Introduction to Automaton Theory

on infinite trees

By Huang Yuantao

Supervised By Prof. Riccardo Camerlo

Table of contents

1 Introduction . . . 5

1.1 Finite and infinite words . . . 5
1.2 Finite and infinite trees . . . 8

2 Tree Automata . . . 12

2.1 Automata on finite trees . . . 12
2.2 Automata on infinite trees . . . 15
2.2.1 Büchi tree automata . . . 16
2.2.2 Muller tree automata . . . 19
2.2.3 Rabin basis theorem . . . 23

3 Games on tree automata . . . 26

3.1 Automaton and pathfinder . . . 26
3.2 Rabin's tree theorem . . . 28
3.3 Still Rabin's basis theorem . . . 29

4 Topology . . . 31

4.1 The definition of topology of infinite trees . . . 31
4.1.1 Prelinimary knowledge . . . 31
4.1.2 The space of infinite trees . . . 32
4.2 Suslin sets . . . 33
4.3 The topological complexity of recognizable sets of trees . . . 34
4.4 Wadge hierarchy of tree languages . . . 35

5 Monadic second order logic of two successors . . . 37

5.1 Introduction to monadic second-order logic . . . 37
5.2 S2S . . . 39

6 Practical uses of tree automata . . . 43

6.1 Applications on XML . . . 43
6.1.1 A basic for pattern languages and pattern test . . . 44
6.1.2 A method of processing queries and schema languages . . . 46
6.1.3 An algorithm toolbox . . . 48
6.2 Concurrent program verification . . . 50
6.3 Applications on NLP . . . 51

6.4 Synthesizing reactive programs 52
6.5 Automating data completion 54

7 Conclusion 57

Bibliography 59

Abstract

We study tree automaton here, and mainly on infinite trees. In the first part of thesis,we will introduce how there is a generalization from words to trees and what is the definition of infinite tree. Then we introduce the tree automata working on finite and infinite trees. Büchi automata and Muller automata classically corresponds to the acceptance mode of infinite trees. We will introduce them and their recognizability and later the theoretical importance will be introduced together. Next a game used to simulate the possible runs of a tree automaton will be introduced, which can prove Rabin's complementation theorem. In the next part, we shall study the trees from the topological aspects. Subsequently, the relations between logics and tree automata will be showed. Last but not least, the practical uses of the tree automaton will be showed.

1 Introduction

In this thesis, trees defined here serve as functions who map from words to sets of labels. So before we introduce the infinite trees and their automata, we need to have a general mind of how words are defined.

1.1 Finite and infinite words

At the very first, we need to define the words and review 4 basic operations of the sets.

A finite sequence of elements of a set A, which can also be called alphabet here, is called a finite word on A. For example, if A is $\{a_0, a_1, ...,$ $a_n\}$, $a_0 a_1$ for sequence (a_0, a_1) and $a_0 a_1 ... a_n$ for sequence $(a_0, a_1, ..., a_n)$ are both finite words on A. Additionally, it owns the empty word, called neutral element, denoted by ε for the empty sequence.

4 basic operations on the sets are as following:

(1) the set union $\cup$:

$$X \cup Y = \{u | u \in X \text{ or } u \in Y\}$$

(2) the set concatenation product $\cdot$:

$$X \cdot Y = \{xy | x \in X \text{ and } y \in Y\}$$

(3) the star $*$:

$$X^* = \{x_1 ... x_n | n \geqslant 0 \text{ and } x_1, ..., x_n \in X\}$$

which means finite iteration.

(4) the omega ω:

$$X^\omega = \{x_0 x_1 ... \mid \text{for all } i \geqslant 0, x_i \in (X \text{ exclusive with } \{1\} \text{ or } \{\varepsilon\}\ )\}$$

where X,Y is a set.

Suppose we have two words here, $x = a_0 a_1 ... a_p$ and $y = b_0 b_1 ... b_q$, then the word xy = $a_0 a_1 ... a_p b_0 b_1 b_q$. We can see the operation here is associative. Based on the set theory, we denote by A^+ the set of nonempty words, which is called the free semigroup on A since its operation is associative and A^* the set of empty or nonempty words, which is called the free monoid on A since it has not only the associative operation but also a neutral element. We can see that $A^* = A^+ \cup \{1\}$ and $A^+ = AA^*$.

Similarly, we call an infinite sequence of elements of A the infinite word on A, which we denote by the following:

$$u = a_0 a_1 ... a_n ...$$

where u is an infinite word and a_i is a letter. We denote the set of infinite words on A by A^ω.

And we let

$$A^\infty = A^* \cup A^\omega$$

which is an overall set.

The elements of a subset of A^* sometimes can't be compared for an order, then the subset is said to be prefix-free. For example, if $A = \{a, b\}$, the set $\{a^n b | n \geqslant 0\}$ is prefix-free. On the other hand, a subset of A^* is called prefix-closed if it contains the prefixes of all of its elements. In particular, if X is a subset of A^+ or of A^ω, the set Pref(X), which stands for all prefixes of the elements of X, is prefix-closed.

Another important concept is for the definition of rational sets of words. The operations $\cup$, $\cdot,^*$ and $^+$ introduced above are all rational operations. We define that the sets are rational sets if they are closed under finite union, finite product and finite Kleene star operations. We can immediately see that all finite subsets of A^* are rational sets. If u is a finite word, the singleton $\{u\}$ apparently belongs to the class of rational sets. Furthermore, if $u = a_1 ... a_n$, $\{u\} = \{a_1\} ... \{a_n\}$.

Let's extend to the definition of ω-rational sets. Just adding one property, we define that the sets are ω-rational if they are closed under finite union, finite product, the star and the ω operations. We can derive a theorem:

Theorem 1. *A subset of A^ω is ω-rational if and only if it is a finite union of sets of the form XY^ω where X and Y are rational subsets of A^*.*

Proof. *We denote by $\mathcal{R}$ the class of w-rational subsets of A^ω.*

From the definition of ω-rational sets and rational subsets, it's apparent that it's as the form XY^ω.

For the converse direction, we can first see what properties a rational subset X of A^∞ has. Then we construct a class of subsets who has these properties and if it also satisfies the definition of the class of ω-rational subsets of A^∞. Then the theorem proved.

X has the following properties:

i. $X \cap A^*$ *is a rational subset of* A^*.

ii. $X \cap A^\omega \in \mathcal{R}$ *and when* $X \subset A^\omega$, $X \in \mathcal{R}$.

Let $\mathcal{Q}$ be the class of subsets of A^∞ satisfying the properties above and we now check if $\mathcal{Q}$ has the essential properties of the class of ω-rational subsets.

a) *If* $u \in A$, $\{u\} \in \mathcal{Q}$ *and* $\varnothing \in \mathcal{Q}$.

b) *If* $X \in \mathcal{Q}$ and $Y \in \mathcal{Q}$, $X \cup Y \in \mathcal{Q}$. $\mathcal{Q}$ *is closed under finite union.*

c) *If* $X \in \mathcal{Q}$ *and* $Y \in \mathcal{Q}$, $X \cdot Y \in \mathcal{Q}$. *Actually, we can prove it in this way. Since* $X \subset A^*$ *and* $Y \subset A^\infty$, $X \cdot Y \cap A^* = X \cdot (Y \cap A^*)$ *is rational for the first property of* $\mathcal{Q}$ *and* $X \cdot Y \cap A^\omega = X \cdot (Y \cap A^\omega) \in \mathcal{R}$ *for the second property of* $\mathcal{Q}$. $\mathcal{Q}$ *is closed under finite product.*

d) *If* $X \in \mathcal{Q}$, *so does* X^*. $\mathcal{Q}$ *is closed under the Kleene star operation.*

e) *If* $X \in \mathcal{Q}$, *so does* X^ω. $\mathcal{Q}$ *is closed under omega operation.* □

1.2 Finite and infinite trees

According to the related notions above, we can introduce finite and infinite trees now. Since we mainly focus on infinite trees and the corresponding automata here, the definitions will be specific, which are of valued trees here. They correspond to a relation over ω-words or just ω-relation and tree automata can be applied to the relation. They are so expressive that they can encode an arbitrary set of words, any nondeterministic system or a strategy in an infinite game.

Suppose that the two alphabets D and A are both non empty. The elements of D and the elements of A are respectively called directions and labels. Then we say a tree like this is a D-tree on A. It's defined as a map $t\colon P \longrightarrow A$ where P is a nonempty subset of D^* and it is prefix-closed. The set of all D-trees on the alphabet A is denoted by $T(D, A)$. The set P then is known as the tree's domain, denoted by $\mathrm{Dom}(t)$. Each element of P is a node of the tree. If $x \in P$ is a node of t, then each node, who forms like xi for $i \in D$, is called a child of x.

Additionally, if the domain of a tree is a finite set, the tree is also said to be finite. And among the elements of domain, some have the max length. The length of these elements is called the height of a finite tree. For example,

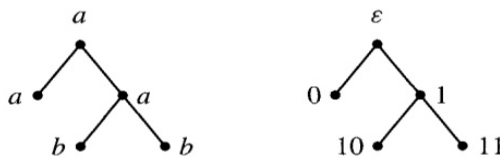

Figure 1.

which is a finite tree and its domain. As is shown, it's a very simple binary tree. $A = \{a, b\}$ and D here as a set of directions, whose values are 0 and 1. $P = \{\varepsilon, 0, 1, 10, 11\}$ is a subset of D^* and is apparently closed by prefix as defined above.

We can easily imagine that when D contains only one symbol, a tree will become a subset of A^* closed by prefix, just like a word! Specifying a tree t and a corresponding word $u \in D^*$, symbol t_u is denoted as the tree defined by $t_u(x) = t(ux)$.

Let's define the leaves of the tree now. As is known, we call the node who has no child nodes a leaf node. In a formal way, suppose an element $x \in P$, if $x D \cap P = \varnothing$, x is a leaf of the tree t. We denote by $\mathrm{Fr}(t)$ the set of the leaves of t:

$$\mathrm{Fr}(t) = \{x \in P \mid x D \cap P = \varnothing\}$$

which is called the frontier of t. The outer frontier of t is equivalent to the set $PD - P$, denoted by $\mathrm{Fr}^{+}(t)$. For instance, corresponding to the figure above, the outer frontier is the set $\{00, 01, 100, 101, 110, 111\}$, which is like a shell of the domain. Regarding the domain, it's defined that $\mathrm{Dom}^{+}(t) = \mathrm{Dom}(t) \cup \mathrm{Fr}^{+}(t)$.

Regard to the finite tree, a path π via the tree t is actually a chain from the root to one of the leaves. Extended to the scope of D^{∞}, we say that π through t is a maximal chain,which is defined as a chian in a poset to which no element can be added without losing the property of being totally ordered, for the prefix order in Dom(t). As we choose one direction on every node, a path has a form exactly like the sequence of prefixes of a finite or infinite word in D^{∞}. According to our binary-relationed definition on tree, we denote by $t \mid \pi$ the restriction of t to π and $t \mid \pi$ is a word on the alphabet A. For example, if π is a sequence $(\varepsilon, 1, 10)$ of the domain in the figure above, $t \mid \pi = aab$, of which b here is the left one.

Using the definitions mentioned until now, if we just consider the infinite trees, the domain will be infinitely large. The generalization from finite words to finite trees will also apply: when we let the cardinality of tree be 1 or just choose a path in the tree, it will become an path or word on the alphabet A.

The tree on which we've applied the examples until now is a very simple finite and full binary tree. In theory, D can be very large,which means there can be many directions on each node, and the tree can be multiply splitting. But considering those situations will not be helpful for our topic, so we limit the scope of discussion to the full binary trees for simplicity. Formally, $D = \{0, 1\}$ and each node of the tree has no or two child nodes. We denote by T_A the set of finite trees on alphabet A and by T_A^{ω} the set of infinite trees. Let $T_A^{\infty} = T_A \cup T_A^{\omega}$ be the overall set instead of $T(D, A)$.

The basic difference between words and trees is the cardinality. Card(t) equals to an integer greater than 1. Therefore, the operations between the trees will be a little bit different from the ones between words. But as we'll see, the essence is somehow similar and those of trees are more general.

Let's define the concatenation product of trees now. We let $S \subset T_A$ be a set of finite trees, $T \subset T_A^{\infty}$ be a set of finite or infinite ones and $c \in A$. And the concatenation product can be written as $S \cdot_c T$. We can obtain the set of products like this: replace in the tree $s \in S$ with each appearance of c on Fr(t) by a tree from T (various branches of trees of T may replace for various presences of c). We can recall that if there are a set of finite words and a set finite or infinite words, the concatenation product is just the set that contains all the possible words whose left factor is the one in the first set and right factor is the one in the second set. For the trees, the situation is a little bit more complex, but we can consider it in this way: the only place we can concatenate a tree into another tree is to find the suitable leaves. As is the same with the relations between words, the ones between trees are associative. Formally, it means that, if S and $T \subset T_A$ are sets of finite trees, $U \subset T_A^{\infty}$ is a set of finite or infinite trees and c, $d \in A$. Then the equation $(S \cdot_c T) \cdot_d U = S \cdot_c (T \cdot_d U)$ holds.

