫被提出，其中收集了數百個時序邏輯和其對應的自動機。使用者可以藉由查表的方式得到特定的特性所對應的自動機，而不是使用轉換演算法。由於 Büchi Store 中將收集數百甚至數千個時序邏輯，因此需要一個適當的分類方法，以方便使用者搜尋。

在本篇論文中，我們研讀了時序性質轉換與分類的方法，其中性質可以邏輯式或自動機表示。藉由了解不同類別的時序性質，我們可以為程式做更精確的規範。我們在 GOAL 這項工具上實做了 Manna 和 Pnueli 所提出的分類演算法，並將其應用於 Büchi Store。這將讓使用者在這些經過分類的邏輯式中能夠更容易地搜尋。此外，檢查兩個邏輯式是否等價，或者尋找和給定邏輯式等價的式子都將變得更容易，因為兩個邏輯式只有在屬於同一類別時才會等價。因此，GOAL 的研究和教育能力將提升，Büchi Store 的功能也將增加。

關鍵字：Büchi 自動機、Büchi Store、分類、GOAL、Omega 自動機、時序邏輯、轉換、程式驗證

THESIS ABSTRACT

Graduate Institute of Information Management
National Taiwan University

Student: Chang, Yi-Wen
Adviser: Tsay, Yih-Kuen

Month/Year: July, 2010

Transformation and Classification of Temporal Properties with Applications

In the automata-based approach, the model checking problem can be stated as follows: given a system M and a temporal property f , determine whether $L(A_M \cap A_{\neg f}) = \emptyset$, where A_M is a Büchi automaton representing the system M and $A_{\neg f}$ is an automaton representing the negation of the given property f . In principles, a smaller $A_{\neg f}$ would be speed up the model checking process. An open repository called Büchi Store has been proposed recently, where numerous temporal formulae and their corresponding automata are collected. One can obtain the Büchi automaton of a desired formula by table look-up rather than applying translation algorithms. Since there will be hundreds or even thousands of formulae in the Büchi Store, an appropriate formulae classification is needed for the user to browse and search readily.

In this thesis, we study property transformation and classification methods, where properties are represented as formulae or automata. With the understanding of different classes of temporal properties, one can specify a program more completely and avoid underspecification. We implement the classification algorithm proposed by Manna and Pnueli in GOAL, which is a tool for creating, manipulating, and testing temporal formulae and ω -automata, and apply the classification methods on formulae in the Büchi Store. These will make it easier for the user to search in the classified formulae. Moreover, checking the equivalence between two formulae or finding an equivalent formula for a given formula becomes easier, as two formulae are equivalent only if they belong to the same class. As a result, the capability of research and education will be enhanced in GOAL and the functionality of the Büchi Store will also be enriched.

Keywords: Büchi Automata, Büchi Store, Classification, GOAL, ω -Automata, Temporal Logic, Transformation, Verification.

Contents

1	Introduction	1
1.1	Background	1
1.2	Motivation and Objectives	2
1.3	Thesis Outline	3
2	Related Work	5
2.1	Formula Classification	5
2.2	Formula Rewriting	6
2.3	Tools	7
2.3.1	LTL2BA	7
2.3.2	GOAL	7
2.3.3	Büchi Store	10
3	Preliminaries	12
3.1	Automata on Infinite Words	12
3.2	ω -automata	13
3.2.1	Büchi Automata	13
3.2.2	Generalized Büchi Automata	15
3.2.3	Muller automata	16
3.2.4	Rabin automata	17
3.2.5	Streett automata	18
3.2.6	Parity automata	19
3.3	Propositional Linear Temporal Logic (PTL)	20
4	Transformation	24
4.1	Formula Rewriting	24
4.2	Formula Translation	29
4.3	Automata Transformation	31
4.3.1	Safra Tree	32
4.3.2	Nondeterministic Büchi to Deterministic Muller and Deterministic Rabin	34
5	Classification	37
5.1	Temporal Hierarchy	37
5.1.1	The Temporal Logic View	37
5.1.2	The Automata View	39
5.2	Deterministic Büchi v.s. Nondeterministic Büchi	44

6	Implementation and Applications	51
6.1	Implementations in GOAL	51
6.2	Applications on the Büchi Store	52
7	Conclusion	57
7.1	Contributions	57
7.2	Future Work	58
	Bibliography	60



List of Figures

2.1	The result BA from the web application LTL2BA	8
2.2	The editing environment of GOAL	9
2.3	A screen shot of the Büchi Store	11
4.1	Examples GBAs for rewrite formula rules proposed in [22]	25
4.2	Examples GBAs for rewrite formula rules proposed in [6]	28
4.3	The create graph algorithm.	32
4.4	The expand function.	33
4.5	An example of translating a nondeterministic Büchi automaton into an equivalent deterministic Muller automaton and an equivalent deterministic Rabin automaton	36
5.1	Inclusion relations between the classes	39
5.2	An example of safety automata	40
5.3	An example of guarantee automata	40
5.4	An example of obligation automata	41
5.5	An example of recurrence automata	42
5.6	An examples of persistence automata	42
5.7	An examples reactivity of automata	42
5.8	An examples of reactivity automata	44
5.9	Examples of translating DMW into DBW	48
5.10	Another examples of translating DMW into DBW	49
5.11	Another examples of translating DMW into DBW	50
6.1	Test of temporal hierarchy classification for an input formula in GOAL	52
6.2	Convert a deterministic Muller automaton to a deterministic Büchi automaton by the power of GOAL	53
6.3	Browse automata sorted by temporal hierarchy in the Büchi Store	56

List of Tables

4.1	Rewriting rules proposed in [22]	26
4.2	Rules to check for $\varphi \leq \psi$ in [22]	26
4.3	Definitions of New and Next functions for non-literals	30
6.1	The experiment result of deterministic Muller to deterministic Büchi converting algorithm for 15 random cases.	54



Chapter 1

Introduction

Program verification is a fundamental issue about the correctness of programs. A program should always accomplish the goal the programmer proposed and should not cause any unexpected side-effect. One common solution is to test the program with different use-cases. Yet it might cost lots of time and human work and may not be able to cover all the possible user behaviors. Hence some systematical methods are proposed to guarantee the correctness of a program.

1.1 Background

In the early years, several methods are used to seize this goal, such as testing and simulating. In 1981, “model checking” was introduced by E. M. Clarke and E. A. Emerson, which is an automatic process to check whether a system satisfies a given property [4].

The fundamental problem of model checking is to solve, given a system M and a specification property f , whether $M \models f$. It involves three main phases, which are modeling, specification, and verification. In modeling, a system M is usually given as a model of the target program. The system can also be formalized by a finite state machine, which can be represented by Kripke structure. Kripke structure contains nodes to represent each state in the target program and arcs to represent the variation between states for each statements in the program. Because a Kripke structure can be transformed into an equivalent ω -automaton[5], we will use ω -automaton to describe M in the rest of the thesis. In specification, temporal logic is used to describe the specification property. Temporal logic, which is a logic language, is widely used to describe the desired property with temporal operators in terms of time. A temporal formula is a combination of several

temporal operators to describe a property. In this step, we translate the temporal formula property f into an equivalence automaton A_f . Usually, Büchi automaton is chosen to represent A_f . Büchi automaton is first represented by J. R. Büchi in the 1960's, which is the first work related to ω -automata [1]. It is also proved in [1, 19] that each temporal logic formula can be translated into an equivalent Büchi automaton. Actually, lesser the temporal operators are used in f , smaller A_f will be. Rewriting the given formula to obtain a equivalent formula with lesser operators before translating it into automaton is very essential method for model checking performance. In verification, we solve the model checking problem whether $M \models f$ by solving this equivalence containment problem whether whether $L(A_M) \subseteq L(A_f)$. The containment problem can be rewritten as an emptiness check problem, whether $L(A_M) \cap L(A_{\neg f}) = \emptyset$, which is also equivalent to $L(A_M \cap A_{\neg f}) = \emptyset$. To check $L(A_M \cap A_{\neg f}) = \emptyset$, we have to construct the intersection automaton A of A_M and $A_{\neg f}$. For a given system, the size of $A_{\neg f}$ determines the size of A . Therefore, the smaller $A_{\neg f}$ is, the faster the model checking task may be carried out.

The specification automaton $A_{\neg f}$ is often generated by applying temporal formulae to Büchi automata translation algorithms. However, none of the translation algorithms could always generate the smallest Büchi automaton for a given temporal formula in terms of state size. One way to obtain the smallest automaton is to translate the given formula into Büchi automaton and choose the smallest one, which spends lots of space and time. In order to avoid this situation, we built up a Web-based open repository, which is called Büchi Store, to store the Büchi automata corresponding to a collection of frequently-used temporal formulae. In Büchi Store, one can search for the Büchi automaton of a property formula. Temporal formulae are classified into different classes for the user to find the formula easily.

1.2 Motivation and Objectives

In order to provide better user environment, the methods of formulae classification in Büchi Store must be enriched. There are hundreds of formulae stored in Büchi Store now. For the user to browse the temporal formulae quickly, a systematical ordering is essential. We had classified the formulae intuitively by the length of the formula and

the state size of the corresponding Büchi automaton. It would be useful to classify the formulae based on the semantic meanings. The classification method should not only base on the semantic meanings of the formulae but also be easy for user to understand. Moreover, the classification method should be systematical and be able to classify the formulae automatically.

In addition, compare to nondeterministic Büchi automata, complementation and empty check of deterministic Büchi automata is easier. Hence, classification of a property as deterministic or nondeterministic Büchi would be useful.

For these goals, we will develop the classification and transformation algorithms based on [17] and [15]. We will also implement the algorithms in GOAL, which is a graphical interactive tool for user to create, manipulate, and test temporal formulae and ω -automata. Last but not the least, we will apply the classification algorithm on the formulae in Büchi Store.

1.3 Thesis Outline

The rest of this thesis is organized as follows:

- In Chapter 2, we will introduce the approaches of properties classification and some formula rewriting approaches. We will also describe some tools which are related to temporal logic classification.
- In Chapter 3, we will introduce several kinds of automata and temporal logic which will be used in this thesis and the relevant research.
- In Chapter 4, we will present some formulae rewrite rules and some translation algorithms, including from formulae to automata and from automata to automata.
- In Chapter 5, we will present more details about properties classification and some classification algorithms.
- In Chapter 6, we will talk about implementations and applications of the algorithms described in Chapter 4 and 5.

- In Chapter 7, we briefly summarize this thesis and conclude out contributions. We would also describe some work we should do in the future.



Chapter 2

Related Work

2.1 Formula Classification

We say that a program P has the property Π if all the computations of P belong to Π . One can use properties to specify a program, but it may lead to *underspecification* in some cases. To avoid underspecification, one possible solution is to classify different types of properties, and provide a list of properties to the specifier to consider. L. Lamport suggested that properties of a reactive program can be partitioned off into two classes, *safety* and *liveness* properties [14]. In 1990, Z. Manna and A. Pnueli proposed a hierarchy of temporal properties. They classified temporal properties into six classes [17]. Note that a formula contains no future operators is called a *past* formula.

- A *safety formula* is a formula of the form

$$\Box p,$$

for a past formula p .

- A *guarantee formula* is a formula of the form

$$\Diamond p,$$

for some past formula p .

- A *simple obligation formula* is a formula of the form

$$\Box p \vee \Diamond q,$$

where p and q are past formulae.

- A *general obligation formula* is a formula of the form

$$\bigwedge_i [\Box p_i \vee \Diamond q_i],$$

where p_i and q_i are past formulae.

- A *recurrence formula* is a formula of the form

$$\Box \Diamond p,$$

for some past formula p .

- A *persistence formula* is a formula of the form

$$\Diamond \Box p,$$

for some past formula p .

- A *simple reactivity formula* is a formula of the form

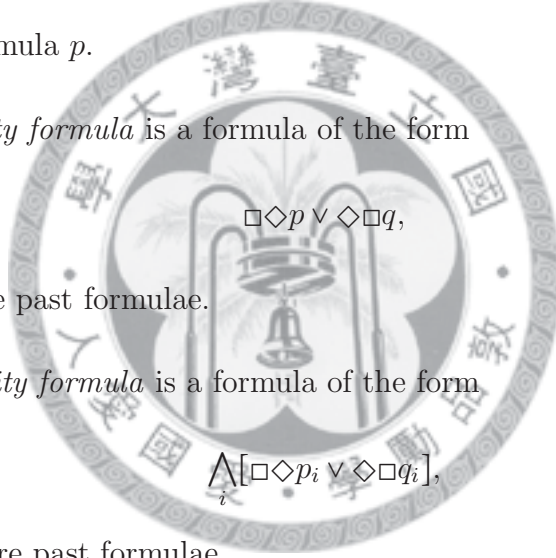
$$\Box \Diamond p \vee \Diamond \Box q,$$

where p and q are past formulae.

- A *general reactivity formula* is a formula of the form

$$\bigwedge_i [\Box \Diamond p_i \vee \Diamond \Box q_i],$$

where p_i and q_i are past formulae.



The classes of safety and guarantee properties are disjoint, so do recurrence and persistence properties. Every temporal formula is equivalent to a reactivity formula.

Another way to specify temporal properties is to use finite-state predicate automaton.

2.2 Formula Rewriting

An LTL to BA translation algorithm is introduced by F. Somenzi and R. Bloem, which consists of some heuristic approaches in the three stages of the translation [22]. They proposed several formulae and their congruent formulae for replacement. One can replace the sub-formulae of the input formulae by its congruent formula, which would help

reducing the state size of the result BA of some translation algorithms since the possible covers of the formulae produced by those algorithms are reduced. Moreover, they also proposed an approach to simplify the result Büchi automata using simulation.

K. Etessami and G. J. Holzmann also introduced several optimization methods for formula rewriting [6]. They proposed an idea to take the advantage of the suffix language of a formula. The formulae with the same suffix language can be replaced from one to another.

2.3 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.

2.3.1 LTL2BA

LTL2BA (<http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/index.php>) is a web application which translates the input formula into a BA based on the translation algorithm P. Gastin and D. Oddoux proposed in [7]. The algorithm translates an LTL formula into a Büchi automaton with three stages: (1) translating from LTL formula to very weak alternating co-Büchi automaton (VWAA), (2) translating VWAA with co-Büchi condition into TGBA, and (3) translating TGBA into BA. One can give an LTL formula and choose several translation preferences to obtain the result BA of their interest. The result BA will not only be shown in the web page by graph but also presented in PROMELA format. Figure 2.1 shows the result BA for input formula $\square\Diamond p \wedge \square\Diamond q$ translated by the web application LTL2BA.

2.3.2 GOAL

GOAL (<http://goal.im.ntu.edu.tw>) [23, 24] 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 **T**ool 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.

