State Complexity of Tree Automata

by

Xiaoxue Piao

A thesis submitted to the
Graduate Program in School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada
January 2012

Abstract

Modern applications of XML use automata operating on unranked trees. A common definition of tree automata operating on unranked trees uses a set of vertical states that define the bottom-up computation, and the transitions on vertical states are determined by so called horizontal languages recognized by finite automata on strings. The bottom-up computation of an unranked tree automaton may be either deterministic or nondeterministic, and further variants arise depending on whether the horizontal string languages defining the transitions are represented by DFAs or NFAs. There is also an alternative syntactic definition of determinism introduced by Cristau et al.

It is known that a deterministic tree automaton with the smallest total number of states does not need to be unique nor have the smallest possible number of vertical states. We consider the question by how much we can reduce the total number of states by introducing additional vertical states. We give an upper bound for the state trade-off for deterministic tree automata where the horizontal languages are defined by DFAs, and a lower bound construction that, for variable sized alphabets, is close to the upper bound.

We establish upper and lower bounds for the state complexity of conversions between different types of deterministic and nondeterministic unranked tree automata.

The bounds are, usually, tight for the numbers of vertical states. Because a minimal deterministic unranked tree automaton need not be unique, establishing lower bounds for the number of horizontal states, that is, the combined size of DFAs used to define the horizontal languages, is challenging. Based on existing lower bound results for unambiguous finite automata we develop a lower bound criterion for the number of horizontal states.

We consider the state complexity of operations on regular unranked tree languages. The concatenation of trees can be defined either as a sequential or a parallel operation. Furthermore, there are two essentially different ways to iterate sequential concatenation. We establish tight state complexity bounds for concatenation-like operations. In particular, for sequential concatenation and bottom-up iterated concatenation the bounds differ by an order of magnitude from the corresponding state complexity bounds for regular string languages.

Acknowledgments

I would like to thank my supervisor, Dr. Kai Salomaa, for his constant and patient guidance, invaluable encouragement and thoughtful advice he has given me during my PhD studies. Thanks to the faculty and staff of the school, who welcomed me as one of them and showed me how diligent people worked in the field of science. Thanks to my colleagues and friends, for their constant concerning and encouragement. Thanks to my parents, for their love, understanding and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Since introduced in the 1990's, XML [62] has played an increasingly important role in data representation and exchange through the web. Many concepts in formal language theory [3, 14, 16, 24], e.g., regular expressions, grammars and automata have been used for various XML data management and processing tasks.

Every XML document has a tree structure and for many applications it is adequate to model this structure by labeled trees without data contents. Consequently tree automata become a natural way to represent sets of XML documents. Since there is no a priori restriction on the number of children of a node in an XML document, unranked tree automata [4, 9, 16, 40, 59] are used for XML applications.

Unranked tree automata serve XML research in various ways. One of the major applications of unranked tree automata is to recognize sets of trees that model XML documents. A schema is a set of rules from which an XML document can be generated. It defines the eligible structures for XML documents and gives a precise description of the XML documents which are permitted. We can check whether a given XML document is legal with respect to a schema.

Unranked tree automata are an abstraction of various schemata of XML languages [31, 39]. A Document Type Definition (DTD) is the simplest and most frequently used schema for XML documents and is already specified in the XML standard [62]. There is an equivalent deterministic unranked tree automaton for every DTD. Extended DTDs provide a general framework for XML schemata. Many specific schema languages describing different XML schemata can be defined as restricted variants of EDTDs. Examples include DTDs and restrained competition EDTDs. Restrained competition EDTDs define exactly the languages allowing 1-pass pre-order typing, which is the property that the type of a node can be uniquely determined without looking at its children. For every extended DTDs there is an equivalent non-deterministic unranked tree automaton accepting the same tree language. Thus, unranked tree automata can be used to test whether a tree is valid with respect to a given schema [40]. The validation of an XML document with respect to a schema is the same as the membership problem for automata, however, the form of the input influences the complexity of the validation problem.

Other applications of unranked tree automata on XML processing tasks include XML navigation [37], queries [15] and transformations [61]. The purpose of navigation is to locate positions in XML documents. It usually does not return any output, but is used as a subtask for other processing jobs, such as queries. Navigation can be performed by unranked tree automata equipped with selecting states [12]. Automata serving for XML transformations are usually referred to as transducers. As the transformation language XSLT (Extensible Stylesheet Language Transformations) uses modes that are similar to the states in automata, automata working as tools for XML transformations have been widely investigated [26, 42, 43].

1.1 Unranked tree automata

Tree automata, particularly tree automata operating on unranked trees, have gained renewed interest, as XML has played increasingly important roles in data representation and exchange through the web. The classical tree automata [9, 14] are an extension of finite automata to trees where the label of each node determines the number of children. This type of tree is called a ranked tree. On the other hand, in the trees used to represent the structure of XML documents a given node may have an unbounded (albeit finite) number of children that are ordered from left to right corresponding to the order in XML documents. While the number of transitions of a tree automaton operating on ranked trees is finite, the same is not true when the automata process unranked trees.

One method to handle unranked trees is to encode them as binary trees, and then use the classical theory of ranked tree automata. Two such encodings are considered in [5, 40]. However, the encoding may result in trees of unbounded height since there is no a priori restriction on the number of children of a node in unranked trees. Also depending on various applications, it may be difficult to come up with a proper choice of the encoding method. The other approach that we consider here is to define the computation of the tree automaton directly on unranked XML-trees [9, 59]. XML documents can be abstracted as unranked trees which makes unranked tree automata a natural and fundamental model for various XML processing tasks [7, 16, 8]. An early reference for unranked tree automata is [4].

In unranked trees, the label of a node does not determine the number of children and there is no a priori bound on the number of children of a node. Due to this reason, the set of transitions of an unranked tree automaton is, in general, infinite and the transitions are usually specified in terms of a regular language, which is called a *horizontal language*. Thus, in addition to the finite set of *vertical states* used in the bottom-up computation, an unranked tree automaton needs for each vertical state q and input symbol σ a finite string automaton to recognize the horizontal language consisting of strings of states defining the transitions associated with q and σ [9].

Thus, an unranked tree automaton has two different types of states, horizontal states that are used in a finite string automaton to recognize a *horizontal language* and vertical states that are used in the bottom-up computation.

The total state size of a tree automaton is defined by the number of vertical states and the number of horizontal states used by automata to specify the horizontal languages [35, 46, 49, 50]. Since there is no a priori restriction on the number of children of a node in unranked trees, many problems for unranked tree automata become essentially different and more difficult than the corresponding problems for automata operating on ranked trees (or for ordinary finite automata on strings).

We get different unranked tree automaton models depending on whether the bottom-up computation is nondeterministic or deterministic and whether the horizontal languages are recognized by an NFA or a DFA ((non-)deterministic finite automaton).

It should be noted that there are other deterministic automaton models used for applications on unranked trees, such as syntactically deterministic tree automata [10], stepwise tree automata [5, 7] and nested word automata [1, 2, 41, 45]. The first mentioned model from [10] we consider also in this thesis under the name of strongly deterministic tree automata.