In exactly the same way, for the star operation of a set, a tree need to be defined with a definite symbol $c \in A$. And we denote by $T^{n,c}$ the set defined by the methods induction:

$$T^{0,c} = \{c\}\,, \text{and} \quad T^{n,c} = T \cdot_c T^{n-1,c} \cup T^{n-1,c}$$

for $T \subset T_A$ and an integer $n \geqslant 0$. Correspondingly, we set

$$T^{*,c} = \bigcup_{n \geqslant 0} T^{n,c}$$

where we can see c is always the rightmost factor of the elements.

According to the definition of the rational set in the set theory (closed under union, concatenation product and star operations), we can define if a set of trees $T \subset T_A$ is rational. We say if there is a finite set of C containing A, then the the set $T \subset T_A$ is considered to be retional so that a finite number of combinations can be made. From finite subsets of T_C, the set T can be taken by finite unions, concatenations $\cdot_c$ and stars $*, c$ for $c \in C$.

Now let's consider in a more specific way on the concatenation product between trees, where instead of a single symbol, a tuple $c = (c_1, ..., c_m)$ is used and each element in the tuple corresponds to a tree in the tree set. For instance, for $T, T_1, ..., T_m \subset T_A$, we denote by $T \cdot_c (T_1, ..., T_m)$ and c_i corresponds to T_i.

Now let's define the infinite product. Let $c = (c_1, ..., c_m)$ and let $T_1, ..., T_m \subset T_A$. For the set of infinite trees t, $(T_1, ...T_m)^{\omega,c}$ is denoted. Considering infinite sequences $(t_0, t_1, ...)$ of trees for every of sequence like this $t_0 \in \{c_1, ..., c_m\}$ and for $n \geqslant 0$, the relation holds $t_{n+1} \in t_n \cdot_c (T_1, ..., T_m)$. So for every t_i, there is a extension on one of the leaves of it as the next tree. Therefore, there will be a extension of all trees in a common format and we can denote it by t.

Next we can define the ω-rationality of a set. The set $T \subset T_A^\omega$ of infinite trees is defined as ω-rational if there exists a finite set $C = \{c_1, ..., c_m\}$ and several sets $T_0, T_1, ..., T_m \subset T_{A \cup C}$, which are all rational, such that

$$T = T_0 \cdot_c (T_1, ..., T_m)^{\omega,c}$$

For the simplicity of notaion, here we denote $t \cdot_c (t_1, ..., t_m)$ instead of $\{t_0\} \cdot_c (\{t_1\}, ..., \{t_m\})$. We apply a similar denotion to it when t_i's are infinite trees with a rule such that the first occurences of the c_i are used for replacement instead of frontier occerences of the c_i.

Recall that for infinite words, a subset of A^ω of infinite words is ω-rational if and only if it is a finite union of sets of the form $X \cdot Y^\omega$ where X and Y are rational subsets of A^*. We can find that the definition of ω-rationality for infinite trees generalizes from the one for infinite words. An additional definition of set C is for finding the appropriate node to concatenate the tree in the set.

2 Tree Automata

From the section, let's begin to introduce the tree automata, which will be the main topic of the thesis. In this section, we define the automata working on finite and infinite trees, of which two important forms named Büchi automata and Muller automata, which has more power, will be the main focus.

But before all of these, let's recall what the automaton generally is. Automata are automatic machines or machines or control mechanisms designed to automatically follow predetermined operational sequences or respond to predetermined instructions.

Conventional automata are based on the operations on words, where there can only be one state before or after the corresponding state transfer. As an extension of the conventional ones, tree automata are based on the operaions on words, where allow multiple, or we say a set of, states before or after the state transfer. And this is the essential difference between conventional automata and tree automata. We can see that tree automata are suitable for the operations on trees.

For automata, the notion of recognizabilty and deterministic or non-deterministic ones are very important. These properties directly point out the power of an automaton. However, these properties of tree automata are quite different with the ones of word automata. Tree automata are thought to have several strong points respect to word automata:

1. Tree automata suits for modeling non-determinism more.
2. There is a closer connection between logical theories and tree automata. The Rabin's theorem states that we can reduce the problems in logic to the correspongding problems in tree automata.

We'll discuss it in next sections.

2.1 Automata on finite trees

A more general tree automaton on the alphabet A can be defined as a tuple (Q, D, E, I, F), where D is a finite set of all possible directions. But as we presented above, we use a specific $D = \{0, 1\}$ for simplicity

and all tree automata including infinite tree automata will always use the specific D.

A tree automaton on the alphabet A can be defined as a tuple (Q, A, E, I, F), where Q is a finite set of states, $E \subset Q \times A \times Q \times Q$ is a set of edges of tree automaton, $I \subset Q$ is a set of initial states and $F \subset Q$ is a set of final states. E here is like a set of transition rules, but as we mentioned above, here all trees are restricted as full binary trees, so there are exactly two next states transtioned from one state (top-down) or one from two (bottom-up).

With an automaton $\mathcal{A}$ on the tree t, we can have a run. In a formal way, the run is defined as a map r: $\mathrm{Dom}^+ \to Q$ with $r(\varepsilon) \in I$ such that $(r(x), t(x), r(x_0), r(x_1)) \in E$ for all $x \in \mathrm{Dom}(t)$. We say that the run is successful if $r(\omega) \in F$ for all ω on the outer frontier $\mathrm{Fr}^+(t)$ of t. We can see here the use of outer frontier. Meeting the leaves of a tree, the automaton needs to verify whether there are next states.

A simple example will be very helpful. Using a tree like the one mentioned in the first section, the tree and its domain are like this:

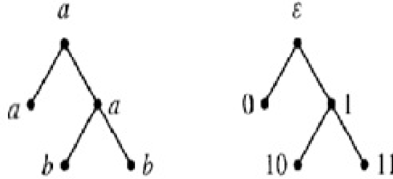

Figure 2. A tree t and its domain

$A = \{a, b\}$, $\mathrm{Dom}(t) = \{\varepsilon, 0, 1, 10, 11\}$, $\mathrm{Fr}^+(t) = \{00, 01, 100, 101, 110, 111\}$ and let $Q = \{1, 2\}$, $I = \{1\}$, $F = \{2\}$ and

$$E = \{(1, a, 1, 1), (1, b, 2, 2)\}$$

Thus, $r(\varepsilon) = 1 \in I$, $r(0) = 1$, $r(1) = 1$, $r(10) = 1$ and $r(11) = 1$. For $r(\omega) \in F$, $r(00) = r(01) = 1$ and $r(100) = r(101) = r(110) = r(111) = 2$. We can get a successful run on t:

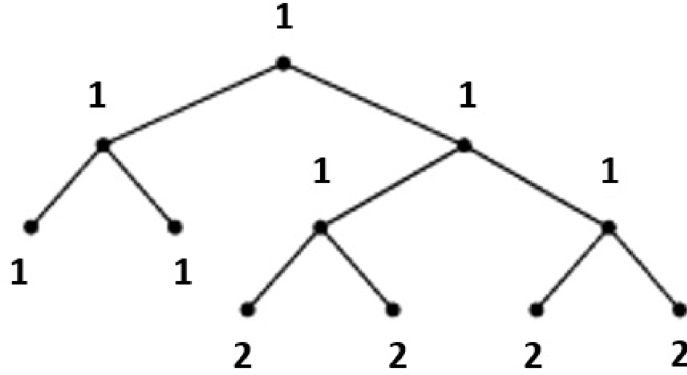

Figure 3. A run on T

The finite tree set of automatic recognition is formed by every tree, so the operation on $\mathcal{A}$ is achieved successfully. If the set $T \subset T_A$ can be recognized by an existed tree automaton $\mathcal{A}$, then T is defined as recognizable. Therefore if $T = \{t\}$, $\mathcal{A}$ recognizes T.

As we can imagine, a tree has two ways to go: downwards and upwards. For the top-down direction, we say that a tree automaton $\mathcal{A} = (Q, E, I, F)$ is top-down deterministic if $\mathrm{Card}(I) = 1$ and with an edge $(p, a, q, r) \in E$, at most one pair $(q, r) \in Q \times Q$ corresponds for each pair $(p, a) \in Q \times A$.

Correspondingly, we say that a tree automaton $\mathcal{A} = (Q, E, I, F)$ is bottom-up deterministic if $\mathrm{Card}(F) = 1$ and with a edge $(p, a, q, r) \in E$, at most one state $p \in Q$ corresponds for each triple $(a, q, r) \in A \times Q \times Q$. And we say that a bottom-up tree automaton is complete if with an edge $(p, a, q, r) \in E$, exactly one and not zero state $p \in Q$ corresponds for each triple $(a, q, r) \in A \times Q \times Q$.

With the definition of complete bottom-up deterministic tree automata, we can have the following statement.

Proposition 2. *The family of recognizable sets of trees is closed under complement.*

And we have the following Kleene's theorem for finite trees.

Theorem 3. *A set $T \subset T_A$ is recognizable if and only if it is rational.*

Now let's turn to the introduction of automata on infinite trees.

2.2 Automata on infinite trees

Now we come to the discussion of automata on infinite trees. Literally, they are state machines that deal with infinite tree structures. It's interesting that they can not only be viewed as an extension of finite tree automata but also as an extension of conventional Büchi and Muller automata on words. Actually, for the reason of modeling, trees are better than words to model nondeterminism, so unless we point out in particular, automata we mention below are all thought as nondeterministic tree automata. Another advantage of focusing on nondeterministic automata is succinctness since it can be imagined that describing tree automata is already space-consuming.

Although we'll mainly focus on two tree automaton models, Büchi and Muller, it's quite convenient to list most of formal definitions, based on the same alphabet A, of famous automata here. We'll sometimes need to use them in our discussion:

i. Büchi tree automaton: $(Q, E, I, F), F \subseteq Q$ is the set of accepting states.

ii. Muller tree automaton: $(Q, E, I, \mathcal{F}), \mathcal{F} \subseteq 2^Q$ is a family of sets of states which contains some final states.

iii. Rabin tree automaton: $(Q, E, I, \mathcal{R}), \mathcal{R}$ is a list of pairs of state sets: $\langle (E_1, F_1), ..., (E_k, F_k) \rangle$, there exists a i such that it contains finite states from E_i and infinite states from F_i.

iv. Streett tree automaton: $(Q,\ E,\ I,\ \Omega)$, Ω is a list of pairs of states:$\langle (E_1, F_1), ..., (E_k, F_k) \rangle$, there exists a i such that it contains infinite states from E_i and finite states from F_i.

v. parity tree automaton: $(Q, E, I, c), c$ named priority, is a mapping $Q \rightarrow \mathbb{N}$ that assigns a natural number to every state. It satisfies that $\max(\mathrm{Inf}(c(\pi)))$ is even.

Actually, we can see that these automata only differ according to their acceptance conditions and other letters have almost the same meanings: Q contains all possible states, E stands for the set of transitions and I or i is the set or just one initial state.

2.2.1 Büchi tree automata

Let's recall that the traditional Büchi automaton accepts an infinite input sequence, and if the automaton runs, it always visits at least one final state (at least). It is named after the Swiss mathematician Julius Richard Büchi, who discovered this kind of automaton in 1962. It's formally defined as a 5-tuple $\mathcal{A} = (Q, A, E, I, F)$, where $E \subset Q \times A \times Q$. But Büchi tree automata were formally defined by Rabin, who originally named them special automata.

A Büchi tree automaton has a slightly different definition: $\mathcal{A} = (Q, A, E, i, F)$, in which $E \subset Q \times A \times Q \times Q$ as set of edges and $i \in Q$ as the initial state. The Büchi tree automaton is usually introduced first because it's the simplest recognizing mode for infinite trees but it has weaker power than other recognizing modes, for example, the Muller tree automaton.

The run of a Büchi tree automaton $\mathcal{A}$ on a tree t ($D = \{0,1\}$) is a map $r\colon \{0,1\}^* \to Q$ with $r(\varepsilon) = i$ such that $(r(x), t(x), r(x_0), r(x_1)) \in E$ for all $x \in \{0,1\}^*$. We say that such a run will be successful if any final state will occur on all paths infinitely often, such that:

$$\mathrm{Inf}(r|\pi) \cap F \neq \varnothing$$

where π stands for any path in the tree t. If there is a successful run of $\mathcal{A}$ on an infinite tree t, then we say the automaton $\mathcal{A}$ recognizes the tree t. And the set of all infinite trees recognized by $\mathcal{A}$ is just equivalent to the set recognized by $\mathcal{A}$. Regard to Büchi tree automaton, if a set T of infinite trees is recognized by a Büchi automaton, it is called Büchi recognizable.

With a Büchi recognizable set, we have the following statement.

Theorem 4. *A set $T \subset T_A^w$ is Büchi recognizable if and only if it is ω-rational.*

The class of Büchi recognizable trees has its own closure properties: closure under finite union and projection. The proof can make us understand it better. But we will just point out the main thinking here because the thesis is mainly for introduction.