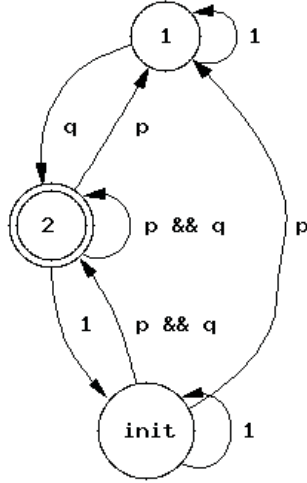


Figure 2.1: The result BA from the web application LTL2BA

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 complementation 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

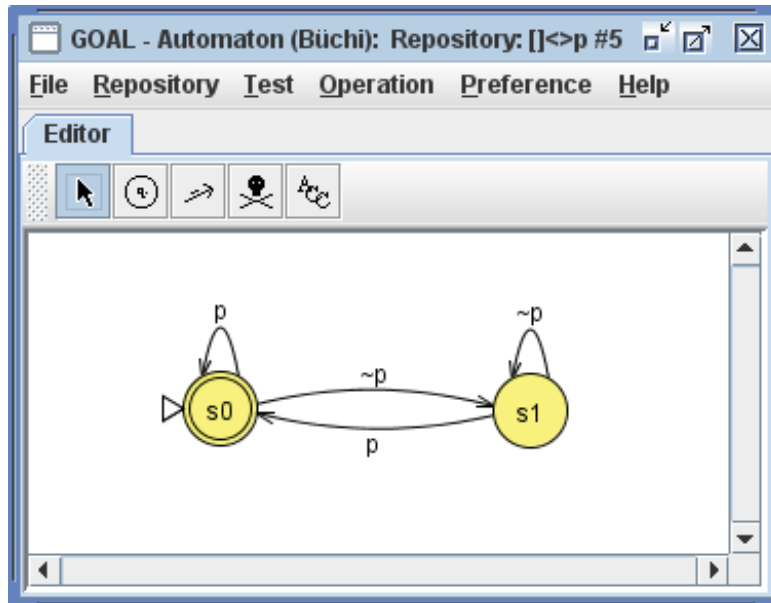


Figure 2.2: The editing environment of GOAL

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

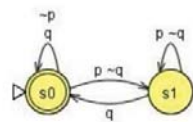
and their corresponding equivalent automata. This is a very convenient way for learning the relation between Büchi automata and QPTL for beginners.

2.3.3 Büchi Store

Büchi Store is an open Web-based repository of Büchi automata which is developed on PHP and Y.-K. Tsay is the leader of Büchi Store team at National Taiwan University [2]. Hundreds of Büchi automata are stored in Büchi Store now and categorized by temporal formulae. For each temporal formula stored in Büchi Store, which is considered as most common used specification formula, it stores the top three smallest BA. One can search for a temporal formula and obtain smaller automata by his/her choice. Users can also browse the temporal formulae by several kinds of categories which helps users reach their desired formula and the corresponding BA. Büchi Store provides two different file formats for user to download, which are GFF (GOAL file format) and Promela (a file format used in SPIN [10, 11], a common used verifier tool). Users can choose one of them depends on his/her usage.

Büchi Store also provide equivalent class and complement class for each formula. There are many algorithms to compute the complementation of a given automaton, such as in [21, 13, 12]. The state size of a complement automaton computed by a complementation algorithm is usually large. We can apply simplification algorithms or some heuristic to get a smaller automaton. However, with complement class provided by the Büchi Store, one can obtain a small complement automaton quickly.

Despite of obtaining BA for specification formula, one can upload a BA which he/she claims to be the smallest BA of a temporal formula. Büchi Store will verify whether the uploaded BA and the given temporal formula. Hence, the contents in Büchi Store could be enriched by users' contributions. Figure 2.3 illustrates the homepage of Büchi Store.



Random Sample: $\llbracket p \rightarrow p \cup q \rrbracket$

Formula Description Author ID All

Figure 2.3: A screen shot of the Büchi Store

Chapter 3

Preliminaries

In this chapter, we will briefly introduce several kinds of ω -automata and propositional linear temporal logic. These are basic knowledge of this thesis. We will describe each of them in the following.

3.1 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 [1].

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 $w = w_0w_1w_2\dots, w \in \Sigma^\omega$ and each $w_i \in \Sigma$.

An ω -automaton \mathcal{A} is a 5-tuple $(\Sigma, Q, \delta, Q_0, \mathcal{F})$ where

- Σ is the finite set of **symbols**, called alphabet,
- Q is the finite set of **states**,
- δ , is the **transition function**,
- $Q_0 \subseteq Q$ is the **initial states**, and
- \mathcal{F} is the **acceptance component**.

If $|\delta(q, \sigma)| = 1$ for $q \in Q$ and $\sigma \in \Sigma$, the automaton is *deterministic*, otherwise, it is *nondeterministic*.

A run ρ of an automaton \mathcal{A} on infinite word $w = w_0w_1w_2\dots \in \Sigma^\omega$ is a sequence of states $q_0, q_1, \dots \in Q^\omega$ where

$$q_0 \in Q_0, q_i \in Q \text{ and } q_{i+1} \in \delta(q_i, w_i) \text{ for } 0 \leq i.$$

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

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

3.2 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, Muller automata, Rabin automata and Streett automaton. We will give the definition for all of them in the following.

3.2.1 Büchi Automata

Büchi automata are often used for automata-based model checking. An ω -automaton $\mathcal{A} = (\Sigma, Q, \delta, Q_0, \mathcal{F})$ is called Büchi automaton if the acceptance condition is defined as follows: $\mathcal{F} \subseteq Q$ and a run ρ on a infinite word w is accepted by \mathcal{A} if

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

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 3.1. *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 $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$. [3]*

Proof. Let \mathcal{A}_1 and \mathcal{A}_2 be defined as follows:

$\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1, Q_{01}, \mathcal{F}_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2, Q_{02}, \mathcal{F}_2)$. Let \mathcal{A} is a 5-tuple $(\Sigma, Q, \delta, Q_0, \mathcal{F})$, where

1. $\Sigma = \Sigma_1 \cup \Sigma_2$,
2. $Q = Q_1 \cup Q_2$,
3. $Q_0 = Q_{01} \cup Q_{02}$,
4. $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$, and
5. $\delta(q, w) = \begin{cases} \delta_1(q, \sigma) & \text{if } q \in Q_1 \\ \delta_2(q, \sigma) & \text{if } q \in Q_2 \end{cases}$

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 3.2. *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 $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ [3].*

Proof. Let \mathcal{A}_1 and \mathcal{A}_2 be defined as follows:

$\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1, Q_{01}, \mathcal{F}_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2, Q_{02}, \mathcal{F}_2)$. Let \mathcal{A} is a 5-tuple $(\Sigma, Q, \delta, Q_0, \mathcal{F})$, where

1. $\Sigma = \Sigma_1 \cup \Sigma_2$,
2. $Q = Q_1 \times Q_2 \times \{1, 2\}$,
3. $Q_0 = Q_{01} \times Q_{02} \times \{1\}$,
4. $\mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2 \times \{1\}$, and
5. $\delta((q_1, q_2, i), \sigma) = (q'_1, q'_2, j)$ where $q'_1 = \delta_1(q_1, \sigma)$, $q'_2 = \delta_2(q_2, \sigma)$ 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 $\text{inf}(\rho)$ 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 3.3. *Let \mathcal{A} be a Büchi automaton. Then there exists a Büchi automaton $\overline{\mathcal{A}}$ such that $L(\overline{\mathcal{A}}) = \Sigma^\omega - L(\mathcal{A})$ [1].*

3.2.2 Generalized Büchi Automata

An ω -automaton $\mathcal{A} = (\Sigma, Q, \delta, Q_0, \mathcal{F})$ is called Generalized Büchi automata iff the acceptance condition is defined as follows: $\mathcal{F} \subseteq 2^Q$, e.g. $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_k\}$ and for all $1 \leq i \leq k$, $\mathcal{F}_i \subseteq Q$. A run ρ on an infinite word w is accepted by \mathcal{A} iff

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

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 3.4. *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 $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1, Q_{01}, \mathcal{F}_1)$, where $\mathcal{F}_1 = \{\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_k\}$. Let \mathcal{A} is a 5-tuple $(\Sigma, Q, \delta, Q_0, \mathcal{F})$, where

1. $\Sigma = \Sigma_1$,
2. $Q = Q_1 \times \{1..k\}$,
3. $Q_0 = Q_{01}$,
4. $\mathcal{F} = \mathcal{F}_1 \times \{1\}$,
5. $\delta(q_0, \sigma) = (q, 1)$ if there exists a $q_i \in Q_{01}$, $\delta_1(q_i, \sigma) = q$, and
6. $\delta((q', i), \sigma) = (q, j)$ if $\delta_1(q, \sigma) = q'$ 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} .

3.2.3 Muller automata

An ω -automaton $\mathcal{A} = (\Sigma, Q, \delta, Q_0, \mathcal{F})$ is called Muller automaton iff the acceptance condition is defined as follows: $\mathcal{F} \subseteq 2^Q$, which means the acceptance set is a set of subsets of states, e.g. $\mathcal{F} = \{F_1, F_2, \dots, F_k\}$ and for all $1 \leq i \leq k$, $F_i \subseteq Q$. A run ρ on an infinite word w is accepted by \mathcal{A} iff

$$\inf(\rho) = F_i \in \mathcal{F}.$$

In other words, there exists at least a set of states $F_i \in \mathcal{F}$ that for every state $q \in F_i$, q 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 3.5. *Let \mathcal{A}_1 be a Muller automaton. There is a Büchi automaton \mathcal{A} which accepts the same language of \mathcal{A}_1 , which means $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1)$ [16].*

Proof. Let $\mathcal{A}_1 = (\Sigma, Q_1, \delta_1, Q_{01}, \mathcal{F}_1)$ with $\text{cal}\mathcal{F}_1 = \{F_1, F_2, \dots, F_k\}$. Let \mathcal{A} is a 5-tuple $(\Sigma, Q, \delta, Q_0, \mathcal{F})$, where

1. $\Sigma = \Sigma_1$,
2. $Q = Q_1 \cup (Q_1 \times 2^{Q_1} \times \{1..k\})$,
3. $Q_0 = Q_{01}$,
4. $\mathcal{F} = \{(q, \emptyset, i) \mid q \in Q \text{ and } i \in \{1..k\}\}$, and
5. $\delta(q, \sigma) = \delta_1(q, \sigma) \cup \{(q, \emptyset, i) \mid i \in \{1..k\} \text{ and } q \in \delta(q, \sigma)\}$,

$$6. \delta((q, P, i), \sigma) = \begin{cases} \{(q', P \cup \{q\}, i) \mid q' \in \delta_1(q, \sigma)\} & \text{if } P \cup \{q\} \neq F_k \\ \{(q', \emptyset, i) \mid q' \in \delta_1(q, \sigma)\} & \text{if } P \cup \{q\} = F_k \end{cases}$$

which $\sigma \in \Sigma_1$, $q \in Q_1$, $P \subseteq Q$ and $i \in \{1..k\}$

□

Proposition 3.6. *Let \mathcal{A}_1 be a Büchi automaton. There is a Muller 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 $\mathcal{A}_1 = (\Sigma, Q, \delta, Q_0, \mathcal{F}_1)$ and $\mathcal{A} = (\Sigma, Q, \delta, Q_0, \mathcal{F})$, where

$$\mathcal{F} = \{\mathcal{G} \in 2^Q \mid \mathcal{G} \cap \mathcal{F}_1 \neq \emptyset\}$$

□

3.2.4 Rabin automata

An ω -automaton $\mathcal{A} = (\Sigma, Q, \delta, Q_0, \Omega)$ is called Rabin automaton iff the acceptance condition is defined as follows: $\Omega \subseteq 2^Q \times 2^Q$, which means the acceptance set is a pair of set of subsets of states, e.g. $\Omega = \{(E_1, F_1), (E_2, F_2), \dots, (E_k, F_k)\}$ and for all $1 \leq i \leq k$, both E_i and $F_i \subseteq Q$. A run ρ on an infinite word w is accepted by \mathcal{A} iff

$$\exists (E_i, F_i) \in \Omega, (\text{inf}(\rho) \cap E_i = \emptyset) \wedge (\text{inf}(\rho) \cap F_i \neq \emptyset).$$

In other words, there exists a pair of subsets of states (E_i, F_i) that at least one state $q \in F_i$ is visited infinitely often on run ρ , while every state in E_i is visited only finite times on ρ . A word $w \in \Sigma^\omega$ is accepted by \mathcal{A} iff there is a corresponding accepting run ρ .

Proposition 3.7. *Let \mathcal{A}_1 be a Rabin automaton. There is a Büchi automaton \mathcal{A} which accepts the same language of \mathcal{A}_1 , which means $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1)$ [16].*

Proof. Let $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1, Q_{01}, \Omega_1)$ with $\Omega_1 = \{(E_1, F_1), \dots, (E_k, F_k)\}$. Let \mathcal{A} be a 5-tuple $(\Sigma, Q, \delta, Q_0, \mathcal{F})$, where

1. $\Sigma = \Sigma_1$,
2. $Q = Q_1 \cup (Q_1 \times \{1..k\})$,

$$3. Q_0 = Q_{01},$$

$$4. \mathcal{F} = \bigcup_{i=1}^k F_i \times \{i\}, \text{ and}$$

$$5. \delta(q, \sigma) = \delta_1(q, \sigma) \cup \{(p, i) \mid i \in \{1..k\} \text{ and } p \in \delta_1(q, \sigma)\},$$

$$6. \delta((q, j), \sigma) = \begin{cases} \emptyset & \text{if } q \in E_k \\ \{(p, j) \mid p \in \delta_1(q, \sigma)\} & \text{otherwise} \end{cases}$$

which $\sigma \in \Sigma_1$, $q \in Q_1$, and $j \in \{1..k\}$.

□

3.2.5 Streett automata

An ω -automaton $\mathcal{A} = (\Sigma, Q, \delta, Q_0, \Omega)$ is called Streett automaton iff the acceptance condition is defined as follows: $\Omega \subseteq 2^Q \times 2^Q$, which means the acceptance set is a pair of set of subsets of states, e.g. $\Omega = \{(E_1, F_1), (E_2, F_2), \dots, (E_n, F_n)\}$ and for all $1 \leq i \leq n$, both E_i and $F_i \subseteq Q$. A run ρ on an infinite word w is accepted by \mathcal{A} iff

$$\forall (E_i, F_i) \in \Omega, (\text{inf}(\rho) \cap E_i \neq \emptyset) \vee (\text{inf}(\rho) \cap F_i = \emptyset).$$

In other words, for every state set pair $(E_i, F_i) \in \Omega$, either there exists at least one state $q \in E_i$ would be visited infinitely often or every state in F_i would appear only finite time on ρ would happen. A word $w \in \Sigma^\omega$ is accepted by \mathcal{A} iff there is a corresponding accepting run ρ . As you can see, Streett acceptance condition is dual to Rabin acceptance condition.