1.2 Descriptional complexity

Regular languages have many representations in the world of finite automata [22]. The analysis of different representations of regular languages is not only restricted to their expressive power, but also with respect to descriptional complexity of these representations. In fact, descriptional complexity is one of the fundamental problems in automata theory. Descriptional complexity measures the succinctness of different models that represent regular languages by the number of states (i.e. state complexity), transitions (i.e. transition complexity), or alphabet size.

Most of the work on descriptional complexity is focused on state complexity, where the size of the automaton is measured by the number of states. The work on state complexity can be classified into areas of representational and operational state complexity. Representational state complexity studies the state complexity of transformations between models. To be more specific, given two classes of automata A and B, how many states are sufficient and necessary in the worst case to construct an automaton from A that is equivalent to an automaton from B? In other words, representational state complexity studies the cost of converting one model to another. Operational state complexity describes how the size of an automaton changes under regularity-preserving operations. For example, how many states are sufficient and necessary in the worst case to recognize the intersection of two automata with m and n states, respectively?

Descriptional complexity of finite automata and regular languages has been extensively studied in recent years. Tight bounds for the state complexity of basic operations and many combined operations, and for the representational state complexity of finite automata and regular languages have been established, see e.g. [11, 17, 20, 21, 22, 25, 30, 33, 56, 64, 65]. Descriptional complexity of extensions of finite automata, such as tree automata [35, 50] or nested word automata [2, 41, 45] has also been considered. The operational state complexity of nested word automata has been studied in [18, 45]. The transformational state complexity of stepwise tree automata is studied in [35]. Much work remains to be done on state complexity of tree automata.

State complexity of operations on regular languages has been comprehensively studied and tight state complexity bounds are known for most operations [27, 64, 66], however, very few results have been obtained for tree automata. Many properties of finite automata operating on strings carry over, more or less straightforwardly, to finite tree automata operating on ranked trees. For example, it is well known that every n-state (bottom-up) nondeterministic tree automaton can be determinized and the minimal deterministic automaton needs in the worst case 2^n states. However, descriptional complexity of unranked tree automata leads to many new questions that will be explored in this thesis. Furthermore, as we will see, even when restricted to ranked trees the state complexity of (certain variants of) operations that extend concatenation or Kleene-star to trees is of a different order of magnitude than the corresponding bounds for regular string languages. There are various representations of unranked tree automata for regular tree languages. It is natural to investigate the efficiency of each model. The study on the state complexity of unranked tree automata helps to optimize the space requirements. Obtaining a proper quantitative understanding of unranked tree automata will be useful also for their applications.

1.3 Contributions of the thesis

Below we list the main contributions of the thesis. Most of the work has appeared or been accepted for publication. In the thesis we study

- state trade-offs in (non-)deterministic unranked tree automata [49]
- the state complexity of the conversions between variants of unranked tree automata [50, 52]
- the operational state complexity of tree languages, including:
    1. union and intersection [46]
    2. concatenation and star [47, 51]
    3. projection and quotient [48]
- tree homomorphisms for unranked trees [53]

In the following subsections we describe the contributions of the thesis in a little more detail. Before that we introduce some notation. We use DTA(DFA) and DTA(NFA) (respectively, NTA(DFA) and NTA(NFA)) to denote the class of deterministic (respectively, nondeterministic) unranked tree automata where the horizontal languages are specified, respectively, by a DFA or an NFA.

1.3.1 State trade-offs

A regular tree language does not need to have a unique minimal DTA(DFA) [35]. In particular, it is possible that an automaton with the smallest total number of states does not have the smallest possible number of vertical states, and we will consider the question by how much we can reduce the number of horizontal states by adding more vertical states, that is, we will consider trade-offs between the numbers of vertical and horizontal states.

We establish upper bounds for the maximal state trade-offs in DTA(DFA)s by considering, roughly speaking, how much smaller can be a set of DFAs whose disjoint union recognizes a regular language defined by a given DFA. Finding the worst-case upper bound leads to questions that are related, but not the same, as the question of maximizing the product of summands considered by Krause [29]. We also give an exponential lower bound for the state trade-offs using a fixed size alphabet. With a variable sized alphabet we give an improved lower bound that by, roughly, doubling the number of vertical states reduces the number of horizontal states from 4^n to $8n$. Also by relying on nondeterministic state complexity of regular languages [19] we show that for DTA(NFA)s there can be no trade-offs between the number of vertical and horizontal states, that is, any regular tree language has a minimal DTA(NFA) that has also the smallest possible number of vertical states. However, this does not mean that a minimal DTA(NFA) would be unique, because it is well known that a minimal NFA for a regular language need not be unique [60, 64].

We also consider corresponding state trade-offs for nondeterministic tree automata. Here the situation becomes more involved due to the fact that also the bottom-up computations with a minimal number of vertical states can be constructed in very different ways. For trade-offs in NTA(DFA)s we give an exponential lower bound, however, establishing lower bounds for NTA(NFA)s is more challenging. We give a lower bound example for NTA(NFA)s where the number of horizontal states is reduced from $O(n^2)$ to $O(n)$ by adding one vertical state.

1.3.2 Conversions between variants of unranked tree automata

We consider bottom-up (frontier-to-root) unranked tree automata. Roughly speaking, we get different models depending on whether the bottom-up computation is nondeterministic or deterministic and whether the horizontal languages are recognized by an NFA or a DFA ((non-)deterministic finite automaton). Furthermore, there is more than one way to define determinism for unranked tree automata and we compare here two of the variants.

The more common definition of determinism [9, 59] requires that for any input symbol σ and two distinct states q_1, q_2, the horizontal languages associated, respectively, with q_1 and σ and with q_2 and σ are disjoint. The condition guarantees that the bottom-up computation assigns a unique state to each node. To distinguish this from the syntactic definition of determinism of [10, 54], we call a deterministic tree automaton where the horizontal languages defining the transitions are specified by DFAs, a *weakly deterministic tree automaton.* A weakly deterministic tree automaton where the horizontal languages are described by DFAs is, using the notation described earlier, a DTA(DFA). Note that a computation of a weakly deterministic tree automaton still needs to "choose" which of the DFAs (associated with different states) is used to process the sequence of states that the computation reached at the children of the current node. Since the intersection of distinct horizontal languages is empty, the choice is unambiguous, however, when beginning to process the sequence of states the automaton has no way of knowing which DFA to use.

A different definition, that we call *strong determinism,* was considered in [10, 54].[1] A strongly deterministic automaton associates to each input symbol a single DFA H_σ

[1] The paper [10] refers to weak and strong determinism, respectively, as semantic and syntactic determinism.

equipped with an output function, and the state at a parent node labeled by σ is determined (via the output function) by the state H_σ reaches after processing the sequence of states corresponding to the children. Strongly deterministic automata can be minimized efficiently and the minimal automaton is unique [10, 54]. On the other hand, it was shown in [35] that for weakly deterministic tree automata the minimization problem is NP-complete and a minimal automaton need not be unique. Another unranked tree automaton model for which the minimal deterministic automaton is unique consists of the *stepwise tree automata* that also allow a Myhill-Nerode type characterization of regularity [35].