Proof. First we prove the property of closure under finite union. Suppose here is two sets of infinite trees: T_A and T_B and we need to prove that $T_A \cup T_B$ can also be recognized by some Büchi automaton. We can prove it by the method of construction. Suppose the states of two automata $\mathcal{A} = (Q_A,\ A,\ E_A,\ i_A,\ F_A)$ and $\mathcal{B} = (Q_A,\ B,\ E_B,\ i_B,\ F_B)$ are disjoint, which means $Q_A \cap Q_B = \varnothing$, then we can construct an automaton $\mathcal{C} = (Q_A \cup Q_B,\ A \cup B,\ E_A \cup E_B,\ \{i_A,\ i_B\},\ F_A \cup F_B)$ recognizing $T_A \cup T_B$. For satisfying the formal description of Büchi automaton, we use a new initial state i and a special alphabet $\{c\}$, where $\{i\} = Q_i$ and $E_i = \{(i, c, i_A, i_B)\}$. Then we get a final construction $\mathcal{C} = (Q_A \cup Q_B, A \cup B \cup \{c\}, E_A \cup E_B \cup E_i, i, F_A \cup F_B)$. Proved. □

Before we prove the closure under projection, let's first introduce the concept of projection. In set theory, a projection is defined as a mapping of a set into a subset, which has the property of idempotence. And the restriction to a subspace of a projection is also called a projection. In our specific circumstance, we define a projection as a map $\boldsymbol{\varphi}: A \rightarrow B$ where A and B are two alphabets of the tree. A map from $T(D, A)$ into $T(D, B)$, defined by $t \rightarrow \boldsymbol{\varphi} \circ t$, where $\circ$ is called the circle funtion defined as the function composition, is also called a projection.

Proof. Now we can prove it. Suppose here is a Büchi recoginzable tree set T_A, we need to prove that T_B, which is outputed by the projection of T_A, is also Büchi recognizable. By the definition, we get

$$T_B = \boldsymbol{\varphi}(T_A) = \{\boldsymbol{\varphi}(t) \,|t \in T_A\}$$

As we can see, the alphabet B is a subset of A and the statement is quite obviously true. Let $\mathcal{A} = (Q_A, A, E_A, i_A, F_A)$ and we can construct a $\mathcal{B} = (Q_B, B, E_B, i_B, F_B)$, where Q_B is a subset of Q_A, A is a subset of B, $i_B = \boldsymbol{\varphi}(i_A)$, F_B is a subset of F_A, and

$$E_B = \{(q, \boldsymbol{\varphi}(a), q', q'')\}$$

where a is a letter in A and $\boldsymbol{\varphi}$ is the corresponding letter in B and the edges from q to q' or q'' always has the corresponding edges in E_A. Then the construction is completed and the statement is proved. □

But as we said, Büchi tree automata isn't so powerful that the Büchi recognizable set of trees isn't closed under complementation. While Muller tree automata have this property, we'll discuss it later.

Let's recall that we define the ω-rationality of the set of infinite trees above. Now we have the first important theorem for our infinite tree automata:

Theorem 5. *A set $T \subset T_A^\omega$ is Büchi recognizable if and only if it is ω-rational.*

Proof. The Theorem need to be proved from two directions. Let's recall that when T is ω-rational, it must have the form $T = T_0 \cdot_c (T_1, \ldots, T_m)^{\omega, c}$. We can use a similar method of construction to build a Büchi automaton recognizing it. Then any ω-rational set is Büchi recognizable.

From the converse direction, we suppose that we have a Büchi automaton $\mathcal{A} = (Q, A, E, i, F)$ and if each set recognized by it has the format shown above, the theorem will be proved. Let $F = \{q_1, \ldots q_m\}$ and T_{q_i} be the set of all finite trees on $A \cup Q$, where the frontier nodes have values in F and other nodes in A. Here we can see the frontier nodes are valued by final states and we want to make every T_{q_i} recognizable according to the definition, so the run r which starts at q_i on the tree need to be such that: $r(x) = t(x)$, where $x \in \mathrm{Fr}(t)$. Supposing a run r on tree t starting at node p, it passes infinitely often in F. We can see every t belongs to such a set:

$$T_p \cdot_q (T_{q_1}, \ldots, T_{q_m})^{\omega, q}$$

where $q = (q_1, \ldots q_m)$. For every starting node, the format is valid. Then we finish the proof. □

Now we finish the introduction to Büchi tree automaton temporarily. We'll continually refer to it when we compare it with other tree automata.

2.2.2 Muller tree automata

Now we introduce a more powerful tree automaton, Muller tree automaton. Its definition is more general. Actually every kind of automaton differs it from others by its acceptance condition. There are other kinds of tree automata (for example, Rabin tree automaton, Streett tree automaton), but we focus on non-deterministic Muller tree automaton in the thesis becauses it's either more general than or equivalent to other kinds of automaton. It's named after David E. Muller, an American mathematician and computer scientist, who invented it in 1963. With the Muller tree automaton, we can define the regularity of a tree language. We say that a tree language is regular if any Muller tree automaton accept it.

A Muller tree automaton on the alphabet A is defined as $\mathcal{A}=(Q,A,E,i,\mathcal{F})$. It has a very similat format with a Büchi tree automaton except that there is no longer a set of final states F, but $\mathcal{F}$, which moreover stands for a set of sets of Q. We call $\mathcal{F}$ a family of subsets of Q and it defines the acceptance condition. We say that a run of $\mathcal{A}$ on a tree t is successful if for every path π of r, $\mathcal{F}$ contains such a set of states which appears infinitely often in it. It can be described in the following formal way:

$$\mathrm{Inf}(r|\pi)\in\mathcal{F}$$

From the definition, we see that the family of Büchi tree automata, either deterministic or non-deterministic, is a special case of the family of Muller tree automata. The relationship of these two kinds of automata can be shown by a figure like this:

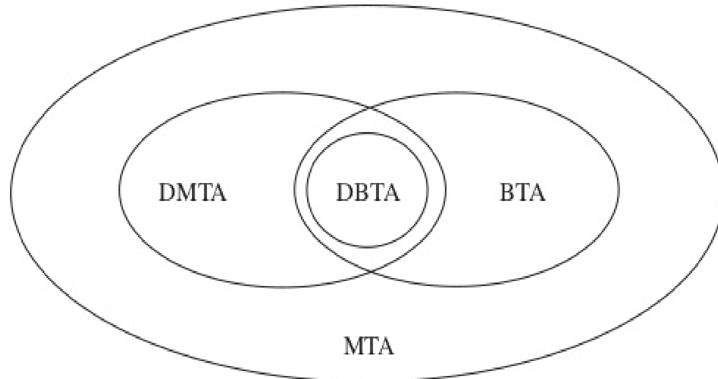

Figure 4.

where D means deterministic, B means Büchi, M means Muller and TA means tree automaton.

An example can show clearly that Büchi tree automata have less expressive power than Muller tree automaton.

Suppose T be the set of infinite trees on a alphabet $A=\{a,b\}$ and there's a finite number of the letter a along any path. First we can see that it is recognizable by a Muller tree automaton, or to say, Muller-recognizable. For transitions, we say that there are two states 1 and 2 and the edges like (p, x, q, q). When $x = a$, it results $q = 1$, otherwise $q = 2$. We suppose a quite specific family $\mathcal{F}$ here, which is just the singleton $\{2\}$.

So if we show that the set T can't be recognized by a Büchi automaton, we say that Büchi tree automaton are less powerful than Muller tree automaton. We use $\mathcal{A}$ to symbol a Büchi tree automaton which accepts every tree in T and n to symbol the amount of states of the automaton $\mathcal{A}$. For the proof, all individuals in the complement of T also need to be accepted by $\mathcal{A}$. For $i \geqslant 0$, consider the set $U_i = \bigcup_{k=0}^{i-1} (1^+0)^k$, which is shown as the figure below, and let the equation holds $t_i(x) = a$, where $t_i(x)$ is an infinite tree, for $x \in U_i$ and $t_i(x) = b$ otherwise.

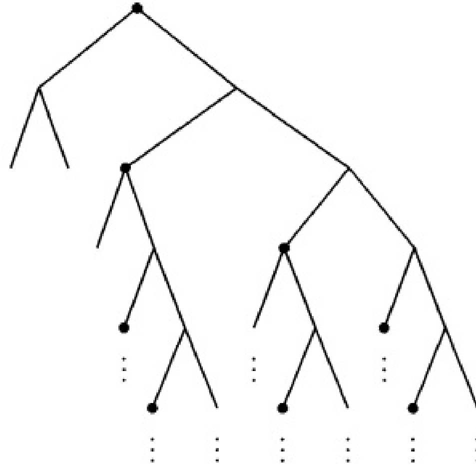

Figure 5. The set $U_i = \bigcup_{k=0}^{i-1} (1^+0)^k$

Here $t_n \in T$ holds. Hence, there exists a run r of $\mathcal{A}$ being successful. As below, We can use the methods of induction on the number of states to prove that there exists a path in t_n with three nodes $n < v < w$ such

that $r(u) = r(w) \in F$ and $t_n(v) = a$ in an n-state automaton accepting all trees of T.

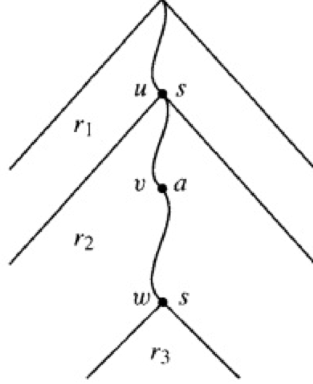

Figure 6. A path in t_n

When $n = 2$, it's definitely true in this case. Here $\mathcal{A} = (\{i, f\}, E, i, \{f\})$, so there are only two possibilities: the run for $r(\varepsilon) = i$ and the run for $r(x) = f$.

Next, let's consider when n is a bigger number. Assume that the property holds true when there are $(n - 1)$ states. According to the definition of U_i, it's clear to find a path passing infinitely often through F. The path 1^* satisfies the condition. Thus, there exists an $u \in 1^*$ so that $r(u) \in F$ holds. We see that the relation between t_n and t_{n-1} is like this: $(t_n)_{u0} = t_{n-1}$. So the run r_{u0}, which is also a tree, is of the automaton like $(Q, r(u0), F)$ on t_{n-1} and it is successful. If here we replace this run directly with $r(u)$, then the claim will have already been proved.

On the other hand, $r(u0)$ is still a run on the automaton with $n-1$ states. It is successful and accepts all trees of T. Then it continues to the loop following by induction. So we see that r_u is a part which can be pruned. By deleting r_u from r, we set it be r_1 and $r_1(u) = c$. And by deleting $r_{u'}$ from r_u, we get r_2 and we get r_3 as the rest part $r_{u'}$. Then we can get the overall $r = r_1 \cdot r_2 \cdot r_3$ and in a similar way for s_u, we set $t_n = s_1 \cdot s_2 \cdot s_3$. We find out that $s_1 \cdot s_2^\omega$ has an infinite number of symbols a and it also has a successful run, which is the tree $r_1 \cdot r_2^\omega$. Then the assertion is proved.

So it is obvious that the properties which Büchi tree automata have are also owned by Muller tree automata. It's the properties of closure under union, intersection and projection. They are all owned by Muller tree automata.

We can prove it easily. Suppose T and $T^{'}$ are recognizable and nondeterministic, then $T \cup T^{'}$ is obviously recognizable. And it's the same for closure under intersection. As for closure under projection, using two alphabets A and B, we consider a function $f\colon A \rightarrow B$ and their corresponding Muller antomaton $\mathcal{A}$ and $\mathcal{B}$. Say that $\mathcal{A}$ recognizes T, it's quite straightforward to see that $\mathcal{B}$ recognizes $f(T)$.

Besides these properties, the class of Muller recognizable sets of trees has an important property of closure under complementation. But its proof is the most difficult problem for nondeterministic automata. The proof will be given in the next section when we give an introduction on a particular game on trees.

But before introducing the games played on trees, let's introduce the Rabin chain tree automaton, which has an equivalent expressive power with the Muller tree automaton. But we need to introduce it because chain automata are of special significance in the games we'll introduce in the next section. The games we specify here are infinite two-person games. Generally speaking, the game strategies must be funtions of past states, but both sides have strategies are just based on the current states for chain automata.

A Rabin chain tree automaton is defined of the form $\mathcal{A} = (Q, q_0, E, \mathcal{S})$ where $\mathcal{S}$ is a strictly increasing sequence $E_1 \subset F_1 \subset E_2 \subset F_2 \subset \ldots \subset E_n \subset F_n$ of subsets of states of Q. Alternatively, the Rabin chain condition can be formulated by a parity condition. We say that a subset $X$of Q satisfies the parity condition $c\colon Q \rightarrow \mathbb{N}$ if and only if $\min \{c(q) | q \in X\}$ is odd.If there is an integer k for each path π such that:

$$\mathrm{Inf}(r|\pi) \cap E_k = \varnothing \text{ and } \mathrm{Inf}(r|\pi) \cap F_k \neq \varnothing$$

Then the run is successful.