Proposition 3.8. *Let \mathcal{A}_1 be a Streett automaton. There exists a Büchi automaton \mathcal{A} which accepts the same language of \mathcal{A}_1 , which means $L(\mathcal{A}) = L(\mathcal{A}_1)$ [16].*

Proof. Let $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1, Q_{01}, \Omega_1)$ with $\Omega_1 = \{(E_1, F_1), \dots, (E_k, F_k)\}$. Let \mathcal{A} be a 5-tuple $(\Sigma, Q, \delta, Q_0, \mathcal{F})$, where

$$1. \Sigma = \Sigma_1,$$

$$2. Q = Q_1 \cup (Q_1 \times 2^{\{1..k\}} \times 2^{\{1..k\}}),$$

$$3. Q_0 = Q_{01},$$

$$4. \mathcal{F} = \mathcal{Q}_1 \times \{\emptyset\} \times \{\emptyset\},$$

$$5. \delta(q, \sigma) = \delta_1(q, \sigma) \cup \{(p, \emptyset, \emptyset) \mid p \in \delta_1(q, \sigma)\}, \text{ and}$$

$$6. \delta((q, I, J), \sigma) = \begin{cases} \{(p, I', J') \mid p \in \delta_1(q, \sigma)\} & \text{if } I' \not\subseteq J' \\ \{(p, \emptyset, \emptyset) \mid p \in \delta_1(q, \sigma)\} & \text{if } I' \subseteq J' \end{cases}$$

which $\sigma \in \Sigma_1$, and $q \in \mathcal{Q}_1$.

□

Streett acceptance can be written as a different version. Let $\Omega = \{(R_1, P_1), \dots, (R_n, P_n)\}$ and for all $1 \leq i \leq n$, both R_i and $P_i \subseteq \mathcal{Q}$. A run ρ on an infinite word w is accepted by \mathcal{A} iff

$$\forall (R_i, P_i) \in \Omega, (\text{inf}(\rho) \cap R_i \neq \emptyset) \vee (\text{inf}(\rho) \subseteq P_i).$$

3.2.6 Parity automata

An ω -automaton $\mathcal{A} = (\Sigma, Q, \delta, Q_0, \mathcal{F})$ is called Parity automaton iff the acceptance condition is defined as follows: $\mathcal{F} \subseteq 2^Q$ is a partition $\{F_0, F_1, \dots, F_n\}$ of \mathcal{S} . A run ρ on an infinite word w is accepted by \mathcal{A} iff

$$\min(\{i \mid q \in \text{inf}(\rho) \wedge q \in F_i\}) \text{ is even,}$$

where $\min(N)$ is a function which outputs the minimum integer $i \in N$. A word $w \in \Sigma^\omega$ is accepted by \mathcal{A} iff there is a corresponding accepting run ρ .

Proposition 3.9. *Let \mathcal{A}_1 be a parity automaton. There exists a Büchi automaton \mathcal{A} which accepts the same language of \mathcal{A}_1 , which means $L(\mathcal{A}) = L(\mathcal{A}_1)$.*

Proof. Let $\mathcal{A}_1 = (\Sigma_1, Q, \delta_1, q_{01}, \mathcal{F})$ with $\mathcal{F} = \{F_0, F_1, \dots, F_{2k}\}$. Let \mathcal{A} be a 5-tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, \mathcal{F})$, where

$$1. \Sigma = \Sigma_1,$$

$$2. Q = \mathcal{Q}_1 \cup (\mathcal{Q}_1 \times \{0..k\}),$$

3. $q_0 = q_{01}$,
 4. $\mathcal{F} = \bigcup_{i=0}^k F_{2i} \times \{i\}$,
 5. $\delta(q, \sigma) = \delta_1(q, \sigma) \cup \{(p, i) \mid i \in \{0..k\} \text{ and } p \in \delta(q, \sigma)\}$, and
 6. $\delta((q, j), \sigma) = \begin{cases} \emptyset & \text{if } q \in F_i, 0 \leq i < 2j \\ \{(p, j) \mid p \in \delta(q, a)\} & \text{otherwise} \end{cases}$
- which $\sigma \in \Sigma_1$, $q \in Q_1$ and $j \in \{0..k\}$.

□

3.3 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 [20].

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 (Q, Q_0, R, L) where Q is the set of states, Q_0 is the set of initial states, 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 sequence σ of \mathcal{M} from a state q is an infinite sequence of states $\sigma = q_0, q_1, \dots$ 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 of a state or a path, which would be called as state formulae or 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]. Moreover, the temporal operator can be separated into two parts, which are future operators and past operators. The semantics of PTL in terms of $(\sigma, i) \models f$, which means f holds at the i -th position is given below. A sequence of states satisfies a PTL formula f or σ is a model of f , denoted $\sigma \models f$, if $(\sigma, 0) \models f$.

State Formulae

- For a state formula p ,

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

Boolean Operators

The following are the semantics of some boolean operations.

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

$$(\sigma, i) \models \neg p \Leftrightarrow (\sigma, i) \not\models p.$$

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

$$(\sigma, i) \models p \vee q \Leftrightarrow (\sigma, i) \models p \text{ or } (\sigma, i) \models q.$$

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

$$(\sigma, i) \models p \wedge q \Leftrightarrow (\sigma, i) \models p \text{ and } (\sigma, i) \models q.$$

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 are the semantics of the future operators.

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

$$(\sigma, i) \models \circ p \Leftrightarrow (\sigma, i + 1) \models p.$$

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

$$(\sigma, i) \models \diamond p \Leftrightarrow \text{for some } k \geq i, (\sigma, k) \models p.$$

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

$$(\sigma, i) \models \square p \Leftrightarrow \text{for all } k \geq i, (\sigma, k) \models p.$$

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

$$(\sigma, i) \models p \mathcal{U} q \Leftrightarrow \text{for some } k \geq i, (\sigma, k) \models q, \text{ and for all } i \leq j < k, (\sigma, j) \models p.$$

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

$$(\sigma, i) \models p \mathcal{R} q \Leftrightarrow \text{for all } j \geq i, (\sigma, i) \not\models p \text{ for every } k, i \leq k < j, \text{ then } (\sigma, j) \models q.$$

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

$$(\sigma, i) \models p \mathcal{W} q \Leftrightarrow (\text{for some } k \geq i, (\sigma, k) \models q \text{ and for all } j, i \leq j < k, (\sigma, j) \models p) \\ \text{or } (\text{for all } j \geq i, (\sigma, j) \models p).$$

Past Operators

Here are the semantics of the past operators.

- **Previous:** $\ominus p$,

$$(\sigma, i) \models \ominus p \Leftrightarrow i > 0 \text{ and } (\sigma, i - 1) \models p$$

- **Before:** $\odot p$,

$$(\sigma, i) \models \odot p \Leftrightarrow i = 0 \text{ or } (\sigma, i - 1) \models p$$

- **Once:** $\diamond p$,

$$(\sigma, i) \models \diamond p \Leftrightarrow \text{for some } j, 0 \leq j \leq i, (\sigma, j) \models p$$

- **So-far:** $\exists p$,

$$(\sigma, i) \models \exists p \Leftrightarrow \text{for all } j, 0 \leq j \leq i, (\sigma, j) \models p$$

- **Since:** $p \mathcal{S} q$,

$$(\sigma, i) \models p \mathcal{S} q \Leftrightarrow \text{for some } k, 0 \leq k \leq i, (\sigma, k) \models q$$

and for all $j, k < j \leq i, (\sigma, j) \models p$

- **Back-to:** $p \mathcal{B} q$,

$$(\sigma, i) \models p \mathcal{B} q \Leftrightarrow (\text{for some } k, 0 \leq k \leq i, (\sigma, k) \models q \text{ and for all } j, k < j \leq i, (\sigma, j) \models p)$$

or (for all $0 \leq j \leq i, (\sigma, j) \models p$).



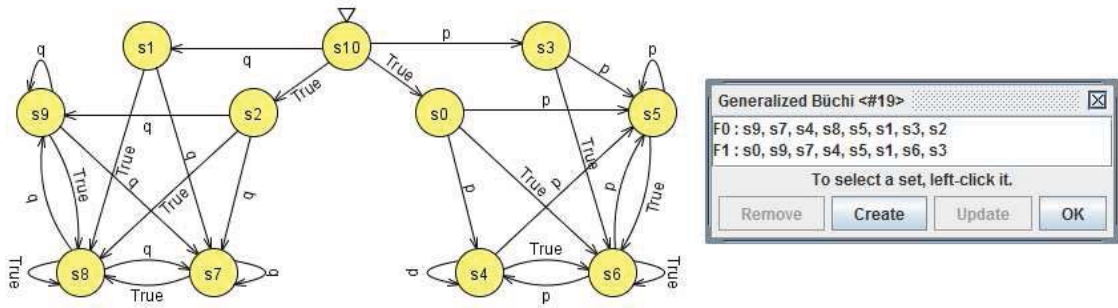
Chapter 4

Transformation

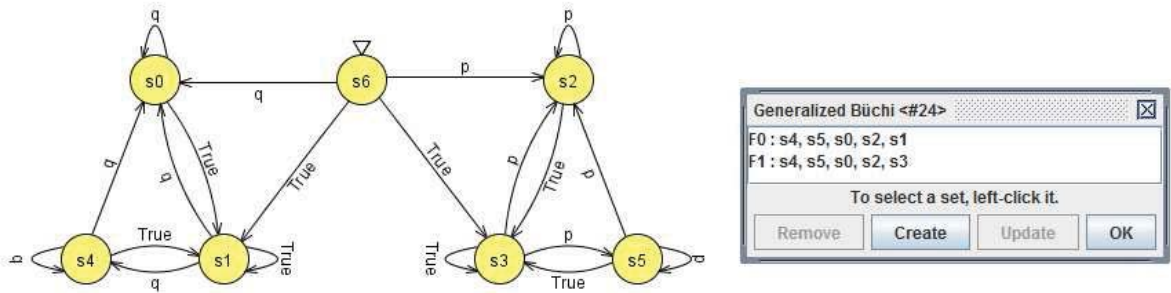
In this chapter, we will introduce some methods of formula rewriting and some algorithms that transform a formula into an equivalent automaton and transform a Büchi automaton into an equivalent deterministic Muller or Rabin automaton.

4.1 Formula Rewriting

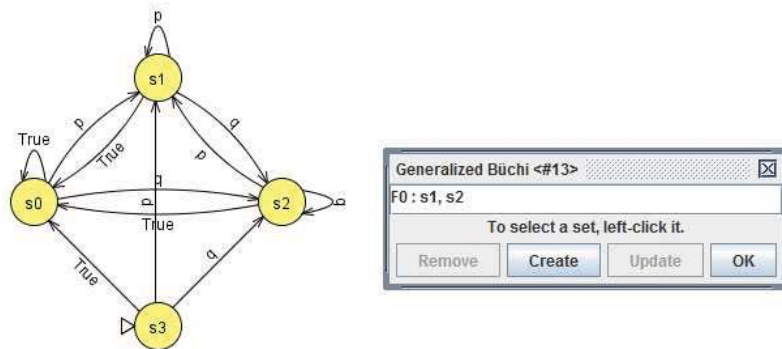
F. Somenzi and R. Bloem introduced an LTL to BA translation algorithm which consists of some heuristic approaches in the three stages of the translation, which are rewriting the input formula, optimizing translation procedure, and simplifying the result automaton [22]. They proposed several formulae and their congruent formulae for replacement, which are showed in Table 4.1, and some rules to check for $\varphi \leq \psi$ are showed in Table 4.2. Transforming the left-hand side by the right-hand side would help reducing the states size of the result automaton of translation algorithms because the possible covers of the formulae produced by those algorithms are reduced. For example, we translate $\phi = \square \diamond p \vee \square \diamond q$ with original GPVW translation algorithm, which will be introduced in Section 4.2, and the result GBA contains eleven states. Furthermore, we translate the formula ϕ with GPVW+, which is an extension of the original GPVW, and the result GBA contains seven states. On the other hand, we first transform formula ϕ into $\phi' = \square \diamond (p \vee q)$ by the rules in Table 4.2 and translate ϕ' with the same algorithm GPVW. There are only four states in the result BA, which is much smaller than the previous result. All the automata is illustrated in Fig. 4.1. Moreover, they also proposed an approach to simplify the result Büchi automata using simulation in the later part of the paper which we will not mention in this thesis.



(a) The result GBA of formula $\Box\Diamond p \vee \Box\Diamond q$ by GPVW.



(b) The result GBA of formula $\Box\Diamond p \vee \Box\Diamond q$ by GPVW+.



(c) The result GBA of formula $\Box\Diamond(p \vee q)$ by GPVW.

Figure 4.1: Examples GBAs for rewrite formula rules proposed in [22]

$\varphi \leq \psi \Rightarrow (\varphi \wedge \psi) \equiv \varphi$	$\Box \Diamond \varphi \vee \Box \Diamond \psi \equiv \Box \Diamond (\varphi \vee \psi)$
$\varphi \leq \neg \psi \Rightarrow (\varphi \wedge \psi) \equiv \mathbf{F}$	$\Diamond \Box \varphi \equiv \Box \Diamond \varphi$
$(\Box \varphi) \mathcal{U} (\Box \psi) \equiv \Box (\varphi \mathcal{U} \psi)$	$\varphi \leq \psi \Rightarrow \varphi \mathcal{U} (\psi \mathcal{U} \gamma) \equiv \psi \mathcal{U} \gamma$
$(\varphi \mathcal{R} \psi) \wedge (\varphi \mathcal{R} \gamma) \equiv \varphi \mathcal{R} (\psi \wedge \gamma)$	$\Box \Box \Diamond \varphi \equiv \Box \Diamond \varphi$
$(\varphi \mathcal{R} \gamma) \vee (\psi \mathcal{R} \gamma) \equiv (\varphi \vee \psi) \mathcal{R} \gamma$	$\Diamond \Box \Diamond \varphi \equiv \Box \Diamond \varphi$
$(\Box \varphi) \wedge (\Box \psi) \equiv \Box (\varphi \wedge \psi)$	$\Box \Box \Diamond \varphi \equiv \Box \Diamond \varphi$
$\Box \mathbf{T} \equiv \mathbf{T}$	$\Diamond (\varphi \wedge \Box \Diamond \psi) \equiv (\Diamond \varphi) \wedge (\Box \Diamond \psi)$
$\varphi \mathcal{U} \mathbf{F} \equiv \mathbf{F}$	$\Box (\varphi \vee \Box \Diamond \psi) \equiv (\Box \varphi) \vee (\Box \Diamond \psi)$
$\varphi \leq \psi \Rightarrow (\varphi \mathcal{U} \psi) \equiv \psi$	$\Box (\varphi \wedge \Box \Diamond \psi) \equiv (\Box \varphi) \wedge (\Box \Diamond \psi)$
$\neg \psi \leq \varphi \Rightarrow (\varphi \mathcal{U} \psi) \equiv (\mathbf{T} \mathcal{U} \psi)$	$\Box (\varphi \vee \Box \Diamond \psi) \equiv (\Box \varphi) \vee (\Box \Diamond \psi)$

Table 4.1: Rewriting rules proposed in [22]

$\varphi \leq \varphi$
$\varphi \leq \mathbf{T}$
$(\varphi \leq \psi) \wedge (\varphi \leq \chi) \Rightarrow \varphi \leq (\psi \wedge \chi)$
$(\varphi \leq \chi) \vee (\psi \leq \chi) \Rightarrow (\varphi \wedge \psi) \leq \chi$
$\chi \leq \psi \Rightarrow \chi \leq (\varphi \mathcal{U} \psi)$
$(\varphi \leq \chi) \wedge (\psi \leq \chi) \Rightarrow (\varphi \mathcal{U} \psi) \leq \chi$
$(\varphi \leq \chi) \wedge (\psi \leq s) \Rightarrow (\varphi \mathcal{U} \psi) \leq (\chi \mathcal{U} s)$

Table 4.2: Rules to check for $\varphi \leq \psi$ in [22]

K. Etessami and G. J. Holzmann also introduced several optimization methods for formula rewriting [6]. They proposed an idea to take the advantage of the suffix language of a formula. The formulae with the same suffix language can be replaced from one to another.