We study the state complexity of determinizing different variants of nondeterministic tree automata. That is, we develop upper and lower bounds for the size of deterministic tree automata that are equivalent to given nondeterministic automata. We define the size of an unranked tree automaton as a pair of integers consisting of the number of states used in the bottom-up computation, and the sum of the sizes of the NFAs defining the horizontal languages. Note that the two types of states play very different roles in computations of the tree automaton. The other possibility would be, as is done, e.g., in [35], to count simply the total number of all states in the different components.

Also, we study the state complexity of the conversions between the strongly deterministic tree automata and the weakly deterministic tree automata. Although the former model can be viewed to be more restricted, there exist tree languages for which the size of a strongly deterministic automaton is smaller than the size of the minimal weakly deterministic tree automaton. It turns out to be more difficult to establish lower bounds for the size of weakly deterministic tree automata, that is, DTA(DFA)s than is the case for strongly deterministic automata. Naturally, this can be expected due to the intractability of the minimization of weakly deterministic tree automata [35]. Using lower bound results for unambiguous finite automata [32] we develop a lower bound criterion for the number of horizontal states of a DTA(DFA).

1.3.3 Operational state complexity

We study two different models of determinism for unranked tree automata. As mentioned before, we call the usual deterministic unranked tree automaton [9] model where the horizontal languages defining the transitions are specified by DFAs (deterministic finite automata), a weakly deterministic tree automaton (or DTA(DFA)). We call the other bottom-up deterministic unranked tree automaton model a *strongly deterministic unranked tree automaton* (or SDTA).

We consider the state complexity of union and intersection of unranked tree languages, that is, the question how many states are sufficient and necessary, in the worst case, to recognize the union (respectively, intersection) of tree languages L_1 and L_2 where L_i is recognized by an unranked tree automaton with m_i vertical and n_i horizontal states, $i = 1, 2$. We give upper and lower bounds for the numbers of both vertical and horizontal states for the operations of union and intersection. The upper bounds for vertical states are tight for both SDTAs and DTA(DFA)s. As expected, the upper bounds for the number of vertical states for union and intersection of DTA(DFA)s and SDTAs are similar to the upper bound for the corresponding operation on ordinary string automata. For the number of the horizontal states, we get upper bounds which are almost tight for SDTAs. Obtaining a matching lower bound for the horizontal states of DTA(DFA)s turns out to be very problematic. This is mainly because the minimal DTA(DFA) may not be unique and the minimization of DTA(DFA)s is intractable [35]. Also, as discussed in the previous subsection the number of horizontal states of DTA(DFA)s can be reduced by adding vertical states, i.e., there can be trade-offs between the numbers of horizontal and vertical states, respectively.

Concatenation and iterated concatenation

While the state complexity of Boolean operations is similar in the tree case as for ordinary finite automata operating on strings, the situation becomes essentially more involved when considering concatenation operations where, roughly speaking, we substitute a leaf node of some tree by another tree. It is possible to extend the concatenation operation from strings to trees either as a sequential or a parallel concatenation operation. In the sequential (respectively, parallel) concatenation of trees t_1 and t_2 one occurrence (respectively, all occurrences) of leaves of t_2 having a designated label are replaced by t_1. The operations are extended in the natural way for sets of trees.

In order to keep the connection with string operations more transparent, we define the substitution operation by replacing a leaf (or leaves) of t_2 by t_1. In the context of trees one could define more general substitutions where a node (or nodes) of t_2 with a given label is replaced by t_1, however, this would not change the worst-case state complexity bounds for either sequential or parallel substitutions.

We consider the state complexity of concatenation operations for regular tree languages. As discussed above, when dealing with the weakly deterministic tree automaton model, obtaining tight lower bounds for the number of horizontal states is challenging already in the case of Boolean operations. For this reason when dealing with concatenation (and iterated concatenation) we concentrate only on bounds for vertical states.

We give tight state complexity bounds both for sequential and parallel concatenation. Interestingly, the state complexity of sequential concatenation of tree languages turns out to be of a different order of magnitude than the corresponding bound for regular string languages. The results for parallel concatenation are more similar to the string case.

It should be noted that the weakly deterministic tree automaton model allows some vertical state transitions to be undefined, that is, the model is an incomplete deterministic automaton. The reason for this convention is that the total size (that is, the number of vertical and horizontal states) of an incomplete weakly deterministic automaton and an equivalent completed automaton may be significantly different.[2] In order to keep the state complexity bounds consistent between different models, we allow also ranked deterministic tree automata to be incomplete. In the case of DFAs operating on strings, it is common to give state complexity bounds in terms of complete DFAs, that is, all transitions of a DFA are required to be defined, see, e.g., [22, 64]. The results for concatenation of string languages known in the literature are stated in terms of complete deterministic finite automata (DFAs) [27, 36, 66], and the bounds are slightly different for incomplete DFAs.

As discussed above, concatenation of tree languages can be defined either as a sequential or a parallel operation. We consider iterated concatenation of trees, that is, an extension of the Kleene-star operation for tree languages. It is easy to see that iterated parallel concatenation is not a regularity-preserving operation and, consequently, we will focus on iterated sequential concatenation. Since sequential concatenation of

[2] This will be discussed in Section 2.3

trees is non-associative, there are two essentially different ways to define the corresponding iterated operation. We name these variants as the *bottom-up star* and the *top-down star* operations.

We give tight state complexity bounds for both bottom-up and top-down Kleene-star operations. We show that the bottom-up star of a tree language recognized by a deterministic bottom-up automaton with n states can be recognized by an automaton with $(n + \frac{3}{2}) \cdot 2^{n-1}$ states and, furthermore, there exist worst-case examples where this number of states is needed. This bound is, roughly, n times the corresponding bound for regular string languages. On the other hand, the state complexity of the top-down star operation is shown to coincide with the state complexity of Kleene-star on string languages.

Other operations

Based on the sequential concatenation we define *top-quotient* and *bottom-quotient* operations on trees which correspond, respectively, to right and left quotient of string languages, when dealing with automata that process the input tree from the leaves to the root. We investigate the state complexity of these two operations on deterministic unranked tree automata. For a tree language T recognized by an n vertical state deterministic unranked tree automaton, n vertical states are necessary and sufficient for any deterministic unranked tree automaton to recognize the top-quotient of T with respect to an arbitrary tree language. This result is analogous to the corresponding result for the right-quotient of string languages. However, for the bottom-quotient operation, the tight state complexity bound is $(n + 1)2^n - 1$ which is of a different order than the state complexity of left-quotient for automata operating on strings.

Recall that the state complexity of left-quotient is $2^n - 1$ [64]. The factor $n + 1$ is necessary here because the automaton has to be sure that the concatenation takes place in only one branch of the tree.

The natural projection on strings plays an important role in the field of supervisory control [6, 63]. The natural projection on strings is a mapping that erases from the string all *unobservable symbols*. The underlying alphabet Σ is considered to be a disjoint union of observable and unobservable symbols. The natural projection could be extended to trees in (at least) two different ways. The first possibility is to delete from a given tree all subtrees where the root is labeled by an unobservable symbol. The second possible way to define a natural projection for trees is to delete each node u labeled by an unobservable symbol and then "attach" the children of u as the children of the parent of u (the children will be listed after the left sibling of u and before the right sibling of u). Note that both variants of the definition rely on the fact that we are dealing with unranked trees, i.e., the label of a node does not need to fix the number of children.