Recall the accepting condition of Rabin automata mentioned above.

Formally, let the Muller tree automaton $\mathcal{A} = (Q, E, i, \mathcal{F})$ and its memory extension is the Rabin chain tree automaton $\mathcal{B} = (S, F, j, \mathcal{S})$. S is the set

$$S = \{(u, v) | uv \in \mathrm{Perm}(Q)\}$$

where Perm(Q) refers to the set of permutations of elements of Q. The action of permutating is like: if we have a set $\{a,\ b,\ c\}$, we'll have 6 permutations listed: $(a,b,c),(a,c,b),(b,a,c),(b,c,a),(c,a,b),(c,b,a)$.

The initial state j has a quite different definitions with traditional ones. It can be any element of S as long as it has the form xi. The set F here is the set of transitions, which is the set of all:

$$((u,v),a,(x,yq),(z,tr))$$

such that $uv=xqy=zrt$ and $(p,a,q,r)\in E$. The set F is defined as above since we need to track every running path in the last order of each state in the memory extension.

Actually, for the autamata on words, we have the similar definitions and results above. So for the similarity, with a proof, we can state similar propostions.

Proposition 6. *The automaton $\mathcal{A}$ and its memory extension $\mathcal{B}$ are equivalent. Consequently, any Muller tree automaton is equivalent to a Rabin chain tree automaton.*

Proposition 7. *Let $X\subset(\{0,1\}\times A)^{\omega}$ be a recognizable set of words on the alphabet $\{0,1\}\times A$. Then, the set $T(X)$ is Muller recognizable.*

If X is recognizable by a Büchi deterministic automaton, then T is Büchi recognizable.

Here, $T(X)$ is the set of all trees $t\in T_A^{\omega}$, where $(\pi,\ t|\pi)\in X$ for all $\pi\in\{0,1\}^{\omega}$.

2.2.3 Rabin basis theorem

Let's recall that for finite trees, we say a set T is recognizable if and only if it is rational. What's the circumstance when it extends to the set of infinite trees? We now introduce a very important theorem, named Rabin basis theorem, which is:

Theorem 8. *Any non-empty Muller recognizable set of infinite trees contains a rational tree.*

If a tree is finite generated in the following sense, it is called a regular tree: There is a deterministic finite automaton equipped with output which tells for any given input $\omega \in \{0,1\}^*$ which label is at node ω.

According to what we've introduced until now, we know that here non-empty Muller recognizable set of infinite trees can be replaced by any equivalent set, such as non-empty parity recognizable set.

Before we prove the theorem, we introduce a lemma which shows that the problem can be reduced to the one with "input-free" tree automata. It'll be helpful to prove the final theorem.

Lemma 9. *For any Muller tree automaton $\mathcal{A} = (Q, A, E, i, \mathcal{F})$, one can build an input-free tree automaton $\mathcal{A}' = (Q', E', i', \mathcal{F}')$ with a function f: $Q' \to A$ such that r' is a successful run of $\mathcal{A}'$ if and only if $r = f \circ r'$ is a successful run of $\mathcal{A}$.*

In particular, the set recognized by $\mathcal{A}$ contains a rational tree if and only if the set recognized by $\mathcal{A}'$ does.

We can build $Q' = Q \times A$, $E' = \{((q, a), (q, a'), (q'', a''))|\ (q, a, q', q'') \in E\}$, then the function f satisfies the property of projection $f(q, a) = a$.

With the lemma, we start to prove the Rabin basis theorem.

Proof. Let's recall that we say a set is Muller recognizable if it can be recognized by a Muller automaton. And we say that an infinite tree t is recognized by the Muller automaton $\mathcal{A}$ if there exists a successful run of $\mathcal{A}$ on t. So the problem of proving Rabin basis theorem can be converted into the problem of proving that the Muller automaton has a rational successful run of it.

Let's introduce a concept called "live". We call a state $q \in Q$ is "live" if $q \neq i$ and if there are transitions (q, a, q', q'') with $q' \neq q$ or $q'' \neq q$. We can see that it has exactly the literal meaning. A state is live if it doesn't always stay there and doesn't move.

And we've also introduced "input-free" tree automata above and get the lemma. So let's mix the concepts together and do the proof by means of induction.

First, if there are no live states in the automaton, all runs will be rational since all of the states from the root will be stock-still.

Next, suppose that some live states are not used in the successful run. We then ignore these states and only consider the automaton without these states.

Then, suppose that there is a node u in r such that the state $q = r(u)$ is live but that there is a live state q' that appears after the node u. In this situation we can construct an antomaton $\mathcal{A}_1$ by replacing all transitions from state q by the unique transition (q, q, q). So in the scope of $\mathcal{A}_1$, q isn't live anymore and according to the first point we just got, we know that $\mathcal{A}_1$ has a rational successful run r_1. In the other situation we construct an automaton $\mathcal{A}_2$ by choosing q as initial state and deleting q' from $\mathcal{A}$. And according to the induction hypothesis, $\mathcal{A}_2$ has a rational successful run r_2. Concatenating the successful runs of $\mathcal{A}_1$ and $\mathcal{A}_2$, we can see that $\mathcal{A}$ has a rational successful run $r_1 \cdot_q r_2$.

Last, suppose that all live states appear in r beyond any given node. Let's choose an arbitrary live state q. We can see that: 1) Each path in r passes by q. 2) There is a finite run s of $\mathcal{A}$ such that

i. s starts at q.

ii. s ends with q.

iii. each path in s passes through all live states.

It states that $\mathcal{A}$ has a rational successful run $r \cdot_q s^{\omega,q}$.

According to all statements above, the induction hypothesis is proved and the proof completes. □

With the theorem, we get a statement.

Corollary 10. *The emptiness problem for Muller tree automata is decidable.*

The emptiness problem is the problem of deciding whether the language recognized by a given automaton is empty.

We can also prove the theorem with the methods of gaming. We'll introduce it in the next section.

3 Games on tree automata

In this section, we'll introduce a game played on trees as a measure of our simulation of automata. We'll use the game to prove the property of closure under complementation we mentioned above. But before we focus the game on tree automata, we'll briefly introduce games on infinite words, as a mathematical technique, to eliminate the confusion, for which you may ask why the automaton is related to games.

Games here are abstract two-player games in which each player chooses a symbol from an alphabet in turn. So there will be a fixed set of plays which makes the first-moving player win. And correspondingly, the second-moving player wins on the complement of this set.

So, what will be the choices of players when they played on trees?

3.1 Automaton and pathfinder

Unlike games on words, every transition of a tree will lead to multiple nodes(we say 2 here). Then the moves of both sides won't seem to be so "symmetric", but there's never true fairness in a game. Here we call the first-moving player Automaton, who chooses transitions from the set E, which is the transition set of a Muller automaton $\mathcal{A} = (Q, q_0, E, \mathcal{F})$ on the alphabet A. We call the second-moving player Pathfinder, who chooses direction from the set $\{0, 1\}$. We call the game an infinite two-person game $G_{A,t}$. In particular, such a game played on graphs is called a parity game. The winning set of a parity game is usually defined by a Rabin chain condition or a parity condition. The parity game has several useful properties.

From the start, it means that Automaton first chooses a transition from the root and Pathfinder secondly chooses a path to follow and Automaton again chooses a transition from this node and so on. This will result in a chosen state sequence $(q_0, q_1, \ldots)$. The winning condition is that if this sequence satisfies the Muller acceptance condition defined by $\mathcal{F}$. If yes, Automaton wins. Otherwise, Pathfinder wins.

A simple graph example may show it clearly:

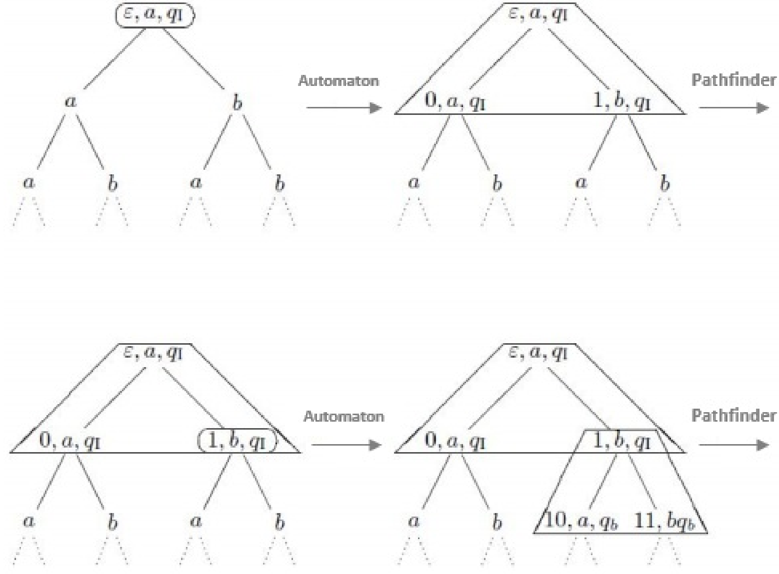

Figure 7. Play of the game

We can see that Automaton has winning strategy in $G_{\mathcal{A},t} \Leftrightarrow t \in L(\mathcal{A})$. Contrarily, Pathfinder has winning strategy in $G_{\mathcal{A},t} \Leftrightarrow t \notin L(\mathcal{A})$.

The game played here can be thought as a graph whose vetrices are the game positions. We can define the game positions formally: For the total set of vetrices, $V = V_1 \cup V_2$. V_1 stands for the postions of Automaton $V_1 = \{(\omega, q) | \omega \in \{0,1\}^*, q \in Q\}$ and V_2 stands for the positions of Pathfinder $V_2 = \{(\omega, e) | \omega \in \{0,1\}^*, e \in E\}$.

The game $G_{\mathcal{A},t}$ is defined as an abstract game on $Q \cup E$, so it's a rational game. And from a known theorem, we know that one of the two players has a rational winning strategy in a rational game. And again from the definition we call such a game is determined.

From the results above, we can get the following proposition first, which is the basis of our proof for closure under complementation later.

Proposition 11. *The automaton $\mathcal{A}$ accepts the tree t if and only if Automaton wins the game $G_{\mathcal{A},t}$. It does not accept t if and only if Pathfinder wins the game $G_{\mathcal{A},t}$.*

If $\mathcal{A}$ is moreover a Rabin chain tree automaton, the winning strategies can be chosen to be memoryless, which depends only on the last vertex of the path.

Proof. Let's recall the definition of a succesful run r of an automaton. $\mathcal{A}$ accepts t if and only if there is a successful run r of $\mathcal{A}$ on t, which means for every path π of r the set of states that occur infinitely often is in $\mathcal{F}$ for Muller automata. So let's consider from both directions.

If $\mathcal{A}$ accepts t, it means that there is a successful run. Hence the path choosed belongs to X, which is the set of winning paths. So Automaton wins the game.

From the other direction, if Automaton wins the game, he must have a winning strategy, which is a function $f: \{0,1\}^* \times Q \rightarrow E$. We could construct a run r based on a path of $\mathcal{A}$. We let a word $\omega \in \{0,1\}^*$ and $r(\varepsilon) = q_0$. Then let $r(\omega 0) = q'$, $r(\omega 1) = q''$ and the funtion $f(\omega, r(\omega)) = (q, t(\omega), q', q'')$. So r is a successful run of $\mathcal{A}$ on t and $\mathcal{A}$ accepts t.

And we know the game $G_{\mathcal{A},t}$ is determined, so the first statement proved.

For the second statement, we've mentioned that the game $G_{\mathcal{A},t}$ is called a parity game. By a known theorem, it's stated that one of the players has a memoryless winning strategy from each vertex in a parity game. Then the whole theorem is proved. □

This proposition also have the equivalent form like this:

Proposition 12. *Let $\mathcal{A} = (Q, q_0, \mathcal{F})$ be an input-free Muller automaton. Then Automaton wins the game $G(\mathcal{A})$ if and only if $\mathcal{A}$ has at least one successful run.*

3.2 Rabin's tree theorem

Now let's introduce the main theorem, named Rabin's tree theorem, for which we states out many preliminary points in this section.

Theorem 13. *The family of Muller recognizable sets of trees is closed under complement.*

Proof. According to Propostion 6, the class of Muller tree automaton has exactly the same expressive power with the class of Rabin chain tree automaton. So proving the theorem equals to proving that the complementation of the set of trees $t \in T_A^\omega$, under the condition of being accepted by $\mathcal{A}$, is still recognizable.

And by Proposition 11, we know that if the tree t is not accepted by a Rabin chain tree automaton $\mathcal{A}$, Pathfinder has a winning strategy which is memoryless.