Definition 4.1. A language L of ω -words is said to be left-append closed if for all ω -words $w \in \Sigma^\omega$, and $v \in \Sigma^*$: if $w \in L$, then $vw \in L$.

Proposition 4.2. Given an formula ψ such that $L(\psi)$ is left-append closed, and any formula γ , the following equivalences hold: (1) $\gamma \mathcal{U} \psi \equiv \psi$, (2) $\Diamond \psi \equiv \psi$.

Definition 4.3. The class of pure eventuality formulae are defined as the smallest set of LTL formulae (in negation normal form) satisfying:

1. Any formula of the form $\Diamond \varphi$ is a pure eventuality formula.
2. Given pure eventuality formulae ψ_1 and ψ_2 , and γ an arbitrary formula, each of $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $\psi_1 \mathcal{U} \gamma$, $\Box \psi_1$, $\psi_1 \mathcal{R} \psi_2$ and $\Box \psi_1$, is also a pure eventuality formula.

Lemma 4.4. Every pure eventuality formula φ defines a left-append closed property $L(\varphi)$.

Definition 4.5. A language L of ω -words is *suffix closed* if whenever $w \in L$ and w' is a suffix of w , then $w' \in L$.

Proposition 4.6. For a formula ψ with a suffix closed language $L(\psi)$, and an arbitrary formula γ , the following equivalences hold: (1) $\gamma \mathcal{R} \psi \equiv \psi$, (2) $\Box \psi \equiv \psi$.

Definition 4.7. The class of *purely universal formulae* is defined inductively as the smallest set of formulae satisfying:

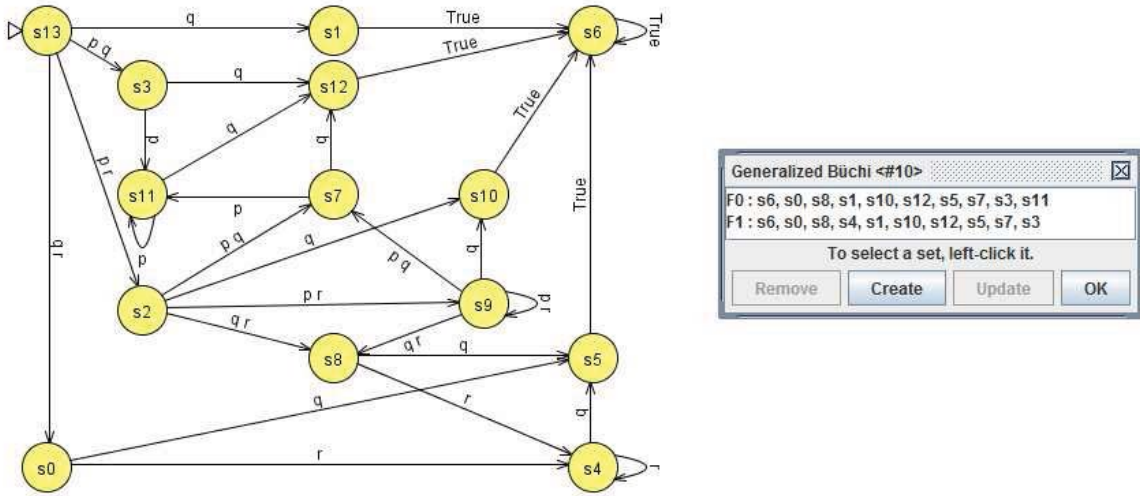
1. Any formula of the form $\Box \varphi$ is purely universal.
2. Given purely universal formulae ψ_1 and ψ_2 , and an arbitrary formula γ , any formula of the form: $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $\psi_1 \mathcal{U} \gamma$, $\Diamond \psi_1$, $\psi_1 \mathcal{R} \psi_2$ and $\bigcirc \psi_1$, is also purely universal.

Lemma 4.8. Every pure universality formula defines a suffix closed property.

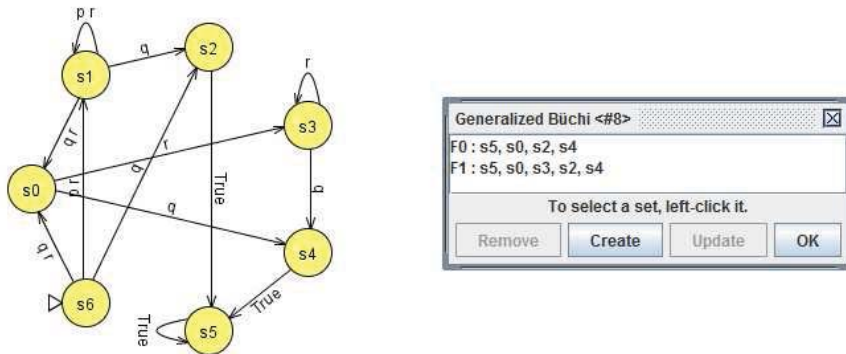
Lemma 4.9. For all LTL formulae φ , ψ , and γ , the following equivalences hold:

1. $(\varphi \mathcal{U} \psi) \wedge (\gamma \mathcal{U} \psi) \equiv (\varphi \wedge \gamma) \mathcal{U} \psi$
2. $(\varphi \mathcal{U} \psi) \vee (\varphi \mathcal{U} \gamma) \equiv \varphi \mathcal{U} (\psi \vee \gamma)$
3. $\Diamond(\varphi \mathcal{U} \psi) \equiv \Diamond \phi$
4. Whenever ψ is a pure eventuality formula $(\varphi \mathcal{U} \psi) \equiv \psi$, and $\Diamond \varphi \equiv \psi$.
5. Whenever ψ is a pure universality formula $(\varphi \mathcal{R} \psi) \equiv \psi$, and $\Box \varphi \equiv \psi$.

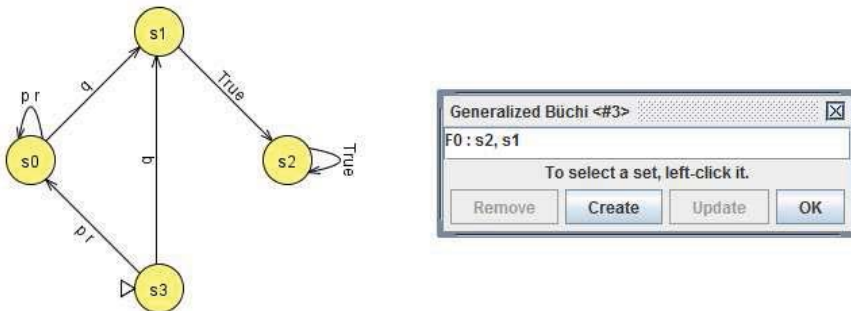
Here is an simple example to show the effect. First, we translate formula $\phi = (p \mathcal{U} q) \wedge (r \mathcal{U} q)$ by GPVW and the result GBA \mathcal{A}_ϕ contains fourteen states. An automaton which contains so many states cannot be easily understand by people. We also translate the formula ϕ with GPVW+ and the result GBA \mathcal{A}'_ϕ contains seven states. On the other hand, we transform ϕ to $\phi' = (p \wedge r) \mathcal{U} q$ and translate ϕ' with GPVW. The result automaton contains only four states, which is much smaller than \mathcal{A}_ϕ and \mathcal{A}'_ϕ .



(a) The result GBA of formula $(p \mathcal{U} q) \wedge (r \mathcal{U} q)$ by GPVW.



(b) The result GBA of formula $(p \mathcal{U} q) \wedge (r \mathcal{U} q)$ by GPVW+.



(c) The result GBA of formula $(p \wedge r) \mathcal{U} q$ by GPVW.

Figure 4.2: Examples GBAs for rewrite formula rules proposed in [6]

4.2 Formula Translation

Here, we are going to introduce an algorithm which is used for linear temporal formulae to generalized Büchi automata translation in “on-the-fly” fashion. GPVW is a simple on-the-fly algorithm proposed in [8]. It stores the information of elementary formulae, \mathcal{U} -formulae, and the right-hand side formulae of \mathcal{U} -formulae in each state. In Addition, the \mathcal{U} -formulae are also used to keep the information of the accepting condition of the result generalized Büchi automata. Considering a \mathcal{U} -formula f , when the right-hand side formula of f holds, the \mathcal{U} -formula is also satisfied which implies if the right-hand side formula of an \mathcal{U} -formula is satisfied at the current state of the result generalized Büchi automata, the \mathcal{U} -formula is also satisfied at the current state. Therefore, the current state will be collected in the accepting set of the result automata corresponding to the \mathcal{U} -formula. They also proposed a new way to detect the contradiction and redundancies for states in the later parts of the paper.

The central idea of GPVW translation algorithm is a tableau-like procedure related to ones described in [25, 26]. The tableau-like procedure actually constructs a graph. The nodes and the arcs in the graph represent the states and the transitions of the result automaton. The data fields we use to represent the graph nodes contain sufficient information for the graph construction algorithm to be able to operate in a DFS order, which are as follows:

- Name
 - A name that distinguish each node
- Incoming
 - A set of nodes with outgoing edge leading to the current node.
- New
 - A set of subformulae which must be hold at the current node and not yet be processed.
- Old

form	New1(form)	Next1(form)	New2(form)
$\mu \mathcal{U} \phi$	$\{\mu\}$	$\{\mu \mathcal{U} \phi\}$	$\{\phi\}$
$\mu \mathcal{R} \phi$	$\{\phi\}$	$\{\mu \mathcal{R} \phi\}$	$\{\mu, \phi\}$
$\mu \vee \phi$	$\{\mu\}$	\emptyset	$\{\phi\}$

Table 4.3: Definitions of New and Next functions for non-literals

- A set of atomic propositions which is hold in the current node.
- Next
 - A set of subformulae which must be hold in all states that are immediate successors of states satisfying the properties in *Old*.
- Father
 - Nodes will be split durion the construction of the graph. This field, *Father*, will be the name of the node which the current one has been split from. It is used for reasoning about the correctness of the algorithm only, and is not that important for the algorithm.

Before we go any further, we should first notice that \mathcal{F} -formula and \mathcal{G} -formula can always be transformed to \mathcal{U} -formula and \mathcal{R} -formula. Therefore, without loss of generality, we can assume the input formula does not contain the Eventually operator ‘ \mathcal{F} ’ and Always operator ‘ \mathcal{G} ’ and is in negation normal form. The first step of the algorithm which is showed in Fig. 4.3 is to generate an initial node for the input formula. The initial node has a single incoming edge, labeled **init**, which marks the fact that it is an initial node. There is only one element in the field *New*, which is the input formula ϕ . Both the fields *Old* and *Next* are empty in the initial node. The core function *expand* is showed in Fig. 4.4. With the current node N , the algorithm checks if there are unprocessed obligations left in *New*. When η , which is the target expanding formula in *New*, is not a literal, the algorithm may splits the node with different formulae set to hold. The splitting rules is illustrated in Table 4.3.

If η is actually in the form $\mu \wedge \phi$, the current node need not to be spilt. Instead, both μ and ϕ are added to *New* as the truth of both formula is needed to make η hold. During

the processing the current node, a formula η in *New* is removed from this list. In the case that η is a proposition or the negation of a proposition, then

- If $\neg\eta$ is in *Old*, the current node is discarded.
- Otherwise, η is added to *Next*.

If there already is a node $n \in Node_Set$ with the same obligations in both its *Old* and *Next* fields, merge the current node and n . If no such node exists in *NodeSet*, then the current node is added to this list, and a new current node is formed for its successor as follows:

1. There is initially one edge from N to the new current node.
2. The set *New* is the set initially to the *Next* field of N .
3. The sets *Old* and *Next* of the new current node are initially empty.

The list of nodes in *Node_Set* can now be converted into a generalized Büchi automaton $B = (\Sigma, S, \delta, s_0, F)$:

1. Σ consists of sets of propositions from AP.
2. The set of states S includes the nodes in *NodeSet* and the additional state s_0 .
3. The initial state is s_0 .
4. $(s, w, s') \in \delta$ iff $s \in Incoming(s')$ and w satisfies the conjunction of the negated and non-negated propositions in *Old*(s').
5. The acceptance set F contains a separate set of states $F_i \in F$ for each subformulae of the form $p\mathbf{U}q$; F_i contains all the states s such that either $q \in Old(s)$ or $p\mathbf{U}q \in Old(s)$.

4.3 Automata Transformation

In this section, we will introduce Safra's construction [21, 9], which is an approach that transforms a Büchi automaton into an equivalent deterministic Muller automaton or deterministic Rabin automaton. Some algorithms mentioned in Chapter 5 will be applied on deterministic Muller automata.

Algorithm: Create Graph**input** : Formula ϕ **output**: Set of nodes $Node_set$ **begin**

return (expand($[Name \leftarrow Father \leftarrow newname(), Incoming \leftarrow \{init\},$
$New \leftarrow \{\psi\}, Old \leftarrow \emptyset, Next \leftarrow \emptyset], \emptyset)$

end

Figure 4.3: The create graph algorithm.

4.3.1 Safra Tree

Safra tree is an extension of the usual subset construction.