Again it is not difficult to see that the latter definition of natural projection does not preserve regularity of tree languages. Hence we concentrate on the first mentioned variant of natural projection on trees. We give a tight upper bound on the number of vertical states that is different than the known state complexity $3 \cdot 2^{n-2} - 1$ of projection for string languages [28, 63].

1.3.4 Tree homomorphisms for unranked trees

Another classical operation on ranked trees is the *tree homomorphism* [14]. A tree homomorphism h is defined by mappings h_m, $m \geq 0$, that associate to each symbol

$\sigma \in \Sigma_m$ a tree $h_m(\sigma)$, possibly over a different ranked alphabet, and having variables $x_1, \ldots, x_m$. The tree homomorphism h then, roughly speaking, replaces each node u labeled by σ with $h_m(\sigma)$ where an occurrence of the variable x_i is replaced by $h(t_i)$, if t_i is the subtree corresponding to the ith child of u, $i = 1, \ldots, m$.

The tree homomorphism is an operation that has no obvious extension for unranked trees and, as far as we are aware, tree homomorphisms for unranked trees have not been considered before. Obviously, when dealing with unranked alphabets, the tree $t_{\sigma,m}$ with m variables that is used to replace an occurrence of a fork $\sigma(x_1, \ldots, x_m)$, for some alphabet symbol σ, should depend on the number of children of the node labeled by σ, that is on m, and m can be arbitrarily large. Thus, it is not at all clear how the trees $t_{\sigma,m}$ should be defined, especially, if we want the mappings to preserve recognizability. In our definition the tree $t_{\sigma,m}$ is output by a string-to-tree transducer that receives as input the string consisting of the node labels of the children of a node labeled by σ.

We focus on mappings that preserve recognizability of tree languages and consider only linear tree homomorphisms, that is, each tree $t_{\sigma,m}$ contains at most one occurrence of each of the variables $x_1, \ldots, x_m$. In the case of unranked trees additional conditions are needed to guarantee that a tree homomorphism preserves recognizability, and arbitrary permutations of the sequence of subtrees cannot be allowed. For example, a mapping that moves the children from odd numbered positions to a contiguous chunk at the beginning, followed by the sequence of children that originally occurred in even numbered positions, does not preserve recognizability. We need to introduce additional restrictions for the string-to-tree transducers in order to guarantee that the defined mappings preserve recognizability.

On the other hand, we attempt to make the definition of tree homomorphisms as general as possible while enforcing the requirement that the mappings preserve recognizability of tree languages. In particular, we want that trees $t_{\sigma,m}$, for sufficiently large m, can include any unranked tree, modulo the restriction of linear variable occurrences.

In order to satisfy these conflicting requirements we introduce string-to-tree transducers that build an output tree in left-to-right depth-first order. Note that models that could build the output tree in parallel at more than one location would define mappings that do not preserve recognizability (at least for naturally defined models that do not involve some artificial restrictions). Another technical issue to deal with in the definition of the transducers is that the output trees need to contain variables from a potentially infinite set, however, the transitions of the transducer need to have a finitary definition. During the computation, the transducer stores variables in a finite number of memory locations, however, the transducer is not able to distinguish between the names of the variables.

Our definition gives one possible way to define tree homomorphisms for unranked trees, however, other extensions of tree homomorphisms from ranked trees to unranked trees could be possible. Establishing bounds for the state complexity of tree homomorphisms remains a topic for future work. If a tree homomorphism h is defined using a string-to-tree transducer M, then a state complexity bound for $h(L)$ would depend on the size of the DTA for the tree language L and on the number of states of the transducer M.

1.4 Organization of the Thesis

We summarize the contents of the thesis. In Chapter 2 we recall definitions for tree automata operating on unranked trees and ranked trees, respectively, some notation, and introduce some lower bound techniques that are used in the later chapters. In Chapter 3 we study the state trade-offs for both deterministic and nondeterministic unranked tree automata. State complexity of conversions between variants of unranked tree automata is presented in Chapter 4. In Chapter 5 we define extensions from strings to trees of operations such as concatenation, iterated concatenation and natural projection. There is more than one way to extend concatenation-like operations from strings to trees. The operational state complexity is investigated in Chapter 6 and Chapter 7. Chapter 8 gives a definition of regularity-preserving tree homomorphisms on unranked trees. Chapter 9 concludes the thesis.

Chapter 2

Preliminaries

Here we recall and introduce some basic definitions concerning strings and trees that will be used in the following subsections. A good general reference on automata and formal languages is the handbook by Rozenberg and A. Salomaa [55]. For more information on tree automata see the electronic book by Comon et al. [9] or the handbook article by Gécseg and Steinby [14].

The set of positive integers is denoted $\mathbb{N}$. The cardinality of a finite set S is denoted $|S|$ and the power set of S is denoted 2^S. When there is no danger for confusion a singleton set $\{s\}$ is denoted simply by s. For a Cartesian product $S = S_1 \times \cdots \times S_n$, the ith projection, $1 \leq i \leq n$, is the mapping $\pi_i : S \longrightarrow S_i$ defined by setting $\pi_i(s_1, \ldots, s_n) = s_i$, $s_j \in S_j$, $j = 1, \ldots, n$. The set of all strings over a set K is denoted K^*.

A nondeterministic finite automaton (NFA) is a tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Σ is the input alphabet, Q is the finite set of states, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \to 2^Q$ is the nondeterministic transition function. A deterministic finite automaton (DFA) is an NFA where the transition function is one-valued, that is, δ is a function $Q \times \Sigma \to Q$. An unambiguous finite automaton (UFA) is an NFA where any accepted string has only one accepting computation. Sometimes we consider also an extension of an NFA where some transitions are labeled by the empty string, this model is called an ϵ-NFA. More information about finite automata on strings can be found in [60, 64].

For a string w over an alphabet Σ, $|w|_\sigma$, $\sigma \in \Sigma$ denotes the number of occurrences of σ in w. For a regular language L, we denote the (respectively, nondeterministic) state complexity of L by $sc(L)$ (respectively, $nsc(L)$) the number of the states of the minimal DFA (respectively, a minimal NFA) recognizing L.

We assume that notions such as the *root*, a *leaf*, a *subtree* and the *height* of a tree are known. We use the convention that the height of a single node tree is zero.

2.1 Basic definitions on trees

A *tree domain* is a prefix-closed subset D of $\mathbb{N}^*$ such that if $ui \in D$, $u \in \mathbb{N}^*$, $i \in \mathbb{N}$ then $uj \in D$ for all $1 \leq j < i$. The set of nodes of a tree t is represented in the well-known way as a tree domain $\mathrm{dom}(t)$ and the labeling of the nodes can be viewed as a mapping $t : \mathrm{dom}(t) \to \Sigma$ where Σ is a finite alphabet of symbols. For $u \in \mathrm{dom}(t)$, $t(u)$ is the element of Σ labeling the node u in the tree t. The label of a node does not determine the number of children, and thus, we use labeled ordered unranked trees. Each node of a tree has a finite number of children with a linear order, but there is no a priori upper bound on the number of children of a node. The set of all Σ-labeled trees is T_Σ. A tree language is any subset of T_Σ.