Let the Rabin chain tree automaton forms like $\mathcal{A} = (Q, q_0, E, \mathcal{S})$ over an alphabet A. We can observe that Pathfinder's strategy is a funtion $f\colon \{0, 1\}^* \times E(\text{his game positions}) \rightarrow \{0, 1\}(\text{directions})$. We can decompose the funtion f into a family of local instructions $I\colon E \rightarrow \{0,1\}$. Then the Pathfinder's winning strategy can be coded by an I-labelled tree s: $\{0, 1\}^* \rightarrow I$. And we let the $A \times I$-labelled tree (s, t) with $(s,t)(\omega) = (s(\omega), t(\omega))$, where $\omega \in \{0,1\}^*$.

Let Z be the set of infinite words on $I \times E \times \{0, 1\}$ formed of all (x, y, π) such that

i. If the instruction sequence x is applied to the transmission sequence y, the result will be in the path π.

ii. The corresponding states' sequence isn't successful.

We can see that Z is therefore recognizable and according to Proposition 7, the set $Y = \{s \in T^{\omega}_{I\times E} \,|\, \text{for all}\, \pi \in \{0,1\}^{\omega}, (s(\pi), \pi) \in Z\}$ is also recognizable. And the set of trees X which are not accepted by $\mathcal{A}$ is projected from Y. For the property of closure under projection, owned by recognizable sets, X is also recognizable. The proof is concluded. □

3.3 Still Rabin's basis theorem

We've introduced Rabin's basis theorem in the previous section. Now we can prove it in another way through the methods of gaming. We state the theorem again: There exists a rational tree for any non-empty Muller recognizable infinite tree set.

According to proposition 6, the class of Muller tree automaton has an exactly the same expressive power with the class of Rabin chain tree automaton. So we just consider the "Rabin chain" recognizable set T of infinite trees, which is the same as considering Muller recognizable ones. The automaton is as the form $\mathcal{A} = (Q, q_0, E, \mathcal{S})$ on the alphabet A. We can construct an input-free automan from it: $\mathcal{A}' = (Q \times A, \{q_0 \times A\}, E', \mathcal{S}')$, who has some successful run r' if and only if $T' \neq \varnothing$.

We now consider the game $G(\mathcal{A}')$. Since $\mathcal{A}'$ has a Rabin chain condition, $G(\mathcal{A}')$ is a parity game. In a partiy game, there must be a memoryless winning strategy from each vertex for one of two players. So we can say that if $T \neq \varnothing$, the player Automaton has a memoryless winning strategy and it means that there's a successful run for him and thus T has a rational tree. The theorem is proved.

4 Topology

In this section, we introduce the infinite trees from a topological perspective. From the previous sections, it's concluded that for Büchi, Muller and those equivalent tree automata, the emptiness problem is decidable. Hence, we can decide whether a given recognized language is empty. After knowing that, we may want more to know how complicated a given language is. That's what the topology of infinite trees can tell us. And a theory, which studies how complex a set is, is called descriptive set theory.

4.1 The definition of topology of infinite trees

Since the topology theory is a very big issue, we just concentrate on the related concepts to our automaton theory on infinite trees. We assume that the readers have associative knowledge such as Borel hierarchy and Wadge hierarchy, but introducing some terminology we need is necessary.

4.1.1 Prelinimary knowledge

Countable sets A set is said to to be countable if there exists an injective map from E into $\mathbb{N}$.

General topology A topology on a set E is a set $\mathcal{T}$ of subsets of E satisfying the following conditions:

1) $\varnothing$ and E are in $\mathcal{T}$,

2) $\mathcal{T}$ is closed under arbitrary union: if $(X_i)_{i\in I}$ is a family of elements of $\mathcal{T}$, then $\bigcup_{i\in I} X_i \in \mathcal{T}$,

3) F is closed under finite intersection: if $(X_i)_{1\leqslant i\leqslant n}$ is a finite sequence of elements of $\mathcal{T}$ then $\bigcap_{1\leqslant i\leqslant n} X_i \in \mathcal{T}$.

The elements of $\mathcal{T}$ are called open sets and the complement of an open set is called a closed set.

Topological space A set E together with a topology on E.

Discrete topology Given two topology $\mathcal{T}_1$ and $\mathcal{T}_2$, we say that $\mathcal{T}_2$ is stonger than $\mathcal{T}_1$ if $\mathcal{T}_1 \subset \mathcal{T}_2$. And the discrete topology defined by $\mathcal{T} = \mathcal{P}(E)$ is the strongest topology.

Metric and metric spaces A metric d on a set E is a map: $E \to \mathbb{R}_+$ satisfying the following conditions, for every $x, y, z \in E$:

1) $d(x, y) = 0$ if and only if $x = y$,

2) $d(x, y) = d(x, y)$,

3) $d(x, z) \leqslant d(x, y) + d(y, z)$

A metic space is a set E together with a metric d on E. The topology defined by d is obtained by taking as a basis the open ε-balls defined for $x \in E$ and $\varepsilon > 0$ by

$$B(x, \varepsilon) = \{y \in E | d(x, y) < \varepsilon\}$$

Cantor space A Cantor space is a topological space with the essential features that it is non-empty, totally disconnected, compact, perfect and metrizable. The Cantor space is Cantor set itself, for example, the set 2^{ω}.

Polish spaces A Polish space is a completely metrizable space which admits a countable basis of open sets.

Borel sets The class of Borel sets is a minimal class of sets containing open sets, which has properties of closure under countable union and complementation. From the class of Borel sets, we can construct a hierarchy, named the Borel hieratchy.

4.1.2 The space of infinite trees

For a finite alphabet A, the set T_A^{ω} of infinite trees is defined as a topological space of infinite trees. There are several choices for us to define the topology of infinite trees here because the topology and tree structures have their own definition and we need to mix them up.

For example, we can define it from the aspect of tree's funtion, the intuitive aspect of topological distance or we can just define the topology as the whole set.

So we make a compromise here.

Any tree can be thought of as a function from D^* to A by appending additional elements to the label set. The functions' set from D^* to A is said to be the topological space of product topology, and A is considered to be a discrete topological space. The set of tree, $T(D, A)$ is a subset of and is closed under this space.

We have a concept of open topology. The set has a form like $S \cdot_c T(D, A)$, in which S is a set consisting of finite trees.

Now we can define the Borel hierarchy of subsets of topological space Σ^ω. It's defined as follows:

Σ_1^0 is denoted as the class of open sets of trees and Π_1^0 is denoted as the class of closed sets.

For a non-empty countable ordinal α, Σ_α^0 is denoted as the class of countable unions of subsets of Σ^ω in $\bigcup_{\gamma<\alpha} \Pi_\gamma^0$ and Π_α^0 is denoted as the class of countable intersections of subsets of Σ^ω in $\bigcap_{\gamma<\alpha} \Sigma_\gamma^0$.

4.2 Suslin sets

As we've introduced the topology of infinite trees, we can see that due to the complexity of tree structure, more complex topological classes are needed. In descriptive set theory, the Suslin sets are the sets of images of trees under certain projection.

Suslin was a Russian mathmatician. He made a major contribution to descriptive set theory and analytic set theory in his lifetime. Suslin called the projections of Borel sets analytic and that there are actually analytic sets which are not Borel measurable. He and his students established most of the basic properties of analytic sets. Due to his contribution, analytic sets are sometimes called Suslin sets. Based on these, beyond Borel hierarchy, another hierarchy named projective hierarchy was constructed.

The classes of Suslin sets are formed by continuous Borel sets, and the classes of Suslin sets contain strictly Borel sets. The classes of Suslin sets Σ_n^1, Π_n^1 and Δ_n^1, for integers $n \geqslant 1$, of the projective hierarchy are obtained from the Borel hierarchy by successive applications of operations of projection, complementation and Δ_n^1 is defined as

$$\Delta_n^1 = \Sigma_n^1 \cap \Pi_n^1.$$

The first level of the projective hierarchy is constructed by the class of Suslin sets Σ_1^1 and the class of co-Suslin sets Π_1^1 (the complements of Σ_1^1). We can see that the class of Suslin sets Σ_1^1 are obtained by projection of Borel sets. The class of Borel subsets of Σ^ω is then strictly included in Σ_1^1.

Regard to the link between the class of Borel and Suslin sets, the following proposition holds true.

Proposition 14. *Let $X \subset A^\omega$ be a Borel subset. Then the set*

$$T = \{t \in T_A^\omega \mid \text{for all}\, \pi \in \{0,1\}^*, t|\pi \in X\}$$

is in the class Π_1^1 of co-Suslin sets.

Proof. *We see that here we want to prove the set T is in the class Π_1^1 of co-Suslin sets. It means that we want to prove the set $\bar{T}$ is in the class Σ_1^1 of Suslin sets. And we call that a subset of T_A^ω is A-Suslin if there are a projection of closed subsets in $T_A^\omega \times A^\omega$ to it.*

So we can try to construct a set of pairs $(t, x) \in T^\omega \times A^\omega$ which allows the projection of closed subsets in this problem. Let the set $V = U \cap \bar{X}$. Here $\bar{X}$ is a closed set and can exclude the trees included in T and with the intersection operation, if we construct a U which is closed and is the set of the specific form of pairs. We'll prove the theorem.

Let $U = \{(t, x) \in T^\omega \times A^\omega \mid \text{there exists}\, \pi \in \{0,1\}^*, t|\pi = x\}$. U can be seperated as the intersection of many open sets. U_n is denoted as the set of pairs (t, x) satisfying the following property: there exists a word $a_1a_2...a_n \in \{0,1\}^*$ such that $t(\varepsilon)t(a_1)t(a_2)...t(a_n)$ is a prefix of x. Then $U = \bigcap_{n>1} U_n$. According to the definitions above, the sets U_n are in the class Σ_1^0 of open sets. Thus, the set U is in the class Π_2^0 of sets. Now we know that U is the desired set. The proposition is proved. □

4.3 Topological complexity of recognizable sets of trees

Topological complexity of the recognizable tree language is seen as evidence of its structural complexity, which also increases the computational complexity of verification problems associated with automata as non-empty problems. In fact, the topological aspect can be regarded as the embryonic form of infinite computational complexity theory.

Let's explore the topological complexity of Büchi recognizable sets and Muller recognizable sets.

Proposition 15. *Any Büchi recognizable set of trees is Suslin.*

Proof. *Use $\mathcal{A}$ as a Büchi tree automaton recognizing a set T. The set of pairs (r,t) where r is a successful run of $\mathcal{A}$ on the tree t is a Borel set, of which is the class Π_2. Thus T is Suslin.* □

Proposition 16. *A Muller recognizable set of trees belongs to the class Δ_2^1.*

Proof. *Use $\mathcal{A}$ be a Muller tree automaton recognizing a set T. By definition above, a tree t is in T if there is a successful run r of $\mathcal{A}$ on t. By propostion 14, it is enough to prove that the set of successful runs of $\mathcal{A}$ is in Π_1^1. In addition, since the map $(r,t) \mapsto t$ is continuous and Σ_{n+1}^1 is defined as the class formed by the continuous images of Π_n^1-sets, we know that the set T belongs to the class Σ_2^1.*

Since the class of recognizable sets is closed under complement, we also know that $T \in \Pi_2^1$. Hence, $T \in \Delta_2^1 = \Sigma_2^1 \cap \Pi_2^1$. □

Comparing these two statements, we have a result showing that the class of Muller recognizable sets of trees has a higher topological complexity than that the class of Büchi recognizable sets has. On the other hand, we must observe that since Muller tree automaton has the most expressive power as we've already presented, we may not recognize the sets of topological complexity higher than Δ_2^1.

4.4 Wadge hierarchy of tree languages

At last, a brief mention to Wadge hierarchy seems to be necessary.

In general, the Wadge hierarchy is thought as the refinement of the Borel and projective hierarchy. It can give the deepest insight into the topological complexity of languages. Although the hierarchy is already determined for the circumstance of ω-regular languages, higher topological complexity of tree languages causes the adaption of Wadge hierarchy on regular tree languages more difficult.

As for the latest progress, the problem of determining the Wadge hierarchy of tree languages accepted by deterministic Muller or Rabin tree automata have already been solved by Murlak. But the case of non-deterministic Muller tree or Rabin tree automata, on which we mainly focus through the introduction, is still unsolved. Besides it, actually many problems remain open about the regular languages of infinite trees due to their complexity.

We mentioned that all kinds of automata we introduced here are nondeterministic, but the case of them are still not determined. So it's enough to stop here.

5 Monadic second order logic of two successors

Automaton theory is so useful because the automaton is a very good tool for reducing some complex problems to problems for corresponding automaton. We mentioned that there's a close connection between logics and automata, especially tree automata. So tree automata are useful to solve the important problems of logics in history. Here we introduce the expressive power of them and the reduction.

5.1 Introduction to monadic second-order logic

For the logical theory, we have basic logical symbols: the logical connectives $\wedge$(and), $\vee$(or), $\neg$(not), $\rightarrow$(implication), $\leftrightarrow$(biconditional), the quatifiers $\exists$(there exists), $\forall$(for all), the equality $=$, an infinite set of variables and parenthesis and other punctuation symbols(for unique readability).