Definition 4.10. *Safra tree is a labeled ordered tree where $T = (N, r, p, \psi, l)$.*

- N is a set of nodes. The name of the nodes are token from some global natural number \mathbb{N} .
- r is the root node. The name of root is 1.
- $p : N \rightarrow N$ is a parenthood function defined over $N - \{r\}$, and defining for every $v \in N - \{r\}$, its parent $p(v) \in N$.
- ψ is the sibling ordering relation, a partial order relation. if $p(v) \neq p(v')$ then there is no order between v and v' , while if $p(v) = p(v')$ then $(v, v') \in \psi$ or $(v', v) \in \psi$. (v, v') means v is an order sibling of v' .
- $l : N \rightarrow 2^Q$ is a labeling of the nodes with subsets of Q . The labels of two siblings are disjoint and the label of every node is a superset of the union of the labels of its sons.

In the usual subset construction, we keep track all reachable states, say S , after reading a prefix of the input word. For an input $\sigma \in \Sigma$, Safra tree will not only remember the successor states of S , but also create a new node to record all the states that can be reached from $S \cap F$ after reading σ . When the label of a node is equal to the union of the label of its children, which means all the states in this computation has a run that visited an accepting state, we will mark this node with a special sign, say '!', and remove all the descendants.

Algorithm: expand**input** : A single node $Node$, Set of nodes $Node_set$ **output**: Set of nodes $Node_set$ **begin**

```

if New(Node) =  $\emptyset$  then
  if  $\exists ND \in Node\_set$  with Old(ND) = Old(Node) then
    Incoming(Node) = Incoming(ND)  $\cup$  Incoming(Node)
    return Node_set
  else
    return (expand([Name  $\leftarrow$  Father  $\leftarrow$  newname(),
      Incoming  $\leftarrow$  {Name(Node)}, New  $\leftarrow$  Next1(Node), Old  $\leftarrow$   $\emptyset$ ,
      Next  $\leftarrow$   $\emptyset$ ], {Node}  $\cup$  Node_set))
  end
else
  let  $\eta \in New$ 
  New(Node) := New(Node)  $\setminus$  { $\eta$ }
  switch  $\eta$  do
    case  $\eta = P_n$ , or  $\neg P_n$  or  $\eta = r$  or  $\eta = F$ 
      if  $\eta = F$  or Neg( $\eta$ )  $\in$  Old(Node) then // Current node contains a
        contradiction // Discard current node
        then return Node_set
      else
        Old(Node) := Old(Node)  $\cup$  { $\eta$ }
        return expand(Node, Node_set)
      end
    case  $\eta = \mu \mathcal{U} \psi$ , or  $\mu \mathcal{R} \psi$ , or  $\mu \vee \psi$ 
      Node1 := [Name  $\leftarrow$  newname(), Father  $\leftarrow$  Name(Node),
        Incoming  $\leftarrow$  Incoming(Node),
        New  $\leftarrow$  New(Node)  $\cup$  ({New1( $\eta$ )}  $\setminus$  Old(Node)),
        Old  $\leftarrow$  Old(Node)  $\cup$  { $\eta$ }, Next  $\leftarrow$  Next(Node)  $\cup$  {Next1( $\eta$ )}] Node2 :=
        [Name  $\leftarrow$  newname(), Father  $\leftarrow$  Name(Node),
        Incoming  $\leftarrow$  Incoming(Node),
        New  $\leftarrow$  New(Node)  $\cup$  ({New2( $\eta$ )}  $\setminus$  Old(Node)),
        Old  $\leftarrow$  Old(Node)  $\cup$  { $\eta$ }, Next  $\leftarrow$  Next(Node)  $\cup$  {Next2( $\eta$ )}] return
        expand(Node2, expand(Node1, Node_Set))
    case  $\eta = \mu \wedge \psi$ 
      return expand([Name  $\leftarrow$  newname(), Father  $\leftarrow$  Father(Node),
        Incoming  $\leftarrow$  Incoming(Node),
        New  $\leftarrow$  New(Node)  $\cup$  ({ $\mu, \psi$ }  $\setminus$  Old(Node)),
        Old  $\leftarrow$  Old(Node)  $\cup$  { $\eta$ }, Next  $\leftarrow$  Next(Node)  $\cup$  {Next2( $\eta$ )}],
        Node_Set)
  endsw
end
end
end

```

Figure 4.4: The expand function.

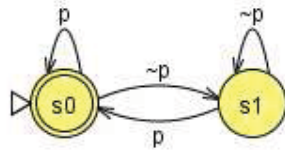
4.3.2 Nondeterministic Büchi to Deterministic Muller and Deterministic Rabin

Let $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$ be a nondeterministic Büchi automaton with n states. There exists an equivalent deterministic Muller automaton $\mathcal{M} = (\Sigma, Q', \delta', q'_0, \mathcal{F})$ or an equivalent deterministic Rabin automaton $\mathcal{R} = (\Sigma, Q', \delta', q'_0, \Omega)$ where

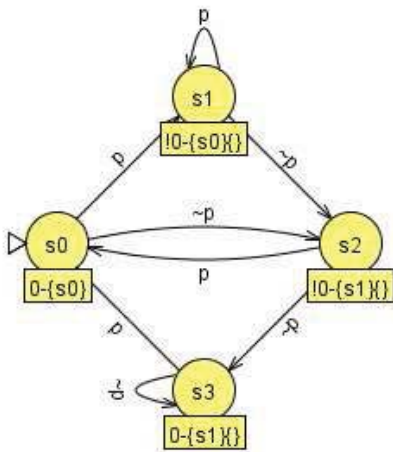
- Q' is a set of Safra tree.
- The initial state q'_0 is the Safra tree consisting of only node 1 labeled $\{q_0\}$.
- $\delta' : Q' \times \Sigma \rightarrow Q'$ is the transition function. For a given input $a \in \Sigma$ and a Safra tree T , $\delta'(T, a)$ is computed as follows:
 1. Remove all marks in the Safra tree T .
 2. For every node v with label L such that $L \cap F \neq \emptyset$, create a new child with label $L \cap F$.
 3. Apply the usual subset construction on every nodes. Replace the label with $\bigcup_{q \in L} \delta(q, a)$.
 4. For every node v with label L and $q \in L$, if q also belongs to the label of an older brother of v , then remove q from L .
 5. Remove all nodes with empty labels.
 6. For every node whose label is equal to the union of the labels of its sons, remove all the descendants of the node and mark the node with '!'.
 1. node i appears in all Safra trees of F_i , and
 2. node i is marked at least once in F_i .
- The Muller acceptance condition $\mathcal{F} = \{F_1, \dots, F_{2n}\}$ is defined as for each F_i ,
 1. node i appears in all Safra trees of F_i , and
 2. node i is marked at least once in F_i .
- The Rabin acceptance condition $\Omega = \{(E_1, F_1), \dots, (E_{2n}, F_{2n})\}$ is defined as for $i = \{1, 2, \dots, 2n\}$,
 1. E_i is the set of all Safra trees without a node i , and
 2. F_i is the set of all Safra trees with node i marked.

Note that deterministic Rabin automaton can be used when computing the complement of a Büchi automaton. First, transform the Büchi automaton into an equivalent deterministic Rabin automaton. Second, get the deterministic Streett automaton for the complement language by dualizing the acceptance condition of the deterministic Rabin automaton. Last, transform the deterministic Streett automaton into an equivalence Büchi automaton by Proposition 3.8, which is the complement of the original Büchi automaton.

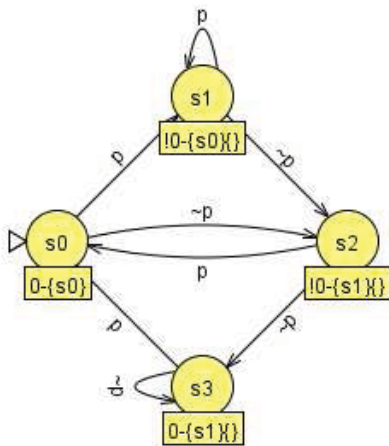




(a) A nondeterministic Büchi automaton \mathcal{B} for $\square\Diamond p$



(b) A deterministic Muller automaton \mathcal{M} where $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{B})$



(c) A deterministic Rabin automaton \mathcal{R} where $\mathcal{L}(\mathcal{R}) = \mathcal{L}(\mathcal{B})$

Figure 4.5: An example of translating a nondeterministic Büchi automaton into an equivalent deterministic Muller automaton and an equivalent deterministic Rabin automaton

Chapter 5

Classification

In this chapter, we will introduce temporal hierarchy classification proposed in [17] and classification of a property as deterministic or nondeterministic Büchi.

5.1 Temporal Hierarchy

We say that a program P has the property Π if all the computations of P belong to Π . One can use properties to specify a program, but it may lead to *underspecification* in some cases. To avoid underspecification, one possible solution is to classify different types of properties, and provide a list of properties to the specifier to consider. L. Lamport suggested that properties of a reactive program can be partitioned off into two classes, *safety* and *liveness* properties [14]. In 1990, Z. Manna and A. Pnueli proposed a hierarchy of temporal properties. They classified temporal properties into six classes [17].

5.1.1 The Temporal Logic View

A formula contains no future operators is called a *past* formula.

- A *safety formula* is a formula of the form

$$\Box p,$$

for a past formula p .

- A *guarantee formula* is a formula of the form

$$\Diamond p,$$

for some past formula p .

- A *simple obligation formula* is a formula of the form

$$\Box p \vee \Diamond q,$$

where p and q are past formulae.

- A *general obligation formula* is a formula of the form

$$\bigwedge_i [\Box p_i \vee \Diamond q_i],$$

where p_i and q_i are past formulae.

- A *recurrence formula* is a formula of the form

$$\Box \Diamond p,$$

for some past formula p .

- A *persistence formula* is a formula of the form

$$\Diamond \Box p,$$

for some past formula p .

- A *simple reactivity formula* is a formula of the form

$$\Box \Diamond p \vee \Diamond \Box q,$$

where p and q are past formulae.

- A *general reactivity formula* is a formula of the form

$$\bigwedge_i [\Box \Diamond p_i \vee \Diamond \Box q_i],$$

where p_i and q_i are past formulae.

Figure 5.1 shows the inclusion relations between the classes. The classes of safety and guarantee properties are disjoint, so do recurrence and persistence properties. The safety properties are the *closed* sets and the guarantee properties are the *open* sets. Every temporal formula is equivalent to a reactivity formula.

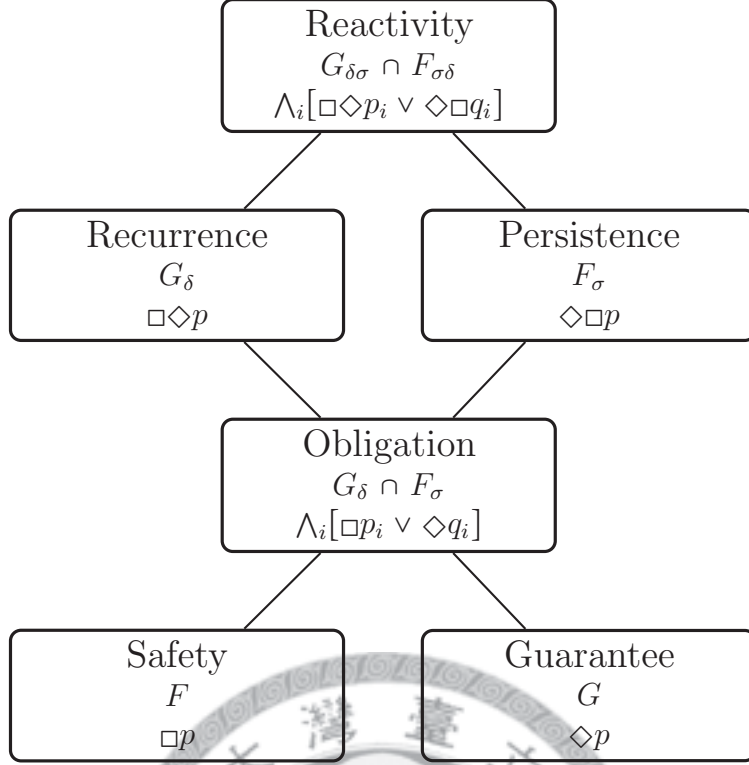


Figure 5.1: Inclusion relations between the classes

5.1.2 The Automata View

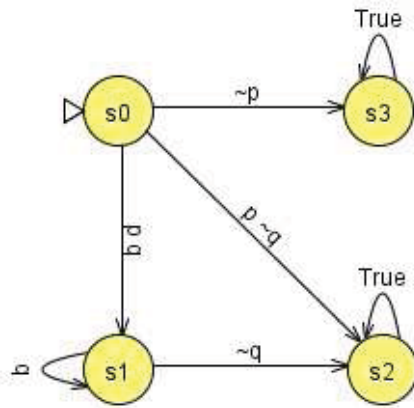
Another way to specify temporal properties is based on finite-state predicate automata. In this thesis, we focus on deterministic Streett automata as predicate automata. First, let us restrict our attention to a special case of Streett automata \mathcal{M} , where $\mathcal{M} = (\Sigma, Q, \delta, q_0, (R, P))$, which means there is only one pair in the acceptance condition. R is a set of *recurrent* automaton-states, and P is a set of *persistent* automaton-states. A run ρ on an infinite word w is accepted by \mathcal{M} iff either $\text{inf}(\rho) \cap R \neq \emptyset$ or $\text{inf}(\rho) \subseteq P$.

Let

- $G = R \cup P$ be referred to as the “good” sets of states and
- $B = Q - G$ be referred to as the “bad” sets of states.

Z. Manna and A. Pnueli defined the following classes of automata by introducing restrictions on their transition conditions and accepting states.

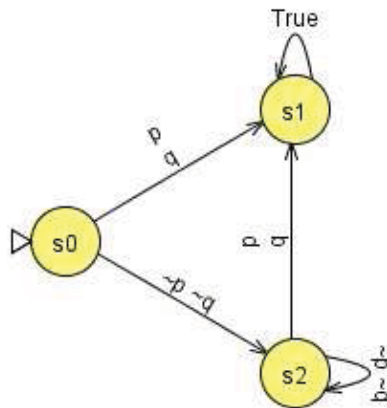
- A *safety* automaton is such that there is no transition from $q \in B$ to $q' \in G$.



$$(R, P) = (\{s0, s1, s3\}, \{\})$$

A safety automaton of $p \rightarrow \Box q$.

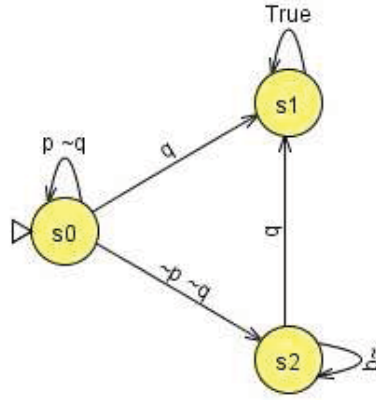
Figure 5.2: An example of safety automata



$$(R, P) = (\{s1\}, \{\})$$

A guarantee automaton of $\Diamond(p \vee q)$.