We introduce the following notation for trees. For $i \geq 0$, $a \in \Sigma$ and $t \in T_\Sigma$, we

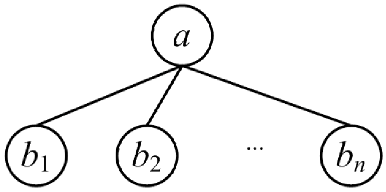

Figure 2.1: The tree $a(b_1, b_2, ..., b_n)$

denote by $a^i(t) = a(a(...a(t)...))$ a tree, where the nodes ε, 1, ..., 1^{i-1} are labeled by a and the subtree at node 1^i is t. When $a \in \Sigma$, $w = b_1b_2...b_n \in \Sigma^*$, we use $a(w)$ to denote a tree $a(b_1, b_2, ..., b_n)$ shown in Figure 2.1, which is also referred to as a *fork*. When L is a set of strings, $a(L) = \{a(w) \mid w \in L\}$. The set of all Σ-trees where exactly one leaf is labeled by a special symbol x ($x \notin \Sigma$) is $T_\Sigma[x]$. For $t \in T_\Sigma[x]$ and $t' \in T_\Sigma$, $t(x \leftarrow t')$ denotes the tree obtained from t by replacing the unique occurrence of variable x by t'.

For $\sigma \in \Sigma$ and $t \in T_\Sigma$, $\mathrm{leaf}(t, \sigma) \subseteq \mathrm{dom}(t)$ denotes the set of leaves of t with label σ. Let t be a tree and u some node of t. The tree obtained from t by replacing the subtree at node u with a tree s is denoted $t(u \leftarrow s)$. The notation is extended in the natural way for a set of pairwise independent nodes U of t and $S \subseteq T_\Sigma$: $t(U \leftarrow S)$ is the set of trees obtained from t by replacing each node of U by some tree in S.

A ranked alphabet is a pair (Σ, r) where Σ is a finite set and $r : \Sigma \longrightarrow \mathbb{N} \cup \{0\}$ is a function that associates with each element $\sigma \in \Sigma$ its rank $r(\sigma)$. The set of elements of rank m is Σ_m, $m \geq 0$. Usually instead of (Σ, r) we speak of the ranked alphabet Σ and assume that r is known. The set of trees over a ranked alphabet Σ, is the smallest set S satisfying the condition: if $m \geq 0$, $\sigma \in \Sigma_m$ and $t_1, \ldots, t_m \in S$ then $\sigma(t_1, \ldots, t_m) \in S$. That is, trees over a ranked alphabet Σ are Σ-labeled trees where a node with m children is labeled by an element of Σ_m. Note that in order to make the distinction more transparent, the set of all trees over a ranked alphabet is denoted

F_Σ (as opposed to T_Σ in the unranked case).

In the following two subsections, we define tree automata operating on ranked and unranked trees, respectively. In each case the automaton reads the tree from the leaves towards the root, and the states at the children of a node u determine the state at node u. In the case of ranked trees, the number of transitions is finite and a tree automaton can be specified simply by listing the transitions. On the other hand, the transitions of an unranked tree automaton need to be specified using regular languages, called horizontal languages. Naturally a ranked tree automaton is a special case of an unranked tree automaton. We include a separate definition for ranked tree automata mainly because the notations are much simpler when we can just list the transitions of the automaton. Some state complexity lower bounds can be reached by tree languages over a ranked alphabet, and in such cases it is convenient to restrict consideration to automata operating on ranked trees.

2.2 Ranked tree automata

A *nondeterministic bottom-up tree automaton*[1] (NTA) is a four-tuple $A = (\Sigma, Q, Q_F, g)$ where Σ is a ranked alphabet, Q is a finite set of states, $Q_F \subseteq Q$ is a set of accepting states and g associates to each $\sigma \in \Sigma_m$ a mapping $\sigma_g : Q^m \longrightarrow 2^Q$, $m \geq 0$. For each $t = \sigma(t_1, \ldots, t_m) \in F_\Sigma$ we define inductively the set $t_g \subseteq Q$ by setting $q \in t_g$ if and only if there exist $q_i \in (t_i)_g$, $i = 1, \ldots, m$, such that $q \in \sigma_g(q_1, \ldots, q_m)$. Intuitively, t_g consists of the states of Q that A may reach at the root of t. The tree language

[1] We view trees to be drawn with the root at the top, and hence a bottom-up automaton processes an input tree starting from the leaves.

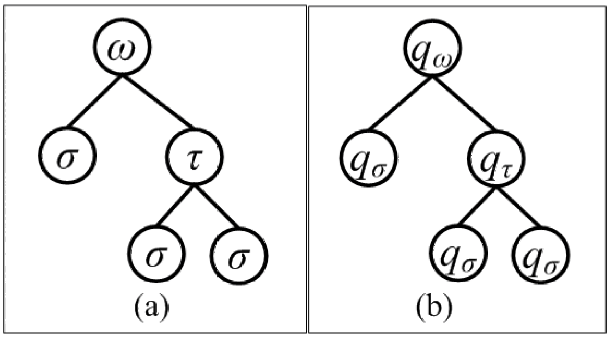

Figure 2.2: An input tree t (a) and the states A assigned to each node in t (b)

accepted by A is $L(A) = \{t \in F_\Sigma \mid t_g \cap Q_F \neq \emptyset\}$. The intermediate stages of a computation, or *configurations,* of A are trees where some leaves may be labeled by states of A. Thus the set of configurations of A consists of Σ'-trees where $\Sigma'_0 = \Sigma_0 \cup \{Q\}$ and $\Sigma'_m = \Sigma_m$ when $m \geq 1$. The set of configurations is denoted as $F_\Sigma[Q]$.

The automaton A is *deterministic* (a DTA) if for each $\sigma \in \Sigma_m$ ($m \geq 0$), σ_g, is a partial function $Q^m \longrightarrow Q$. The nondeterministic (bottom-up or top-down) and deterministic bottom-up tree automata accept the family of *regular tree languages* [9, 14]. A Kleene's Theorem for the family of regular tree languages is defined in [14].

Example 1. Consider a ranked tree automaton $A = (\{\omega, \tau, \sigma\}, \{q_\omega, q_\tau, q_\sigma\}, \{q_\omega\}, g)$, where

$$\sigma_g = q_\sigma, \quad \tau_g(q_\sigma, q_\sigma) = q_\tau, \quad \omega_g(q_\sigma, q_\tau) = q_\omega.$$

An input tree t is shown in Figure 2.2 (a) and the states assigned to each node in t are shown in Figure 2.2 (b).

Note that we allow a deterministic tree automaton to have undefined transitions, that is, σ_g, $\sigma \in \Sigma_m$ is a partial function $Q^m \rightarrow Q$. While adding a dead state to an incomplete ranked tree automaton changes the number of states only by one, the situation is essentially different in the case of unranked tree automata, see Remark 3 in Section 2.3. In order to make the state complexity bounds compatible between the two models, we allow also the ranked tree automata to be incomplete.

Most of our work uses the more general model of tree automata operating on unranked trees that will be defined in the next subsection. We have here given the more restricted definition of ranked tree automata because the model is considerably simpler and it will be used in Chapter 6 where essentially similar state complexity lower bounds can be achieved with ranked automata as with the more general (and more complicated) model of unranked tree automata.