Other than these logical symbols, we have a set $\mathcal{L}$ of nonlogical symbols: relation symbols, also called predicate symbol,(such as $<, >$), funtion symbols and constant symbols(such as $0, 1$).

With these symbols, first-order logic has its own formation rules, defining the terms and formulas. Terms can be any variable or a funtion of n arguments. Formulas are expressions of finite applications of the following rules:

i. If R is a relation symbol, $R(t_1, ..., t_n)$ is a formula, where t_i is a term.

ii. $t_1 = t_2$ is a formula, where t_1 and t_2 are terms.

iii. If blc is a binary logical connective, $\varphi_1 blc \varphi_2$ is a formula, where φ_1 and φ_2 are formulas.

iv. $\neg\varphi$ is a formula, where φ is a formula.

v. $\forall x\varphi$ and $\exists x\varphi$ are formulas, where x is a variable and φ is a formula.

First-order logic covers relation and quantification. It is the standard for the formalization of mathmatics into axioms and is studied in the foundations of mathmatics. It's very important but it is unable to do something.

First-order logic can quantify all individuals of its domain, but it is unable to quantify the relations of its individuals. Besides it, first-order logic has no strength to uniquely describe a structure with an infinite domain. That's why we need a stronger logic as second-order logic.

Second-order logic extends first-order logic by allowing quantification over relations, which are called second-order variables, of individuals in its domain. But whole second-order logic has so wide a range for discussion. So we limit ourselves on monadic second-order logic, which exactly satisfies our requirement. It's expressive enough and simple. With this, many properties will be shown in a quite natural way.

Monadic second-order logic(MSO) is a fragment of second-order logic whose second-order relations is limited to monadic relations(relations only have exactly one argument). Since monadic relations have same expressive as sets, we can say that MSO quantifies over sets. MSO plays an important role in automaton theory and also in computer science.

The fundamental relationship between MSO and automata was discovered independently by Büchi, Elgot and Trakhtenbrot when the logic was proved to be decidable over the class of finite linear orders and over $(\omega, <)$, called weaker MSO(WMSO). It's also proved that the formula can be transformed into automata and vice versa. Moving away from linear orders, McNaughton generalized the transformation that there is a effective transformation from monadic second-order logic of tree structures into finite automata on trees. Based on these connection, Rabin proved that the monadic second-order of the full binary tree(also called the the second-order monadic theory of two successors), denoted by S2S, is decidable. This celebrated theorem, obtained using the notion of tree automata, is often referred to as "the mother of all decidability results". Since then, many problems in the field of formal verification of programs can be reduced to the logics.

However, Rabin's proof seems too difficult for many to understand. Hence, a continual refinement of proof is made by scholars. Here the heart of our introduction is the nondeterminism of Muller tree automaton.

5.2 S2S

We denote the second-order monadic theory of two successors by S2S.

For the definition of syntax, we let symbols x, y, ... be its individual variables and symbols X, Y, ... be its set variables(relation symbols). There is another composition of syntax, called terms. They are composed of the constant ε and the individual variables applying two symbols s_0 and s_1, which are unary functions.

For the definition of semantics, the formulas on trees are interpreted by considering a tree as a model

$$(\mathrm{Dom}(t), \varepsilon, s_0, s_1, <, (\mathbf{a})_{a\in A})$$

where s_0 and s_1 are two successor functions from $\mathrm{Dom}(t)$ to $\mathrm{Dom}^+(t)$. They are defined by equations $s_0(x) = x0$ and $s_1(x) = x1$. The order $<$ is chosen as a prefix relation on $\{0,1\}^*$. The unary relation $\mathbf{a}$ is the set of $x \in \mathrm{Dom}(t)$ such that $t(x) = a$. Here the order $<$ and the constant ε are unfixed and they should be defined in terms of s_0 and s_1.

If φ is an $S2S$-formula, we say that a tree t satisfies the formula φ if φ is true of t.

The set of trees satisfying the formula φ is denoted by $\mathcal{M}(\varphi)$. We also set

$$\mathcal{M}^*(\varphi) = \mathcal{M}(\varphi) \cap T_A\,, \mathcal{M}^\omega(\varphi) = \mathcal{M}(\varphi) \cap T_A^\omega$$

A set T of finite (infinite) trees is definable in $S2S$ if there is a formula $\varphi \in S2S$ such that $T = \mathcal{M}^*(\varphi)$ $(T = \mathcal{M}^\omega(\varphi))$.

An example may be helpful for understanding the logical formula φ.

Example 17.

i. $x \leqslant y$ (node x is a prefix of node y): $\forall X.((y \in X \wedge \forall z(z0 \in X \Rightarrow z \in X) \wedge \forall z.(z1 \in X \Rightarrow z \in X)) \Rightarrow x \in X)$

ii. $\mathrm{Chain}(X)$ (X is linearly order by $\leqslant$): $\forall x.\forall y.((x \in X \wedge y \in X) \Rightarrow ((x \leqslant y) \vee (y \leqslant x)))$

iii. Path(X) (X is a path): Chain(X) $\wedge \neg\exists Y((X\subseteq Y)\wedge(X\neq Y)\wedge$ Chain$(Y))$, where $X\subseteq Y$ is equivalent to $\forall z.(z\in X\Rightarrow z\in Y)$ and $X=Y$ is equivalent to $X\subseteq Y\wedge Y\subseteq X$.

iv. Inf(X) (X is infinite): $\exists Y.(Y\neq\varnothing\wedge\forall\in Y.\exists y'\in Y.\exists x'\in X.(y<y'\wedge y<x'))$

Theorem 18. *A set of infinite trees is definable in S2S if and only if it is Muller recognizable. More precisely, for every formula* $\varphi(X_1, X_2, \ldots, X_n)\in S2S$*, one can construct effectively a Muller tree automaton* $\mathcal{A}$ *which accepts* t *if and only if* t *satisfies* φ*.*

Proof. The proof of this theorem contains two parts: conversion from infinite trees to formulas and from formulas to infinite trees.

Given a Muller tree automaton, we can build an equivalent $S2S$-formula:

We use $\bar{C}=(C_q)_{q\in E}$ to encode the run of tree.

- $\varphi\Leftrightarrow\exists\bar{C}.(\text{Part}\wedge\text{Init}\wedge\text{Trans}\wedge\text{Accept})$
- $\text{Part}\Leftrightarrow\forall x.\bigvee_{q\in Q}\text{State}_q(x)$
- $\text{State}_q(x)\Leftrightarrow C_q(x)\wedge\bigwedge_{q'\in S\setminus\{q\}}\neg C_{q'}(x)$
- $\text{Init}\Leftrightarrow\text{State}_{q_0}(\varepsilon)$
- $\text{Trans}\Leftrightarrow\forall x.\bigvee_{(q,A,q_0',q_1')\in M}\big(\text{State}_q(x)\wedge\big(\bigwedge_{V\in A}V(x)\wedge\bigwedge_{V\notin A}\neg V(x)\big)\wedge\text{State}_{q_0'}(x0)\wedge\text{State}_{q_1'}(x1)\big)$
- $\text{InfOcc}_q(P)\Leftrightarrow\exists Q.(Q\subseteq P\wedge Q\subseteq R_q\wedge\text{Inf}(Q))$
- $\text{Inf}(P)\Leftrightarrow\exists P'.(P'\neq\varnothing\wedge\forall x'\in P'.\exists y\in P.\exists y'\in P'.(x'<y\wedge y<y'))$
- $\text{Muller}(P)\Leftrightarrow\bigvee_{F\in\mathcal{F}}\big(\bigwedge_{q\in F}\text{InfOcc}_q(P)\wedge\bigwedge_{q\notin F}\neg\text{InfOcc}_q(P)\big)$
- $\text{Accept}\Leftrightarrow\forall P.(\text{Path}(P)\Rightarrow\text{Muller}(P))$

Thus we finish the encoding and get a formula from Muller tree automaton.

From the other direction, given an $S2S$-formula, we can construct an equivalent Muller tree automaton.

First, we rewrite the S2S formula as a general form. We have only the following type of equation:

$$x = \varepsilon, x = y0, x = y1, x \in Y, x = y$$

Then we can inductively transform S2S formulas into Muller tree automata $\mathcal{A} = (Q, E, I, \mathcal{F})$ on the alphabet A:

i. $x \in Y$:

- $Q = \{q_0, q_1\}$
- $I = \{q_0\}$
- $E = \{(q_0, A, q_0, q_1) | x \notin A\} \cup \{(q_0, A, q_1, q_0) | x \notin A\} \cup \{(q_0, A, q_1, q_1) | x \in A, Y \in A\} \cup \{(q_1, A, q_1, q_1) | x \notin A\}$
- $\mathcal{F} = \{q_1\}$

ii. $x = y0$:

- $Q = \{q_0, q_1, q_2\}$
- $I = \{q_0\}$
- $E = \{(q_0, A, q_0, q_2) | \{x, y\} \cap A = \varnothing\} \cup \{(q_0, A, q_2, q_0) | \{x, y\} \cap A = \varnothing\} \cup \{(q_0, A, q_1, q_2) | x \notin A, y \in A\} \cup \{(q_1, A, q_2, q_2) | x \in A, y \notin A\} \cup \{(q_2, A, q_1, q_2) | x \notin A, y \in A\} \cup \{(q_2, A, q_2, q_2) | \{x, y\} \cap A = \varnothing\}$
- $\mathcal{F} = \{q_2\}$

iii. And others so on.

Thus we build the formula from a Muller tree automaton.

The theorem is proved. ☐

Consequently, the connection between automaton and logic make the proof of next theorem of the decidability of satisfiability for $S2S$, called Rabin Tree Theorem.

Theorem 19. ***(Rabin's Tree Theorem)*** *The theory S2S is decidable.*

We mentioned that WMSO is a weaker MSO which is over finite orders. Here WS2S is a weaker S2S, in which the set variables range only over finite sets of positions. And we need to keep in mind that not every S2S formula can be translated in WS2S, unlike the property of S1S.

Theorem 20. *A set of infinite trees is Büchi recognizable if and only if it can be defined by a formula*

$$\exists X_1 ... \exists X_n \, \varphi(X_1, ..., X_n)$$

where $\varphi(X_1, ...X_n)$ *is a WS2S-formula.*

A set T of infinite trees is definable in WS2S if and only if T and its complement T_A^ω *are Büchi recognizable.*

Similarly, we can get a consequent theorem showing the decidability of satisfiability for WS2S.

Theorem 21. *The theory WS2S is decidable.*

We can get a corollary showing the connection between S2S and WS2S.

Corollary 22. *WS2S is strictly weaker than S2S.*

Up to now, we have seen the decidability of S2S and we've got a positive result. What about monadic second-order theory of more than 2 successors? Are they also decidable? The answers are always yes.

Theorem 23. *The theory SnS is decidable.*

6 Practical uses of tree automata

Basically, automata are used to specify and verify properties of structures. They have various applications in software verification, model checking, language processing, complexity theory and decidability theory. As we've seen the importance of tree automata in complexity theory and decidability theory, let's explore the practical uses of them.

The realistic application of tree automata on other fields has two preconditions which we've proved:

1. The regular tree languages have strong closure properties.
2. The emptiness problem is decidable for tree automata.

On the basis of that, the family of tree automata becomes a more and more interesting topic because of their strong expressive power. There are many very complex problems in various field in reality and especially with recent explosing increasement of information amount through the Internet, the need for manipulating them more effectively becomes urgent. So we need such an expressive power for helping us solving problems. Actually, tree automata have been designed for a long time since the last century.

Let's see the various fields that tree automata can apply on.

6.1 Applications on XML

The most intuitive application of tree automata is that on processing of XML since XML documents can be naturally viewed as trees.

The Extensible Markup Language (XML) nowadoys become the mostly used format for structured or semi-structured documents. Based on SGML (ISO 8879), XML is designed for meeting the challenges and increasing needs of largescale electronic publications and data exchange on the Web.

The XML data is the semi-structured data, of which queries and views are based on regular path expressions. In general, XML applications in tree automata have the following applications as:

i. a basic for pattern languages and pattern test

ii. a method of processing queries and schema languages

iii. an algorithm toolbox

where tree automata are quite useful for abstraction.

Let's introduce each of them.

6.1.1 A basic for pattern languages and pattern test

One reason of the popularity of XML is that there are many schema languages, including DTD, XML schema, DSD and RELAX, which can be used to define "types" (or "patterns") that describe the constraints of data structures and to improve security of data processing and exchange.

Nevertheless, the main use of processing technology of XML is usually limited to checking data in files, but not programs. In a common way, the XML handler reads the XML files first, then makes a check with the validating parser to match it with the given type. The program then uses a general tree operation Library (such as DOM) or a dedicated XML language, such as XSLT or XML-QL. Because these tools do not establish a system connection between the program and the document type it runs, they do not provide the documents generated by the compile time guarantee program to always conform to the expected type.

A regular expression type is proposed as the basis for static-typed XML files. Regular expression types capture and summarize the regular expression symbols, such as $(*, ?, |)$ etc., of common schema languages in XML, and support the natural semantic concepts of subtypes.