Figure 5.3: An example of guarantee automata



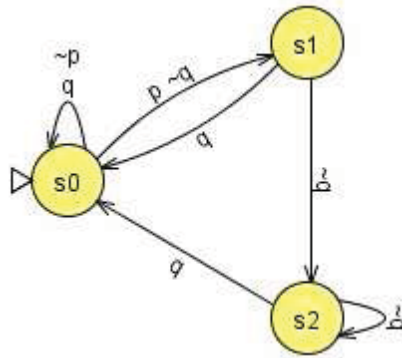
$$(R, P) = (\{s1\}, \{s0\})$$

A simple obligation automaton of $\Box p \vee \Diamond q$.

Figure 5.4: An example of obligation automata

- A *guarantee* automaton is such that there is no transition from $q \in G$ to $q' \in B$.
- A *simple obligation* automaton is such that
 - There is no transition from $q \notin P$ to $q' \in P$.
 - There is no transition from $q \in R$ to $q' \notin R$.
- A (general) *obligation* automaton (of degree k) is an automaton, in which each state $q \in Q$ has a rank $\gamma(q)$, $0 \leq \gamma(q) \leq k$, such that:
 - There is a transition from q to q' only if $\gamma(q) \leq \gamma(q')$.
 - There is a transition from $q \in B$ to $q' \in G$ only if $\gamma(q) < \gamma(q')$.
 - There is no transition from a state $q \in G$ of rank k to a state $q' \in B$.
- A *recurrence* automaton is such that $P = \emptyset$.
- A *persistence* automaton is such that $R = \emptyset$.
- A *simple reactivity* automaton is an unrestricted automaton of the above type.

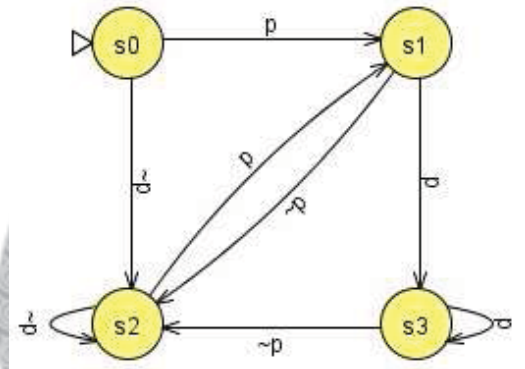
Now, we want to apply the classification technique to more general deterministic Streete automaton \mathcal{M} , where $\mathcal{M} = (\Sigma, Q, \delta, q_0, \Omega)$, and $\Omega = \{(R_1, P_1), \dots, (R_n, P_n)\}$. A run ρ on an infinite word w is accepted by \mathcal{M} iff for each $(R_i, P_i) \in \Omega$, either $\text{inf}(\rho) \cap R_i \neq \emptyset$ or $\text{inf}(\rho) \subseteq P_i$.



$$(R, P) = (\{s0\}, \{\})$$

A recurrence automaton of $\Box(p \rightarrow \Diamond q)$.

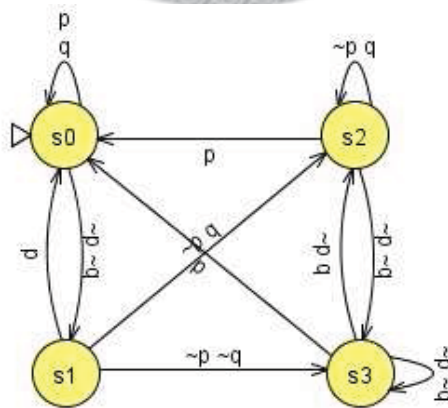
Figure 5.5: An example of recurrence automata



$$(R, P) = (\{\}, \{s0, s3\})$$

A persistence automaton of $\Diamond\Box p$.

Figure 5.6: An examples of persistence automata



$$(R, P) = (\{s0\}, \{s1, s2\})$$

A simple reactivity automaton of $\Box\Diamond p \vee \Diamond\Box q$.

Figure 5.7: An examples reactivity of automata

The following are some definitions.

A set of automaton states $A \subset Q$ is defined to be *closed* if for every $q, q' \in Q$

$$q \in A \wedge \exists \sigma \in \Sigma, \delta(q, \sigma) = q' \longrightarrow q' \in A.$$

The closure \hat{A} of a set of states is the smallest closed set containing A .

For a given Streete automaton \mathcal{M} , define

$$G = \bigcap_{i=1}^k (R_i \cup P_i).$$

- Checking for a *safety* property.

Let $B = Q - G$. The automaton \mathcal{M} specifies a safety property iff $\hat{B} \cap G = \emptyset$.

- Checking for a *guarantee* property.

\mathcal{M} specifies a guarantee property iff $\hat{G} \cap B = \emptyset$.

To check for the other levels of the hierarchy, we define the family of accepting set F .

$$F = \{J \mid J \text{ is an accessible cycle, } J \cap R_i \neq \emptyset \text{ or } J \subseteq P_i \text{ for each } i = 1, \dots, k\}$$

- Checking for a *recurrence* property.

\mathcal{M} specifies a recurrence property iff for every $J \in F$ and every accessible cycle $A \supseteq J, A \in F$.

- Checking for a *persistence* property.

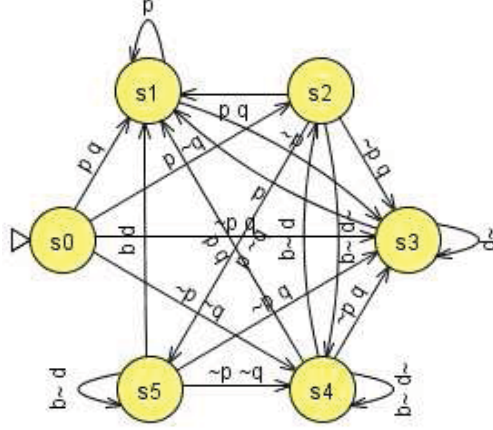
\mathcal{M} specifies a persistence property iff for every $J \in F$ and every accessible cycle $B \subseteq J, B \in F$.

- Checking for a *guarantee* property.

\mathcal{M} specifies a guarantee property iff there do not exist three accessible cycles

$$B \subseteq J \subseteq A$$

such that $J \in F$, but $B, A \notin F$.



$$\{(R_1, P_1), (R_2, P_2)\} = \{(\{s0\}, \{s0, s1, s2, s5\}), (\{s0, s1, s3, s4\}, \{s2, s6\})\}$$

A (general) reactivity automaton of $\diamond\Box p \wedge \diamond q$.

Figure 5.8: An examples of reactivity automata

5.2 Deterministic Büchi v.s. Nondeterministic Büchi

Deterministic Büchi automata has less expressive power than nondeterministic Büchi automata, while nondeterministic Büchi automata, deterministic Muller automata, deterministic Rabin automata, deterministic Streett automata and deterministic parity automata are all equivalent in expressive power. Classification of a property as deterministic or nondeterministic Büchi helps complementation and empty checks. In [15], L. Landweber uses Borel hierarchy to classify languages of different complexity.

Let $\mathcal{M} = (\Sigma, Q, T, q_0)$ be a finite automaton (f.a.) where T is a function such that $T : Q \times \Sigma \rightarrow Q$.

Definition 5.1. $\bar{T} : Q \times \Sigma^* \rightarrow Q$ is the extension of T given by $\bar{T}(q, x\sigma) = T(\bar{T}(q, x), \sigma)$ for $\sigma \in \Sigma$, $x \in \Sigma^*$. $R_{\mathcal{M}}$ is a function, $R_{\mathcal{M}} : \Sigma^* \rightarrow Q$, given by $R_{\mathcal{M}}(x) = \bar{T}(s_0, x)$, called the response function of \mathcal{M} . (To simplify the notation we omit the subscript \mathcal{M} in $R_{\mathcal{M}}$.)

Let $w = w_0w_1\cdots$ be a member of Σ^ω . Abbreviate $w_0w_1\cdots w_i$ by \bar{w}_i and define the partial order $<$ on $\Sigma^* \cup \Sigma^\omega$ by $\bar{w}_i < \bar{w}_j < w$ for $i < j < \omega$. Let $P(S)$ be the set of all subsets of the set S .

1. Let $F \subseteq Q$. \mathcal{M} accepts w with respect to F if $\exists i. R(\bar{w}_i) \in F$.
- 1'. Let $F \subseteq Q$. \mathcal{M} accepts w with respect to F if $\forall i. R(\bar{w}_i) \in F$.
2. Let $F \subseteq Q$. \mathcal{M} accepts w with respect to F if $\text{inf}(\rho) \cap F \neq \emptyset$.
- 2'. Let $\mathcal{F} \subseteq P(Q)$. \mathcal{M} accepts w with respect to \mathcal{F} if $\exists F \in \mathcal{F}. \text{inf}(\rho) \subseteq F$.
3. Let $\mathcal{F} \subseteq P(Q)$. \mathcal{M} accepts w with respect to \mathcal{F} if $\exists F \in \mathcal{F}. \text{inf}(\rho) = F$.

Definition 5.2. An *i-f.a.* is a *f.a.* augmented by an output of type *i*.

Note that a 2-f.a. is actually a deterministic Büchi automaton and a 3-f.a. is a deterministic Muller automaton.

Definition 5.3. $A \subseteq \Sigma^\omega$ is *i-definable* if there is an *i-f.a.* which defines it.

Let F be the closed subset of Σ^ω and G be the open subset of Σ^ω .

F_σ is the set of countable unions of closed subsets of Σ^ω .

G_δ is the set of countable intersections of open subsets of Σ^ω .

$F_{\sigma\delta}$ is the set of countable intersections of F_σ -subsets of Σ^ω .

$G_{\delta\sigma}$ is the set of countable unions of G_δ -subsets of Σ^ω .

The hierarchies $F, F_\sigma, F_{\sigma\delta}$ and $G, G_\delta, G_{\delta\sigma}$ form the *Borel hierarchy*.

Theorem 5.4. Every 1-definable set is in G , and every 1'-definable set is in F .

Theorem 5.5. Every 2-definable set is in G_δ , and every 2'-definable set is in F_σ .

Theorem 5.6. Every 3-definable set is in $G_{\delta\sigma} \cap F_{\sigma\delta}$.

In the following, let $\mathcal{M} = (\Sigma, Q, T, q_0, \mathcal{D})$ be a 3-f.a., which is also a deterministic Muller automaton.

Definition 5.7. For $x, y \in \Sigma^*$, $x < y$, let $\mathcal{R}(x, y) = \{R(z) \mid x \leq z \leq y\}$. For $q \in Q$, let $Ac(q) = \{s \mid s \in Q, \exists x. \bar{T}(q, x) = s\}$. Call $Ac(q)$ the set of states accessible from q and $\mathcal{R}(x, y)$ the state path determined by the interval x, y .

Definition 5.8. For $q \in Q$, let $\mathcal{H}_q = \{\mathcal{R}(x, y) \mid R(x) = R(y) = q, x, y \in \Sigma^*\}$, the set of realizable cycles.

Theorem 5.9. $\mathcal{L}(\mathcal{M})$ is in G_δ if and only if for all $q \in Q$, $D \in \mathcal{D} \cap \mathcal{H}_q$ and $E \in \mathcal{H}_q$ implies $D \cup E \in \mathcal{D}$. [15]

From the above theorem, we know how to check whether a deterministic Muller automaton \mathcal{M} can be transformed into an equivalent deterministic Büchi automaton. The construction of the equivalent deterministic Büchi automaton is shown below.

Theorem 5.10. If $A \in G_\delta$ is 3-definable, then it is 2-definable. [15]

Let $\mathcal{L}(\mathcal{M}) \in G_\delta$, where $\mathcal{M} = (\Sigma, Q, T, q_0, \mathcal{D})$ is a 3-f.a. Now a 2-f.a. \mathcal{M}^* satisfying $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}^*)$ is defined as follows. For each $q \in Q$, let \mathcal{M}_q be an f.a. which for any input sequence w satisfies:

- (a) \mathcal{M}_q enters a designated state ϵ the first time \mathcal{M} would enter q in reading w ;
- (b) \mathcal{M}_q reenters ϵ each time and only at such times that (1) \mathcal{M} is in state q and (2) the set of states entered by \mathcal{M} in reading w , since the previous time \mathcal{M}_q was in ϵ , is in \mathcal{D} .

\mathcal{M}^* is $(\mathcal{M}_{q_0} \times \dots \times \mathcal{M}_{q_n}, D^*)$, where $Q = \{q_0, \dots, q_n\}$, \times is the usual product operation on machines and

$$D^* = \{(d_0, \dots, d_n) \mid d_i \text{ a state of } \mathcal{M}_{q_i}, \exists j. (d_j = \epsilon)\}$$

is the Büchi acceptance condition.

$\mathcal{M}_{q_i} = (\Sigma, Q', T', q'_0)$ can be constructed as follows:

- $Q' = (Q \setminus \{q_i\}) \cup \{\epsilon\} \cup \{\langle q, S \rangle \mid q \in Q, S \subseteq Q\}$,
- $q'_0 = q_0$,
- $T' : Q' \times \Sigma \rightarrow Q'$ is defined, for $q, \langle q, S \rangle \in Q'$ and $\sigma \in \Sigma$, as follows.

$$\begin{aligned} - T'(q, \sigma) &= \begin{cases} \epsilon & \text{if } T(q, \sigma) = q_i, \\ T(q, \sigma) & \text{otherwise.} \end{cases} \\ - T'(\epsilon, \sigma) &= \langle T(q_i, \sigma), \{T(q_i, \sigma)\} \rangle, \end{aligned}$$

$$- T'(\langle q, S \rangle, \sigma) = \begin{cases} \epsilon & \text{if } T(q, \sigma) = q_i \wedge S \cup \{T(q, \sigma)\} \in \mathcal{D}, \\ \langle T(q_i, \sigma), S \cup \{T(q, \sigma)\} \rangle & \text{otherwise.} \end{cases}$$

Two examples are shown in Figure 5.9, 5.10 and 5.11. Note that states with double circles in \mathcal{M}_q denote ϵ states.

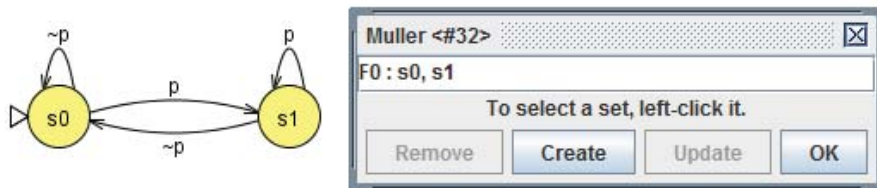
Here we propose an improvement of this construction. Instead of generating \mathcal{M}_q for every $q \in Q$, we generate \mathcal{M}_q for all $q \in S$, where S is a subset of Q such that, for each $D \in \mathcal{D}$, $S \cap D \neq \emptyset$. For example, \mathcal{M}_{s_0} and \mathcal{M}_{s_2} in Figure 5.10 can be ignored, and \mathcal{M}_{s_1} is exactly the result \mathcal{M}^* .