2.3 Unranked tree automata

A *nondeterministic unranked tree automaton* is a tuple $A = (Q, \Sigma, \delta, F)$, where Q is the finite set of states, Σ is the alphabet labeling nodes of input trees, $F \subseteq Q$ is the set of final states, and δ is a mapping from $Q \times \Sigma$ to the subsets of Q^* which satisfies the condition that, for each $q \in Q$, $\sigma \in \Sigma$, $\delta(q, \sigma)$ is a regular language. The language $\delta(q, \sigma)$ is called the *horizontal language* associated with q and σ.

A computation of A on a tree $t \in T_\Sigma$ is a mapping $C : \mathrm{dom}(t) \to Q$ such that for $u \in \mathrm{dom}(t)$, if $u \cdot 1, \ldots u \cdot m$, $m \geq 1$, are the children of u then $C(u \cdot 1) \cdots C(u \cdot m) \in \delta(C(u), t(u))$. In case u is a leaf the condition means that $m = 0$ and $\varepsilon \in \delta(C(u), t(u))$.

Intuitively, if a computation of A has reached the children of a σ-labeled node u in a sequence of states $q_1, q_2, \ldots, q_m$, the computation may nondeterministically assign a state q to the node u provided that $q_1 q_2 \cdots q_m \in \delta(q, \sigma)$. For $t \in T_\Sigma$, $t^A \subseteq Q$ denotes the set of states that in some bottom-up computation A may reach at the root of t. We extend this notation for a string $w \in \Sigma^*$, by setting $w^A \in Q^*$ to denote the string of states that A reaches at leaves labeled by elements of the string w. The *tree*

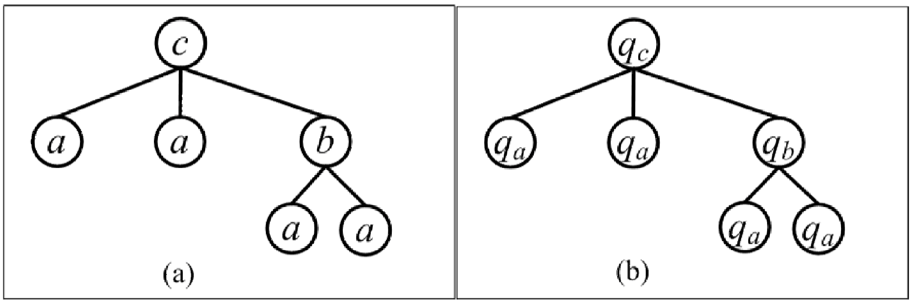

Figure 2.3: An input tree t (a) and vertical states A assigned to each node in t (b)

language recognized by A is defined as $L(A) = \{t \in T_\Sigma \mid t^A \cap F \neq \emptyset\}$.

For a tree automaton $A = (Q, \Sigma, \delta, F)$, we denote by $H^A_{q,\sigma}$, $q \in Q$, $\sigma \in \Sigma$, a nondeterministic finite automaton (NFA) on strings recognizing the horizontal language $\delta(q, \sigma)$. The NFA $H^A_{q,\sigma}$ is called a *horizontal NFA*, and states of different horizontal automata are called collectively *horizontal states.* We refer to the states of Q that are used in the bottom-up computation as *vertical states.*

Example 2. Consider the following bottom-up unranked tree automaton as an example. Let automaton $A = (\{q_a, q_b, q_c\}, \{a, b, c\}, \delta, \{q_c\})$, where

$$\delta(q_a, a) = \epsilon, \qquad \delta(q_b, b) = q_a q_a, \qquad \delta(q_c, c) = q_a q_a q_b.$$

Let t be an input tree shown in Figure 2.3 (a). The vertical states that A assigned to each node in t are shown in Figure 2.3 (b). The horizontal languages associated with q_b, b and q_c, c, respectively, are shown in Figure 2.4. Figure 2.5 shows the horizontal DFA $H^A_{q_c,c}$ recognizing the horizontal language $\delta(q_c, c)$ and $\{0, 1, 2, 3\}$ are the horizontal states.

A tree automaton $A = (Q, \Sigma, \delta, F)$ is said to be (semantically) *deterministic* if for $\sigma \in \Sigma$ and any two states $q_1 \neq q_2$, $\delta(q_1, \sigma) \cap \delta(q_2, \sigma) = \emptyset$. The above condition guarantees that the state assigned by A to a node u of an input tree is unique (but

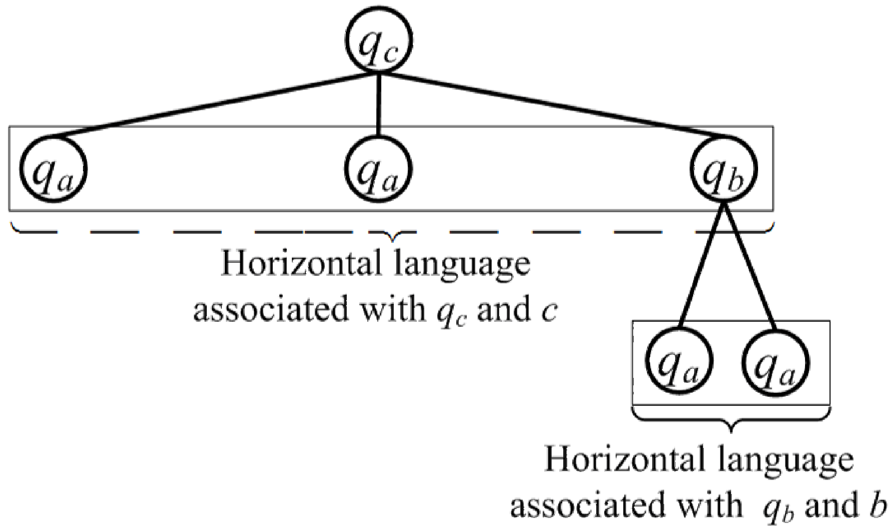

Figure 2.4: Horizontal languages

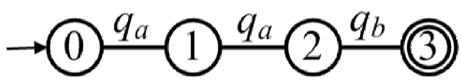

Figure 2.5: Horizontal DFA and horizontal states

A need not assign any state to u if the computation becomes blocked below u).

We get a further refinement of classes of automata depending on whether the horizontal languages are defined using DFAs or NFAs. We use NTA(M) or DTA(M), respectively, to denote (the class of) nondeterministic or deterministic tree automata where the horizontal languages are specified by the elements in class M. For example, NTA(DFA) denotes the nondeterministic tree automata where the horizontal languages are recognized by a DFA.

Note that when referring to a tree automaton $A = (Q, \Sigma, \delta, F)$ it is always assumed that the relation δ is specified in terms of automata $H^A_{q,\sigma}$, $q \in Q$, $\sigma \in \Sigma$, and by saying that A is an NTA(DFA) we indicate that each $H^A_{q,\sigma}$ is a DFA.