Example 24. Regard to types of regular expressions, the following example will be intuitive:

```
type Addrbook = addrbook[Person*]
type Person = person[Name,Email*,Tel?]
type Name = name[String]
type Email = email[String]
type Tel = tel[String]
```

corresponding to the following set of DTD declarations:

```
<!ELEMENT addrbook person*>
<!ELEMENT person (name,email*,tel?)>
<!ELEMENT name #PCDATA>
<!ELEMENT email #PCDATA>
<!ELEMENT tel #PCDATA>
```

The type constructor of the form label [...] uses the tag label (that is, the XML structure of the form) to classify tree nodes. Therefore, names, e-mail and phone types are all strings with appropriate identification tags. Type may also involve regular expression operators * (none or repetition) and ? (optional event) and | (alternation). Therefore, the type Addrbook describes a tag AddrBook with zero or multiple duplicate Person type subtrees. Similarly, the type Person describes a person label, whose contents are name subtree, zero or more Email subtrees and an optional Tel subtree.

After filling in the type Addrbook, the XML document will be like following:

```
<addrbook>
  <person><name>Micheal Huang</name>
          <email>micheal@usapr</email>
          <email>micheal@gmail</email>
  <person><name>Jemein Cat</name>
          <email>jemein@usapr</email>
          <tel>987-654-321</tel></person>
</addrbook>
```

Subtypes are defined in regular expressions in semantic expressions. Typically, one type represents a set of documents; one subtype is just contained between sets represented by two types. For example, think of again the definition of type Person:

```
type Person = person[Name,Email*,Tel?]
```

and the following variant:

```
type Person2 = person[(Name | Email | Tel)*]
```

The elements of type Person can have a name, zero or more e-mail and zero or one Tels in order, and the type Person2 allows any number of such nodes in any order. Therefore, Person2 strictly describes more documents, which means that Person subtypes from Person2. This subtype may be very useful in programming. For example, suppose we initially had a type value of Person. The inclusion above will allow us to process this value by using code that does not care about sortings between e-mail and telephone nodes. (for example, if we want to display subnodes in a linear format, this may happen, and we naturally use the branch of each label to write a single loop on the subnode sequence.)

Basically, regular expression types correspond to tree automata in an exact way, a finite state machine model for accepting trees. It is easy to construct a tree automaton from a given type. Tree automata only accept a set of trees represented by that type. Therefore, the subtype problem can be reduced to the inclusion problem between known decidable tree automata. Unfortunately, in the worst case, the time complexity of solving the problem is in exponential.

In order to deal with this high complexity, several efficient algorithms have been developed that they are expected to encounter by many when it's needed to programme with regular expression types. Instead of testing the tree automata included in the classic algorithm, which worked before transforming their input automata into other fully independent automata to test their properties (expensive), Aiken and Murphy's algorithm may be a good choice as a starting point. Like the standard subtype algorithms of other types of systems, the algorithm operates up and down, that is, for a given pair of types, the match of the topmost type constructor are checked, continuing to the subcomponent of the type and repeating the same check recursively . The main advantage of this top-down algorithm is that it can implement many simple optimizations. In particular, the reflexivity `T<:T` can be used to determine subtype relationships by looking only at a portion of the entire input type expression.

6.1.2 A method of processing queries and schema languages

XPath is a simple language for browsing the XML trees and returning a set of answer nodes. XPath expressions are ubiquitous in XML applications. They are used for XQuery binding variables, defining

keys in the XML mode, referencing elements in the external elements in XLink and XPointer, as mathematical expressions in XSLT and content based packet routing. Examples of containment problems for XPath expressions appear in every application and other applications. For example, the reasoning of the keyword described by the XPath expression can be reduced to include and the algorithm contained can be used to optimize the XPath expression.

The containment problem deals with simple XPath fragments, that consist of:

- node tests
- child axes (/)
- Descendant axes(//)
- Wildcards (*)
- Predicates ([...])

And this class of queries is called XP{[] , * , //}.

The high complexity of containment creates a new challenge: find practical algorithms for checking containment. Two goals may be targeted: (i) to find an effecient, sound algorithm, and show that it is complete in partiular cases of practical interest; and (ii) to find a sound and complete algorithm and show that it is effecient in partiuclar cases of practical interest.

Up to now, we may have introduced several basic types of tree automata. But actually many researches try to refine the basic ones and let them apply in various fields. Here an extended type, called the alternating tree automata, for the problem runs in exponential time in general, but runs in PTIME in some special cases of pratial interest, including when the number of //'s in p is restricted.

Every expression in $XP^{\{[],*,//\}}$ can be translated into a tree pattern of arity one with the same semantics, and, conversely, each pattern of arity one can be translated into an $XP^{\{[],*,//\}}$ expression. Figure 2(b) illustrates a pattern in $P^{\{[],*,//\}}$, with arity one, which is equivalent to the $XP^{\{[],*,//\}}$ expression $a[a]//*[b]//c$. The containment problems for

$\mathrm{XP}^{\{[],*,//\}}$ and $P^{\{[],*,//\}}$ are thus equivalent. While not present in XPath, patterns of higher arity are of great interest to us beause they capture multiple variable bindings, which occur for example in the FOR clauses of XQuery.

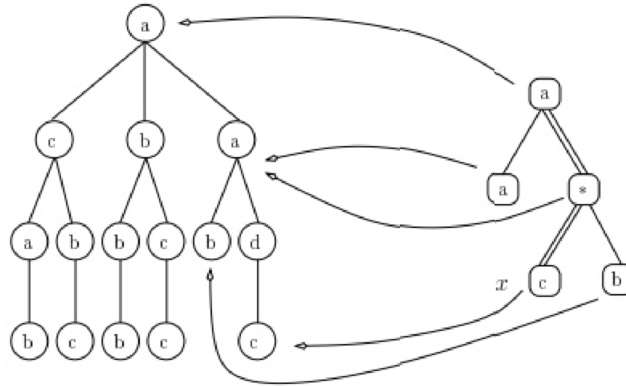

Figure 8. pattern *p* and an embedding from *p* to *t*

Many XML appliations could take benefits from a practical decision procedure for containment of such expressions. The tree automata can be quite useful to show that this fragment of XPath has an uncontrollable containment problem in general.

6.1.3 An algorithm toolbox

How to effectively evaluate a large number of XPath sets in an XML stream is a fundamental problem in data flow applications.

YFilter builds a XPath based NFA automata. All automata are combined into a automaton. If the automaton reaches a specific data driven termination state, the data stream satisfies the query corresponding to the termination state. YFilter can reduce different query processing. Repeated computation in X effectively handles XPath that does not contain {[]}, but it needs to save intermediate state computation to handle XPath queries that support {[]}.

Xtrie is an extension of the query processing based on NFA automata. When each element event is accepted, the NFA automatically finds the related transformation, but XTrie selects the related processor

after receiving the sequence of element events in response. An XPath processor that responds to a series of elements is much less than a XPath processor that responds to a single element. In this way, Xtrie reduces the number of query processors that may accept the possible response to the element input sequence, thereby improving the runtime processing efficiency.

The above methods are essentially the ones of the nondeterministic processors. That is to say, in the running process, there are many active states inside the processors, and the execution efficiency decreases with the increase of XPath. The existing query processing for another type of XML data stream is based on the determination of the automaton to be constructed.

Dan Suciu first constructs a XPath based NFA automaton, and then determines on the NFA automata. Therefore, the system will always have only one running state at any operation time. The processing efficiency of the system is independent of the number of XPath processings. However, relative to the number of queries at index level, this method may lead to extended spatial complexity. Additionally, the entire query processor has limited expressive power and does not support the {[]} operator in XPath.

XPush solves the problem of expressiveness of NFA automata. This extension mainly utilizes AFA automata that support the AND/OR relationship between expression paths. The AFA automaton uses extended states to save the execution of different paths. XPush actuators It is also based on deterministic automata, which improves the query efficiency of the system, but it also faces the problem of determining the space cost at the exponent level.

A machine named XEBT based on tree automata is proposed to solve this problem. Unlike traditional methods, XEBT has the following features: first, it is based on a tree automata with strong expressive ability, which can support Xpath {[]} without requiring additional states or intermediate results. Secondly, XEBT supports many optimization strategies, including the automatic establishment of the XPath tree based on DTD, and some limited additional space costs, determining the concurrency status that reduces runtime and combining the bottom-up and top-down assessments. Experimental results show that XEBT supports complex Xpath, which is superior to previous work in terms of efficiency and spatial cost.

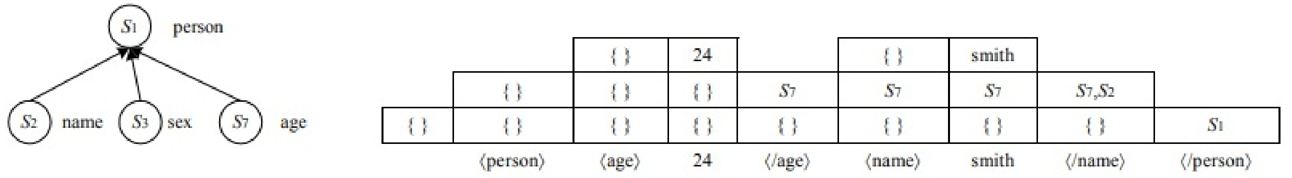

Figure 9. The tree automaton and the running stack of it on XML data stream

6.2 Concurrent program verification

Temporal logic is a modal logic suitable for describing time series events, and has been used as a powerful tool for specifying and verifying concurrent programs. One of the most important developments in this field is the discovery of algorithms for validating the temporal logic characteristics of finite state programs. This derives its importance from the fact that many synchronous and communication protocols can be modeled as finite state programs. A finite state program can be modeled by a transformation system in which each state has a bounded description, so that a fixed number of Boolean atomic propositions can be expressed. This means that finite state programs can be regarded as Kripke structures of finite propositions, whose properties can be specified by propositional temporal logic. Therefore, in order to verify the correctness of the expected behavior of the program, only the program modeled as a finite Kripke structure is a model of the propositional logic formula for the specified behavior. Therefore, the name model checks the validation method derived from this viewpoint.

There are two types of temporal logic: linear and branching. In linear temporal logic, every moment has the only possible future, and in the branch of temporal logic, each moment can be divided into several possible future. For these two temporal logic, we have developed a close and effective connection with the automaton theory on the infinite structure. The basic idea is to associate each temporal logic formula with a finite automaton, which accepts all the formulas of the formula. For linear temporal logic, the structure is infinite word, while for branch temporal logic, the structure is infinite tree. This makes it possible to reduce the time logic decision problem (for example, satisfiability) of the known automata theory (for example, non emptiness), thus producing a clean and asymptotically optimal algorithm.

Initially, non deterministic automata are used in document translation from temporal logic formulas to automata. These translations have two shortcomings. First of all, translation itself is not trivial; in fact, translation requires a series of special intermediate expressions to simplify translation. Secondly, for linear and branching temporal logic, there is exponential explosion between formulas and automata. This indicates that any algorithm that uses these translations as a step will become an algorithm with complexity of exponential time. Therefore, the automaton theory does not seem to be applicable to branch time model checking. In many cases, it can be done in linear runtime.

Recently many researches have shown some extended forms of nondeterministic autamata have better performance on NLP, such as alternating tree automata mentioned above, weighted tree automata and so on.

6.3 Applications on NLP

Linguistics and automaton theory are closely integrated. Long ago, Markov predicted the vowels and consonant sequences in Pushkin's novels with finite state process. Shannon expanded the idea and used Markov process to predict the alphabetic order of English words. Although many theorems of the finite state acceptor (FSA) and the finite state transducer (FST) were proved in 1950s, Chomsky thought the device was too simple to fully describe the natural language. Chomsky uses context free grammar (CFG), and then introduces a more powerful translation grammar (TG). When trying to formalize TG, automata theorists like Rounds and Thatcher introduced the theory of tree transducers. Computational linguistics has also begun to study carefully. Woods uses augmented transformation network (ATN) to perform automatic natural language analysis.

In the final paragraph of his 1973 tree automata survey, Thatcher wrote:

"The number one priority in the area [of tree automata] is a careful assessment of the significant problems concerning natural language and programming language semantics and translation. If such problems can be found and formulated, I am convinced that the approach informally surveyed here can provide a unifying framework within which to study them."

At this point, however, the mainstream work of automata theory, linguistics and computational linguistics can be divided into two parts. Automata theorists pursue some theoretical driven generalizations, while linguists take another way to avoid formalism. Computational linguistics focuses on the extension of CFG, many of which are Turing-equivalent. In 1970s, speech recognition researchers reused the FSA to capture the natural language syntax, this time using the conversion weight that could be trained on a machine readable text corpus. These formal devices have relevant algorithms that are effective enough for the actual computer at the time, and they are very successful in distinguishing right voice transcriptional and incorrect speech transcription. In 1990s, the combination of the finite state string and the large-scale training corpus became the main paradigm for speech and text processing; the universal software toolkit for weighted finite state acceptor and transducer (WFSA and WFST) was developed to support a wide and various applications.