To prove the correctness of this improvement, the proof is similar to the proof described in [15]. Note that \mathcal{M} is built into each \mathcal{H}_q . In the following, $Inf(\alpha)$ and \mathcal{H}_q always refer to \mathcal{M} .

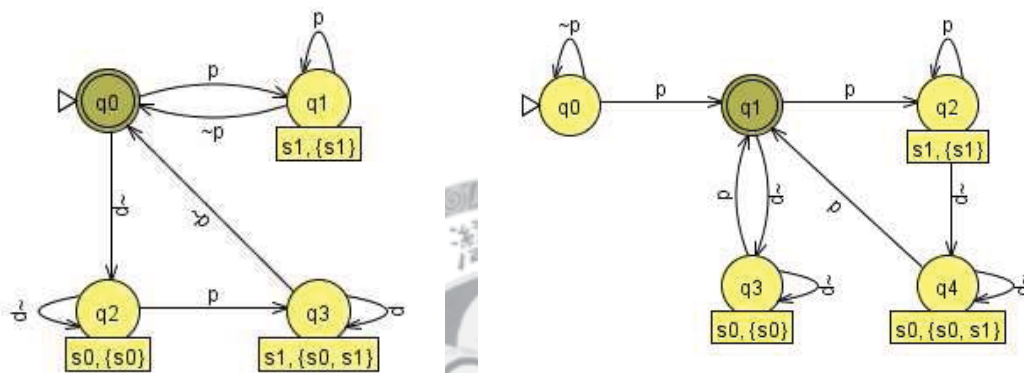
Proof. 1. $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}(\mathcal{M}^*)$. Let $\alpha \in \mathcal{L}(\mathcal{M})$ and $Inf(\alpha) \in \mathcal{D}$. $S \cap Inf(\alpha) \neq \emptyset$ since $S \cap D \neq \emptyset$ for each $D \in \mathcal{D}$. Choose $q \in S \cap Inf(\alpha)$. \mathcal{M}_q enters ϵ the first time \mathcal{M} enters q , while reading α .

Assume \mathcal{M}_q enters ϵ for the n th time at time t and let E_1, E_2, \dots be the sets of states which \mathcal{M} enters between successively entering q after time t ($E_i \neq \emptyset, i = 1, 2, \dots$, since $q \in Inf(\alpha)$). There is a finite sequence $E_j, E_{j+1}, \dots, E_{j+k}$ such that $Inf(\alpha) = \bigcup_{l=0}^k E_{j+l}$. Since $\mathcal{H}_q \cap \mathcal{D} \neq \emptyset$, Theorem 5.9 implies that $\bigcup_{l=1}^{j+k} E_l \in \mathcal{D}$. Hence \mathcal{M}_q enters ϵ an $(n+1)$ st time. This proves that if $\alpha \in \mathcal{L}(\mathcal{M})$, then some \mathcal{M}_q enters ϵ infinitely often, so that $\alpha \in \mathcal{L}(\mathcal{M}^*)$.

2. $\mathcal{L}(\mathcal{M}^*) \subseteq \mathcal{L}(\mathcal{M})$. Let $\alpha \in \mathcal{L}(\mathcal{M}^*)$, so that there is a q such that \mathcal{M}_q enters ϵ infinitely often while reading α . Let E_i ($i = 1, 2, \dots$) be the set of states entered by \mathcal{M} between the i th and $(i+1)$ st times \mathcal{M}_q enters ϵ . Then $E_i \in \mathcal{H}_q \cap \mathcal{D}$ ($i = 1, 2, \dots$) by the definition of \mathcal{M}_q . $Inf(\alpha)$ must be equal to a finite union of E_i 's, but since $\mathcal{H}_q \cap \mathcal{D} \neq \emptyset$, Theorem 5.9 implies that any finite union of E_i 's is in \mathcal{D} . Hence $\alpha \in \mathcal{L}(\mathcal{M})$. \square

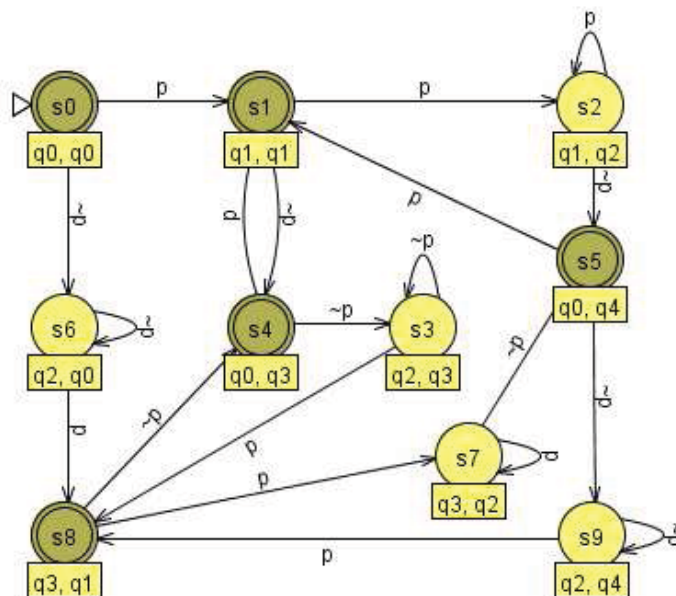


(a) The deterministic Muller automaton \mathcal{M} where $L(\mathcal{M}) = \square \diamond p \wedge \square \diamond \neg p$



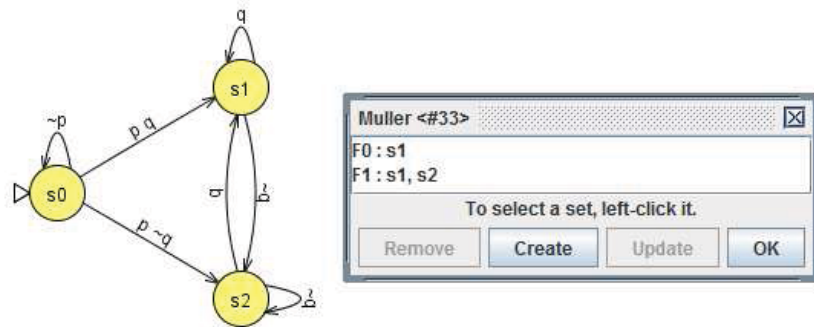
(b) Result of \mathcal{M}_{s_0}

(c) Result of \mathcal{M}_{s_1}

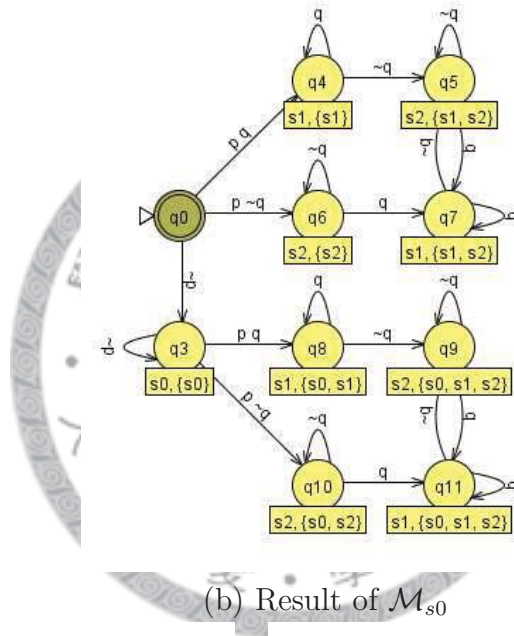


(d) The deterministic Buchi automaton \mathcal{M}^* which is equivalent to \mathcal{M} .

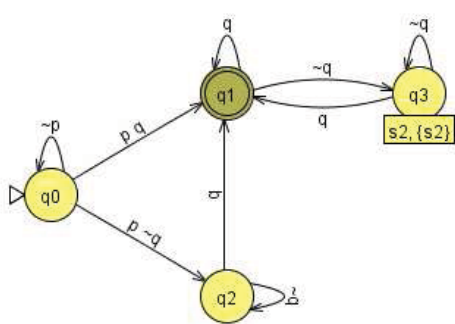
Figure 5.9: Examples of translating DMW into DBW



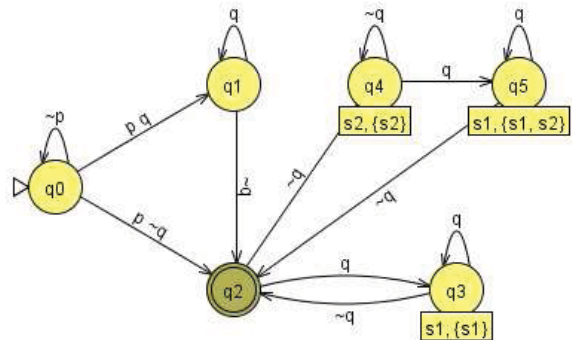
(a) The muller automaton \mathcal{M} where $L(\mathcal{M}) = \diamond(p \wedge \square \diamond q)$



(b) Result of \mathcal{M}_{s_0}

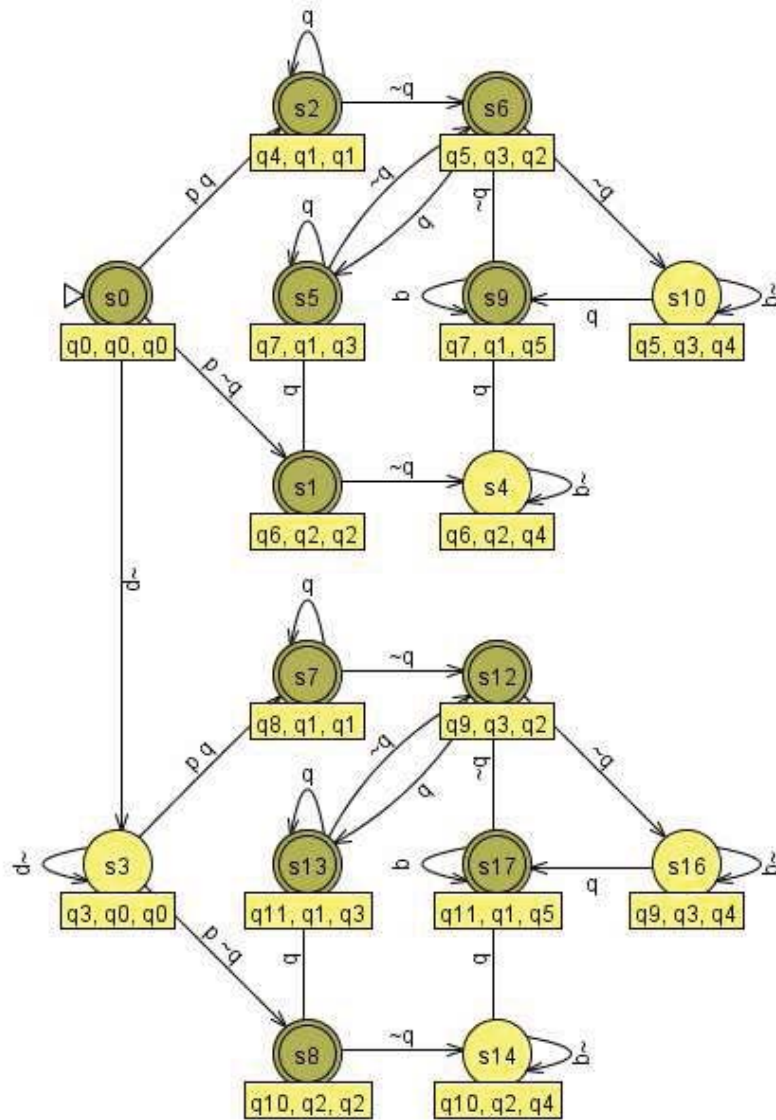


(c) Result of \mathcal{M}_{s_1}



(d) Result of \mathcal{M}_{s_2}

Figure 5.10: Another examples of translating DMW into DBW



(e) The deterministic Büchi automaton \mathcal{M}^* which is equivalent to \mathcal{M} .

Figure 5.11: Another examples of translating DMW into DBW

Chapter 6

Implementation and Applications

6.1 Implementations in GOAL

All the algorithms described in Chapter 4 and Chapter 5 have been implemented in GOAL. Some of the algorithms, which I have used to accomplish my thesis, were implemented by the previous authors.

In the implementation of the temporal hierarchy classification algorithm described in Section 5.1, for an input formula f , there will be four steps:

1. Translate $\neg f$ into an equivalent Büchi automaton $\mathcal{B}_{\neg f}$.
2. Convert $\mathcal{B}_{\neg f}$ into an equivalent deterministic Rabin automaton $\mathcal{R}_{\neg f}$.
3. Obtain the deterministic Streett automaton \mathcal{S}_f , which is equivalent to f , by dualizing the acceptance condition.
4. Apply the classification algorithm on \mathcal{S}_f .

The major weakness of this procedure is that we may not be able to obtain the “best” Streett automaton \mathcal{S}_f to classify the formula f into the lowest class where it belongs to. To classify an arbitrary Büchi automaton \mathcal{B} , we can first get the complement Büchi automaton $\bar{\mathcal{B}}$. Then convert $\bar{\mathcal{B}}$ into an equivalent deterministic Rabin automaton $\bar{\mathcal{R}}$, and get the dual deterministic Streett automaton \mathcal{S} , which is equivalent to \mathcal{B} . The classification algorithm can be applied on \mathcal{S} .