If A is a DTA(NFA), for any tree $t \in T_\Sigma$ the bottom-up computation of A assigns a unique vertical state to the root of t, that is, t^A is a singleton set or empty. If the horizontal automata $H^A_{q,\sigma}$ are DFAs, furthermore, for each transition the sequence of horizontal states is processed deterministically. However, a computation that has reached children of a σ-labeled node in a sequence of states $w \in Q^*$ still needs to make the choice which of the DFAs $H^A_{q,\sigma}$, $q \in Q$, is used to process w. For this reason we consider also the following notion introduced in [10] that we call strong determinism.

A tree automaton $A = (Q, \Sigma, \delta, F)$ is said to be *strongly deterministic* if for each $\sigma \in \Sigma$, the transitions are defined by a single DFA augmented with an output function as follows. For $\sigma \in \Sigma$ define

$$H^A_\sigma = (S_\sigma, Q, \gamma_\sigma, s^0_\sigma, F_\sigma, \lambda_\sigma), \tag{2.1}$$

where $(S_\sigma, Q, \gamma_\sigma, s^0_\sigma, F_\sigma)$ is a DFA over input alphabet Q, with set of states S_σ where $s^0_\sigma \in S_\sigma$ is the start state, $F_\sigma \subseteq S_\sigma$ is the set of final states and $\gamma_\sigma : S_\sigma \times Q \to S_\sigma$ is the transition function, and λ_σ is a function $F_\sigma \to Q$. Then we require that for all $q \in Q$ and $\sigma \in \Sigma$: $\delta(q, \sigma) = \{w \in Q^* \mid \lambda_\sigma(\gamma_\sigma(s^0_\sigma, w)) = q\}$. Note that the definition guarantees that $\delta(q_1, \sigma) \cap \delta(q_2, \sigma) = \emptyset$ for any distinct $q_1, q_2 \in Q$, $\sigma \in \Sigma$. The class of strongly deterministic tree automata is denoted as SDTA.[2]

By the size of an NFA B, denoted size(B), we mean the number of states of B. Because in the computations of an unranked tree automaton the roles played by vertical and horizontal states, respectively, are essentially different, when we want to measure the size of an automaton more precisely we count the two types of states separately. The size of an NTA(NFA) $A = (Q, \Sigma, \delta, F)$ is defined as[3]

$$\text{size}(A) = \left[\ |Q|;\ \sum_{q \in Q, \sigma \in \Sigma} \text{size}(H^A_{q,\sigma})\ \right] \quad (\in \mathbb{N} \times \mathbb{N}). \tag{2.2}$$

[2]Strictly speaking, δ is superfluous in the tuple specifying an SDTA and the original definition of [10] gives instead the automata H^A_σ, $\sigma \in \Sigma$. We use δ in order to make the notation compatible with our other models, and to avoid having to define bottom-up computations of SDTAs separately.

[3]For readability we use [;] to denote a pair instead of (,).

Using notations of (2.1), the size of an SDTA A is defined as the pair of integers size$(A) = [\ |Q|;\ \sum_{\sigma\in\Sigma} |S_\sigma|\ ]$. Comparisons between pairs of integers are done componentwise, that is, we denote $[x_1, y_1] \leq [x_2, y_2]$ if $x_1 \leq x_2$ and $y_1 \leq y_2$.

For any unranked tree automaton $A = (Q, \Sigma, \delta, F)$, we denote as tsize$(A)$ the total number of vertical and horizontal states

$$\mathrm{tsize}(A) = |Q| + \sum_{q\in Q,\sigma\in\Sigma} \mathrm{size}(H^A_{q,\sigma}). \tag{2.3}$$

We say that a DTA(DFA) (NTA(NFA), NTA(DFA), DTA(NFA), respectively) A is *v-minimal* if A has the smallest number of vertical states among all the DTA(DFA)s (NTA(NFA)s, NTA(DFA)s, DTA(NFA)s, respectively) that recognize $L(A)$, and A is *t-minimal* if A has the smallest total number of states.

It is known that a DTA(DFA) with a minimal total number of states for a regular tree language need not be unique [35]. In particular, it is possible that an automaton with the smallest total number of states does not have the smallest possible number of vertical states.

Remark 3. *For deterministic tree automata operating on unranked trees, the sizes of an incomplete deterministic automaton and the corresponding completed version may be significantly different. Adding a dead state for the bottom-up computation, requires adding, corresponding to an input symbol σ, a horizontal language L that is the complement of a finite disjoint union $L(A_1) \cup \ldots \cup L(A_m)$ where A_i, $i = 1, \ldots, m$, are the DFAs recognizing the horizontal languages corresponding to symbol σ and the states of the incomplete automaton. The size of the minimal DFA for L may be considerably larger than the sum of the sizes of A_i, $i = 1, \ldots, m$. When considering state complexity of tree automata operating on unranked trees it is convenient to allow the use of incomplete automata.*

2.4 Lower bound techniques

It is well known that each regular ranked tree language has a unique minimal DTA that can, furthermore, be computed efficiently [14]. The situation is essentially more complicated in the case of unranked tree automata and, in particular, the DTA(DFA) with the smallest total number of (vertical and horizontal) states need not be unique. In this subsection we provide several lower bound criteria that will be used later on for our state complexity bounds for unranked tree automata.

The following two lemmas provide lower bound estimates for vertical and horizontal states of SDTAs, respectively. These are, essentially, consequences of the Myhill-Nerode type characterization of regular unranked tree languages given in [4, 10].

For $t \in T_\Sigma$, a leaf u of t, and $q \in Q$ we denote by $t(u \leftarrow q)$ the tree obtained from t by replacing the label of u by q. We say that states q_1 and q_2 of a DTA(DFA) A are *equivalent* if for any $t \in T_\Sigma$ and any leaf u of t, the computation of A accepts $t(u \leftarrow q_1)$ if and only if it accepts $t(u \leftarrow q_2)$.

For our purposes it is convenient to have separate conditions for vertical states and for horizontal states, although it is possible to have a Myhill-Nerode characterization that integrates the conditions for the vertical and horizontal states into a single subtree equivalence relation.

Lemma 4. *Let A be an SDTA or a DTA(NFA) with a set of vertical states Q recognizing a tree language L. Assume $R = \{t_1, \ldots, t_m\} \subseteq T_\Sigma$ where for any $1 \leq i < j \leq m$ there exists $t \in T_\Sigma[x]$ such that $t(x \leftarrow t_i) \in L$ if and only if $t(x \leftarrow t_j) \notin L$. Then $|Q| \geq |R| - 1$.*

Proof. The condition of the lemma guarantees that the state of A can be undefined at the root of at most one of the trees of R. If Q has less than $|R| - 1$ states, then A must reach in the same state the root of two distinct trees $t_1, t_2 \in R$. According to the assumption, there exists $t \in T_\Sigma[x]$ such that $t(x \leftarrow t_1) \in L$ if and only if $t(x \leftarrow t_2) \notin L$. This is a contradiction. ■

Lemma 5. *Let A be an SDTA with a set of vertical states Q recognizing a tree language L. Let S be a finite set of tuples of Σ-trees and let $b \in \Sigma$. Assume that for any distinct tuples $(r_1, \ldots, r_m)$, $(s_1, \ldots, s_n) \in S$ there exists $t \in T_\Sigma[x]$ and a sequence of trees $u_1, \ldots, u_k$ such that*

$$t(x \leftarrow b(r_1, \ldots, r_m, u_1, \ldots, u_k)) \in L \text{ if and only if } t(x \leftarrow b(s_1, \ldots, s_n, u_1, \ldots, u_k)) \notin L. \tag{2.4}$$

Then the horizontal automaton H_b^A needs at least $|S| - 1$ states.