In the new century, computational linguists rearoused the interest in tree automata, especially for automatic language translation, in which the conversion was very sensitive to the grammatical structure. The generic tree automaton toolkit has also been developed to support the survey.

6.4 Synthesizing reactive programs

The synthesis of reaction systems is a typical problem in computer science. It comes from a problem raised by Church in 1957, which occurs when synthesizing a standard integrated digital circuit written in arithmetic limited logic. This problem was first solved by Büchi and Landweber in 1969.

In 70s, Rabin saw an elegant automaton theory in the infinite tree. This theory has now been fully studied and developed into a beautiful theory, which has become a reasonable result of automata theory.

With the connection between time logic automata and word, tree automaton theory gives the most elegant solution to the church problem: using the singular tree automata of this structure, the specification is compiled into an infinite word automaton system on the deterministic parity automata for the input generation output strategy so that the

tree is in the tree. All paths are accepted by standard automata, and finally check the emptiness of tree automata.

In recent years, comprehensive problems have attracted wide attention from theoretical and practical circles. The theoretical approach includes the extension of the branch time specification, the very important issues in the synthesis of distributed systems, and the synthesis of incomplete information when the environment and systems may not have perfect information about each other's state.

In all these problems, the synthesis focuses on the design conversion system. In other words, the final output conversion system meets the specification of the synthesis algorithm. Although the transformation system is beneficial to define the semantics of the system, the system is seldom designed by describing the transformation system. The design of the system is advanced and concise.

For any traditional specification and a set of Boolean variables, B, we can build a tree automata to accurately accept the tree coding classes that meet all programs that are canonical in B. By checking the emptiness of tree automata, we can synthesize programs that conform to specifications, especially those that meet the specifications.

Let us fix input and output arities $N_I, N_O \in N$ for the rest of this section. Suppose that $\mathcal{A}_{\text{spec}}$ is a non-deterministic Büchi automaton. It accepts $(\{0,1\}^{N_I,N_O})^\omega$ which doer not conform to the specification of the sequence set. We also fix the set of Boolean variables B.

Then consider collecting all the trees corresponding to the program on the variable B, where input and output elements N_I and N_O. We can know that this is a set of rule trees. We assume that $\mathcal{A}_{pgm}$ is a tree automata, which only accepts these trees.

We will now construct a bidirectional alternating Büchi automaton that accepts trees corresponding to programs that do not conform to the specification. Intuitively, the automaton A will guess the specific operation of the program by non-deterministically selecting the input sequence, simulating the programs on these inputs and checking whether there is an operation of the specification automaton $\mathcal{A}_{\text{spec}}$ that is being executed. If there is such an input sequence, the program will generate an appropriate response sequence and be accepted by Aspec, and the program tree will be accepted.

A bidirectional alternative of tree automaton will have two tuples as states. The first one is a tuple of the form $\langle s, q, i, m, t\rangle$ where s is the current state of the program simulation (ie, an estimate of variable B) and q is the current state of the canonical automata $\mathcal{A}_{\text{spec}}$ while observing the input and Output sequence simulation, i is the last input received by the program, m is the mode for $m \in \{\text{in}, \text{out}\}$, the next I/O operation that the logger must perform is input or output. Whenever the specification state is updated, the last bit $t \in \{0, 1\}$ will be toggled to 1 and then set to 0 again.

The second state is the form of $\langle s, v\rangle$, where s is the current state of the program simulation, and the v is a boolean value; these states are used to encode boolean expressions (in if statements and while statements and on the right side of the condition), and to check whether the expression calculates the value v in the pre-state of the program.

Intuitively, automata traverses the program tree, explaining each statement and calculating the current state of variables in s. In this process, it may have to move up and down on the tree, because the while loop needs to traverse the same statement block many times. When it meets the output statement, it updates the specification automaton state and outputs the evaluation on the i. When it meets the input statement, it stores it in the corresponding variables in s and updates the component i.

6.5 Automating data completion

Many application domains use table rows and columns to store data. For instance, R data-boxes, relational databases and Excel spreadsheets treat all the fundamental data as two-dimensioned tables of cells. For such case, it is common to use some of the values stored in some cells to populate the values of other cells. As examples, think of the following ordinary data to accomplish this task:

- **Data imputation**: In statistical meaning, imputation means replacing lost data with substitute values. Because missing values can hinder data analysis tasks, users often need to fill in the table with missing values for other related items. For example, data imputation often occurs in such as R and other statistical computing frameworks.

- **Spreadsheet calculation**: In real life, there are many applications using the formats like spreadsheets. Then users need to calculate cells' values based on the other cells' values. For example, a general target is to calculate the values in newly created columns where these values will be dependent on the values in the previous columns.

- **Virtual columns in databases**: In many cases of relational databases, users need to create views where they are for storing the results of some intermediate database queries. In such cases, we usually need to add a virtual column whose values are dependent on the existing entries' values in the view.

2	Id	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Delta	
3	A			10	13	12	15	??	← 5
4	B	19	12	16	18			??	← -1
5	C				7	10	12	??	← 5
6	D			10	9	14		??	← 4

Figure 10. A data completion task in StackOverFlow

Suppose we give here a domain specific language(DSL) formatted by the following syntax $G = (T, N, P, S)$, where T stands for a finite set of terminals (i.e. constants and variables), N stands for a non terminal symbol, P stands for a set of forms to $F(e_1, \ldots, e_n)$, in which the built-in function ("component") is a built-in function ("component"). For simplifying the representation, we assume that in each production the used components are of first order; if they are of higher order, let's combine the methods we propose with the enumerated search.

Let's show here the working principle of the universal version of the space-learning algorithm. For each input and output example $\sigma \to o$, where σ is an estimated value and o is the output value, we construct a finite tree automaton(FTA) $\mathcal{A} = (Q, F, Q_f, \Delta)$ that exactly represents the assembly that is consistent with the example. Here, FTA's letter F consists of built-in components provided by DSL. To construct the state Q of the FTA, let us assume that each non-terminal symbol $n \in N$ has a pre-defined overall $\{v_1, \ldots, v_k\}$ that it can use. Then we introduce a state for each $n \in N$ and vi $\in U_n$; let's call all non-terminal state sets in N as Q_N. We also construct a set of states Q_T by introducing a state qt for each terminal $t \in T$. Then, the state set in the FTA is given by $Q_N \cup Q_T$.

Next, we use the production P in grammar to construct transformation rules Δ. To define transformation, let's define a function DOM (s), which gives every symbol s the domain of s:

$$\mathrm{DOM}(s) = \begin{cases} s, \text{if } s \text{ is constant} \\ \sigma(s), \text{if } s \text{ is variable} \\ U_s, \text{if } s \text{ is non terminal, and } U_s \text{ is its corresponding universe} \end{cases}$$

Now, consider that generating $n \rightarrow F(s_1, \ldots, s_k)$ forms in the syntax where n is non-terminal and each s_i is a terminal or non-terminal. For each $v_i \in \mathrm{dom}(s_i)$, if there is $v = [[F(v_1, \ldots, v_k)]]$, we add a transition $F(q_{s_1}^{v_1}, \ldots, q_{s_k}^{v_k}) \rightarrow q_n^v$. Additionally, for each variable x we add a transition $x \rightarrow q_x$. Finally, the final state Q_f is a singleton containing the state q_S^o, where S is the start symbol of the grammar and o is the output in the example.

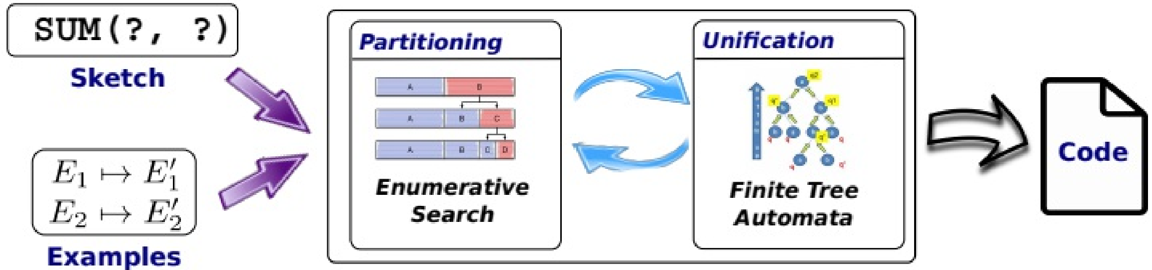

Figure 11. The overview of data completion approach

In view of the general method of building tree automaton for the problem of data completion, the principle of learning algorithm is to construct FTA for each independent example and then intersect them. The final FTA represents the version space of all programs that are consistent with the example. Then we can see how much tree automata are helpful here.

7 Conclusion

In the thesis, an overview of trees and tree automata is presented. We introduce the main theory progress of automata on the structure of infinite trees, which has a very strong expressive power. The basic forms of infinite tree automata are introduced and we discuss about their closure property, regularity and decidability. Among the contributions, Rabin's several theorems are the most beginning of infinite trees and lead to all the refinements and applications afterwards. We also mention the game theory in which a gaming strategy can help us refine the difficult proofs and make them more understandable. After deciding the emptiness problem of infinite tree automata, the introduction to the topological complexity of infinite tree automata lead us to the hierarchical view of them. One important property of infinite tree automata is that they have close relations to logical theory. Many problems are quite difficult to solve in logics, but we may find reducible ways to solve them in automaton theory. Last, we've introduced the applications of tree automata on many practical uses in recent tens of years. As we can see, unlike finite tree automata, automata on infinite trees still remain more in the theory, but the use of them will emerge more and more.

Bibliography

[1] André Arnold, Jacques Duparc, Filip Murlak, and Damian Niwinski. On the topological complexity of tree languages. In *Logic and automata*, pages 9–28.

[2] Vince Bárány, Łukasz Kaiser, and Alex Rabinovich. Expressing cardinality quantifiers in monadic second-order logic over trees. *Fundamenta Informaticae*, 100(1-4):1–17, 2010.

[3] L. Barguno, C. Creus, G. Godoy, F. Jacquemard, and C. Vacher. The emptiness problem for tree automata with global constraints. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 263–272. July 2010.

[4] Achim Blumensath. An algebraic proof of rabin's tree theorem. *Theoretical Computer Science*, 478:1–21, 2013.

[5] Thierry Cachat. Tree automata make ordinal theory easy. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, pages 285–296. Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[6] Arnaud Carayol and Olivier Serre. Counting branches in trees using games. *Information and Computation*, 252:221–242, 2017.

[7] Hubert Comon. Tree automata techniques and applications. *Http://www. grappa. univ-lille3. fr/tata*, 1997.

[8] Jacques Duparc and Filip Murlak. On the topological complexity of weakly recognizable tree languages. In *International Symposium on Fundamentals of Computation Theory*, pages 261–273. Springer, 2007.

[9] P.-C. Héam, V. Hugot, and O. Kouchnarenko. The emptiness problem for tree automata with at least one global disequality constraint is np-hard. *Information Processing Letters*, 118:6–9, 2017.

[10] Haruo Hosoya, Jérôme Vouillon, and Benjamin C Pierce. Regular expression types for xml. In *ACM SIGPLAN Notices*, volume 35, pages 11–22. ACM, 2000.

[11] R. Hossley and C. Rackoff. The emptiness problem for automata on infinite trees. In *13th Annual Symposium on Switching and Automata Theory (swat 1972)*, pages 121–124. Oct 1972.

[12] S. Hummel. Unambiguous Tree Languages Are Topologically Harder Than Deterministic Ones. *ArXiv e-prints*, oct 2012.

[13] Daniel Kirsten. *Alternating Tree Automata and Parity Games*, pages 153–167. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[14] Stephan Kreutzer and Cristian Riveros. Quantitative monadic second-order logic. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 113–122. IEEE Computer Society, 2013.

[15] Sriram C. Krishnan, Anuj Puri, Robert. K. Brayton, and Pravin P. Varaiya. The rabin index and chain automata, with applications to automata and games. In Pierre Wolper, editor, *Computer Aided Verification*, pages 253–266. Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[16] Jonathan May and Kevin Knight. A primer on tree automata software for natural language processing. 2008.

[17] Henryk Michalewski and Damian Niwiński. *On Topological Completeness of Regular Tree Languages*, pages 165–179. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[18] Gerome Miklau and Dan Suciu. Containment and equivalence for an xpath fragment. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 65–76. ACM, 2002.

[19] Frank Neven and Thomas Schwentick. Expressive and efficient pattern languages for tree-structured data. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 145–156. ACM, 2000.

[20] Dominique Perrin and Jean-Éric Pin. Infinite words, automata, semigroups, logic and games, volume 141 of pure and applied mathematics. 2004.

[21] Wolfgang Thomas. Languages, automata, and logic. In *Handbook of formal languages*, pages 389–455. Springer, 1997.

[4][6][5][15][11][13][3][9][17][12][1][8][2][14][21][20][7][18][19][16][10]