In the implementation of the algorithm that converts a deterministic Muller automaton into an equivalent deterministic Büchi automaton, which is described in Section 5.2,

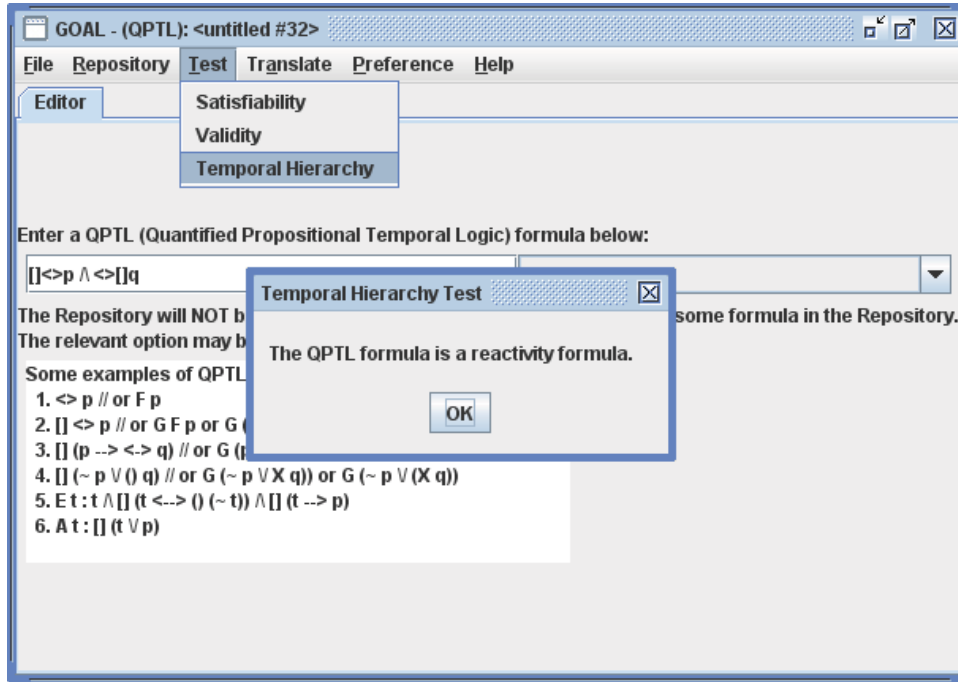


Figure 6.1: Test of temporal hierarchy classification for an input formula in GOAL

the improvement we proposed is also implemented. To implement the improvement, we use a simple method to compute the set S .

1. Put the state q that covers the most accepting sets into S .
2. If there exists any accepting sets that has not been covered by the states in S , choose the state that covers the most remaining accepting sets and put it into S .
3. Repeat step 2. until every accepting set is covered by at least one state in S .

Table 6.1 shows the experiment result of the converting algorithm for 15 random cases. As we can see, the size of the result Büchi automata reduced when we applied our improvement.

6.2 Applications on the Büchi Store

Based on the implementation on GOAL, the Büchi Store becomes more useful. There are three main change of the Büchi Store, which as follows:

- *Smaller and Deterministic BAs can be generated and collected*

As we described in Chapter 4, with the rewriting rules we implemented in GOAL,

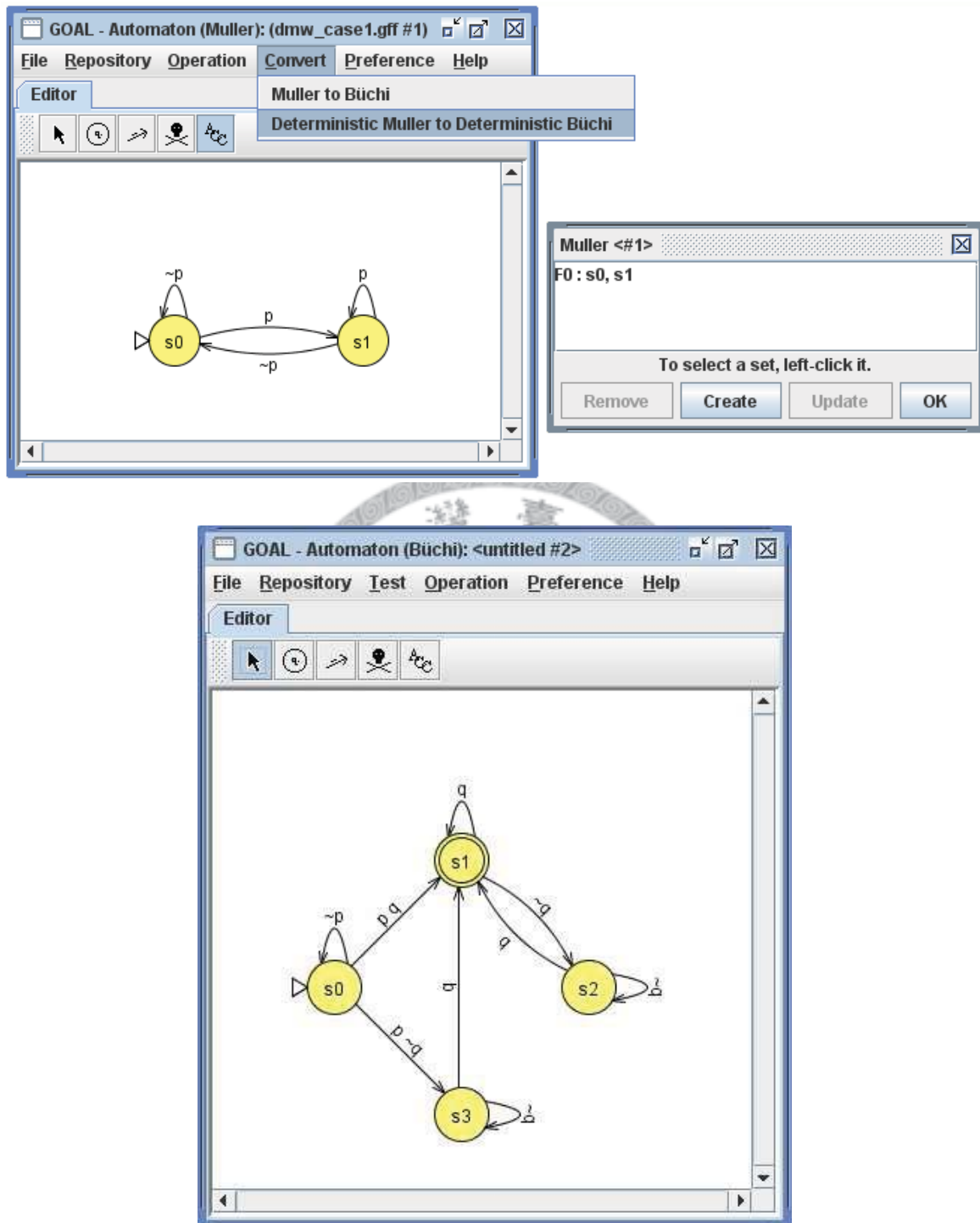


Figure 6.2: Convert a deterministic Muller automaton to a deterministic Büchi automaton by the power of GOAL

Case no.	DMW		DBW		DBW with improvement	
	st.	tran.	st.	tran.	st.	tran.
1	2	3	4	8	2	4
2	4	8	8	32	6	24
3	4	6	10	40	4	16
4	4	8	36	72	11	22
5	4	8	18	72	5	20
6	3	6	12	48	3	12
7	4	10	11	44	8	32
8	3	5	6	12	4	8
9	5	13	70	280	18	72
10	6	12	22	88	13	52
11	4	12	15	120	10	80
12	4	10	36	144	11	44
13	8	21	33	132	15	60
14	6	10	11	44	7	28
15	5	11	86	344	13	52
total	66	143	378	1480	130	526

Table 6.1: The experiment result of deterministic Muller to deterministic Büchi converting algorithm for 15 random cases.

one can generate smaller BA in terms of size for the same formulae. When one generated smaller BA, He can share these interesting BAs on the Büchi Store, and wins the credit of smallest BA ever from the Büchi Store. He then will be encouraged to searching for smaller BAs. Furthermore, one can also generate deterministic BA for formulae. The Büchi Store will provide a deterministic BA pool for these deterministic BA since deterministic BA are useful in some research and academy area. These user's behavior will definitely enrich the BA pool in the Büchi Store.

- *The efficiency and correctness of categorizing formulae are improved*

Last year, the Büchi Store provides a very simple way to categorize the temporal formulae. We had a simple method to category the formulae which cannot recognize all kinds of formulae, which is the first part of 5.1. If the formulae cannot be categorized by the method, we try to categorize it by hands. Now, we have implemented the second part of the method, which means the ability of categorizing input formula is improved. Moreover, with the help of rewriting formulae method, we are able to categorized one formula more efficiently because the intermediate

Streett automaton might be reduced. This improvement provides a better user experiment when people uploading their user-defined BA to the Büchi Store.

- *Searching is more functional and humane*

For people who visiting the Büchi Store and trying to search for a BA for specific formula, the search engine can not only search for the input formula literally, but also search for any equivalent formula with the help of the rewrite roles we had built in. The Büchi Store will check the possible formula by rewriting the input formula into a simplest one and checking whether there is a formula which is literally equivalent with the rewritten one. People will have more opportunity to reach the BA which he/she is interested in for his/her own defined formula.

With these improvements, the Büchi Store becomes more helpful for research, practice, and education.



Search

Formula (Syntax)
 Description
 Author
 ID
 All

Sorted by

- [Formula Length](#)
- [State Size](#)
- **[Temporal Hierarchy](#)**
- [Spec Patterns](#)

Response (Recurrence)

Suggested browsers: [Mozilla Firefox](#) and [Google Chrome](#)

<< [First page](#) < [Pre](#) 1 - 15 of 57 [Next](#) > [Last page](#) >>

Select a category to browse

- [Safety](#)
- [Guarantee](#)
- [Obligation](#)
- [Response \(Recurrence\)](#)
- [Persistence](#)
- [Reactivity](#)
- [Unknown](#)

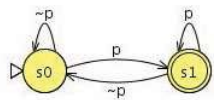
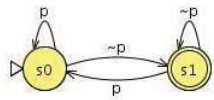
	<p>$[]\langle\rangle p$</p> <p>Equivalent: $[(\text{True } U \text{ } p)] \cdot \langle\rangle [\langle\rangle p]$</p> <p>Complement: $\langle\rangle (\sim p \text{ } W \text{ } (\text{False } \wedge \sim p)) \cdot \sim [(\text{True } U \text{ } p)]$ $\cdot \sim [\langle\rangle p] \cdot [\langle\rangle [\sim p] \cdot \sim \langle\rangle [\langle\rangle p] \cdot \langle\rangle [\sim p]$</p>
	<p>$[]\langle\rangle \sim p$</p> <p>Equivalent: $[(\text{True } U \text{ } (\sim p \wedge \text{True}))] \cdot [(\text{True } U \text{ } \sim p)] \cdot \sim \langle\rangle (\text{False } R \text{ } p) \cdot \sim \langle\rangle (p \text{ } W \text{ } \text{False}) \cdot \sim \langle\rangle [p] \cdot \langle\rangle [\langle\rangle \sim p] \cdot \sim [\langle\rangle [p]]$</p> <p>Complement: $\langle\rangle (\text{False } R \text{ } p) \cdot \langle\rangle (p \text{ } W \text{ } \text{False}) \cdot [\langle\rangle [p] \cdot \langle\rangle [p]$</p>

Figure 6.3: Browse automata sorted by temporal hierarchy in the Büchi Store

Chapter 7

Conclusion

Study of transformation and classification of temporal properties is important because it helps us classify temporal properties better. People can find one BA of a given temporal formula faster and the corresponding BA of a formula may become smaller with the help of formulae transformation and classification. In this thesis, we discussed several transformation and classification methods for temporal properties. We also described the improvements that we made to GOAL and the Büchi Store based on these methods.

7.1 Contributions

The contributions of this thesis can be summarized as follows.

- *Implementation of classification rules for temporal properties*

The temporal hierarchy classification algorithm and conversion of deterministic Muller automaton to deterministic Büchi automaton are implemented in GOAL. The user now can test which hierarchy the input formula belongs to. The topic about properties classification has not been extensively studied. With this thesis and our implementation, the reader can learn more about the difference between classes. The implementations are also applied to the Büchi Store.

- *Improvement of the conversion of DMW to DBW*

In [15], Landweber gave the rules for construction of \mathcal{M}_q when converting a deterministic Muller automaton into an equivalent deterministic Büchi automaton. We proposed a construction of \mathcal{M}_q based on their work. We also made some improvements so that we can get a smaller automaton. One can get more understanding

of the conversion algorithm through our implementation.

- *Improvements on the open repository Büchi Store*

The Web-based open repository is very useful for people who are interested in temporal logic or program verification. The Büchi Store provides a collection of temporal formulae and their corresponding smallest Büchi automata. All the temporal formulae are classified appropriately with our implementations of the classification algorithm. Hence, the user can easily search for a desired formula to obtain the smallest corresponding automaton without trying all the translation algorithms, which would cost lots of time and space.

7.2 Future Work

There are several directions for the future:

- *Study and implementation of more simplification algorithms*

One major purpose of this thesis is to make it easier for people to get smaller Büchi automata. One simplification algorithm proposed in [22] has been implemented in GOAL. There are more simplification algorithms that can help obtain smaller automata. Moreover, the algorithm that converts a deterministic Muller automaton into an equivalent deterministic Büchi automaton described in Section 5.2 generates a large deterministic Büchi automaton. It should be worthwhile to investigate simplification algorithms for DBW, since smaller DBW will be more useful.

- *Formulae equivalence checking*

Formulae equivalence checking can also be considered as one kind of formulae classification methods. However, the semantics equivalence test between two formulae is usually realized by the equivalence check of the corresponding automata, which is costly. It should be worthwhile to develop a more direct method for checking the equivalence between two temporal formulae.

- *Development of more functionality and interactive GUIs for the Büchi Store*

The Büchi Store now is simply an open Web-based repository. Functions like searching for BA without temporal formula, viewing most often downloaded automata, or

showing comments from users of a Büchi automaton would be a valuable addition in the Büchi Store. Formulae other than temporal formulae should also be allowed to describe the corresponding BA in Büchi Store. Besides, the user always feels easy to browse with suitable interactive GUIs. Online temporal formula rewriting, online BA editing, and online laying out of automaton may be useful in the Büchi Store to improve the ability to interact with users.



Bibliography

- [1] 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. Stanford University Press, 1962.
- [2] J.-S. Chang. A comprehensive comparison temporal formula to automata translation algorithms. Master’s thesis, Institute of Information Management, National Taiwan University, 2009.
- [3] Y. Choueka. Theories of automata on ω -tapes: A simplified approach. *Journal of Computer and System Science*, pages 8:117–141, 1974.
- [4] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of Workshop on Logic of Programs, LNCS 131*, pages 52–71, 1981.
- [5] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.
- [6] K. Etessami and G. Holzmann. Optimizing Büchi automata. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000), LNCS 1877*, pages 153–167. Springer, 2000.
- [7] 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.
- [8] 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.

- [9] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games (LNCS 2500)*. Springer, 2002.
- [10] G. J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.
- [11] G.J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.
- [12] N. Klarlund. Progress measures for complementation of omega-automata with applications to temporal logic. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1991)*, pages 358–367, 1991.
- [13] O. Kupferman and M. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(3):408–429, 2001.
- [14] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977.
- [15] L. H. Landweber. Decision problems for ω -automata. *Mathematical Systems Theory*, 3(4):376–384, 1969.
- [16] Christof Löding. Methods for the transformation of ω -automata: Complexity and connection to second order logic. Master’s thesis, Christian-Albrechts-University of Kiel, 1998.
- [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] 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.

- [20] A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
- [21] S. Safra. On the complexity of ω -automata. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1988)*, pages 319–327, 1988.
- [22] 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.
- [23] 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.
- [24] 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.
- [25] P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 28(110):119–136, 1985.
- [26] Pierre Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983.