Proof. If H_b^A has less than $|S| - 1$ states, then for two distinct tuples $(r_1, \ldots, r_m)$, $(s_1, \ldots, s_n)$ of S the automaton H_b^A must be in the same state after reading the strings $r_1^A \cdots r_m^A$ and $s_1^A \cdots s_n^A$ ($\in Q^*$). Note that the condition (2.4) guarantees that at most one tuple of S can contain a tree r for which r^A is undefined. Now A reaches the same vertical state at roots of $t(x \leftarrow b(r_1, \ldots, r_m, u_1, \ldots, u_k))$ and $t(x \leftarrow b(s_1, \ldots, s_n, u_1, \ldots, u_k))$. This contradicts (2.4). ■

By extending the Myhill-Nerode congruence from words to trees, we can prove the following proposition.

Proposition 6. *The v-minimal DTA(DFA) of a regular tree language is unique.*

Proof. Let $A = (Q, \Sigma, \delta, F)$ be a DTA(DFA). We say that states $q_1, q_2 \in Q$ are *equivalent* if for any $t \in T_\Sigma$ and any leaf u of t, the computation of A accepts $t(u \leftarrow q_1)$ if and only if it accepts $t(u \leftarrow q_2)$.

Denote $L = L(A)$ and the above equivalence relation by $\equiv_L$. Now exactly as in the case of ranked tree automata [14] it follows that the vertical states of any v-minimal DTA(DFA) equivalent to A consist of the congruence classes of $\equiv_L$, that is, the set of vertical states is determined by the tree language L. The operations on the congruence classes then uniquely determine the horizontal languages. [4] Note that in any v-minimal DTA(DFA) B equivalent to A, an individual congruence class cannot be divided into separate states because this would destroy the v-minimality of B. ■

The extension of the Nerode congruence is called the top-congruence in [4], and the corresponding construction for tree languages over ranked alphabets can be found in [14]. A Myhill-Nerode theorem for stepwise automata on unranked trees can be found in [35].

Next we develop a lower bound criterion for the number of horizontal states in DTA(DFA)s. Since there is no unique DTA(DFA) with the smallest total number of states, determining lower bounds for horizontal states is more involved than in the criterion of Lemma 5 used above for SDTAs. As will be seen in Chapter 3 it is often possible to reduce the number of horizontal states by introducing redundant copies of vertical states, that is, replacing a horizontal language as a disjoint union of one or more languages. The proof of Lemma 7 below is based on the idea that (at least in certain restricted situations) if in a v-minimal DTA(DFA) we replace a vertical state q with equivalent copies $p_1, \ldots, p_k$ for an input symbol σ, the horizontal DFAs corresponding to σ and each of the p_i's, yield an unambiguous NFA for the horizontal language corresponding to σ and the original vertical state q. This result together

[4] As noted in [4, 10], for unranked tree languages in addition to the congruence having a finite index we need further conditions to guarantee regularity of the horizontal languages. However, in Proposition 6 we just need a v-minimal deterministic automaton for a tree language that is known to be regular, and we do not need to consider the additional conditions.

with existing lower bounds for the size of unambiguous NFAs [32, 58] can then be used as a lower bound criterion for the number of horizontal states in DTA(DFA)s.

We first introduce some technical notions used in the lower bound condition. Let L be a regular tree language over Σ. We say that $\sigma \in \Sigma$ is a *unique height-one label for L* if

1. in trees of L, σ occurs only as a label of nodes of height one, and,
2. for any $\tau_1, \tau_2 \in \Sigma$, $\tau_1 \neq \tau_2$, that label leaves below a height one node u labeled by σ, τ_1 and τ_2 are inequivalent with respect to the Nerode congruence of the tree language L.

Note that any DTA(DFA) can assign only one state to a leaf node. It follows that if σ is a unique height-one label for tree language L, any DTA(DFA) recognizing L must assign a distinct state to all leaves occurring below a node labeled by σ. This observation will be used in the proof of Lemma 7.

Let A be a v-minimal DTA(DFA) and B an arbitrary DTA(DFA) recognizing $L(A)$. By considering the usual equivalence relation among vertical states of B, as was defined in the proof of Proposition 6, we see that the states of B can be partitioned into equivalence classes, each consisting pairwise equivalent states (in terms of the vertical computation) that were used to replace one vertical state of A. For a state q of A, we denote by $[q]_B$ the set of states of B that correspond to q in this way.

Recall that if A is a DTA(DFA), the DFA recognizing the horizontal language associated with state q and symbol σ is denoted as $H^A_{q,\sigma}$. We denote the number of states of $H^A_{q,\sigma}$ as $|H^A_{q,\sigma}|$.

Lemma 7. *Let $A = (Q_A, \Sigma, \delta_A, F_A)$ be a v-minimal DTA(DFA) for tree language L and let $B = (Q_B, \Sigma, \delta_B, F_B)$ be an arbitrary DTA(DFA) for L.*

Suppose that $\sigma \in \Sigma$ is a unique height-one label for L. Then for any $q \in Q_A$,

$$\sum_{p \in [q]_B} |H^B_{p,\sigma}|$$

is greater or equal than the size of the smallest unambiguous NFA for $L(H^A_{q,\sigma})$. (Following [32] we allow here an NFA to have multiple initial states.)

Proof. Since σ is a unique height-one label for L, both A and B assign a unique state to any leaf labeled by τ that in some tree of L may occur below a height-one node labeled by σ. Denote by Ω_σ the set of symbols that in trees of L may label a leaf occurring below a node labeled by σ. Let f_σ be the bijection

$$P_{\text{states\_below\_}\sigma} = \{s \in Q_B \mid \varepsilon \in \delta_B(s,\tau), \tau \in \Omega_\sigma\} \to \{r \in Q_A \mid \varepsilon \in \delta_A(r,\tau), \tau \in \Omega_\sigma\}$$

that for each $\tau \in \Omega_\sigma$, maps the unique $s \in Q_B$ such that $\varepsilon \in \delta_B(s,\tau)$ to the unique $r \in Q_A$ such that $\varepsilon \in \delta_A(r,\tau)$.

Let $q \in Q_A$ be arbitrary (such that $\delta_A(q,\sigma) \neq \emptyset$). Let $t = \sigma(\tau_1,\ldots,\tau_m)$ be a tree of height one with root labeled by σ. For any $p \in [q]_B$, if B reaches the root of t in state p then A reaches the root of t in state q, and conversely if A reaches the root of t in state q then B reaches the root of t in some of the states of $[q]_B$. This means that from the DFAs $H^B_{p,\sigma}$, $p \in [q]_B$, we can construct for the language $\delta_A(q,\sigma)$ a DFA C with multiple initial states simply by taking their disjoint union and replacing each transition labeled by $s \in P_{\text{states\_below\_}\sigma}$ by $f_\sigma(s)$. Since B is deterministic, the languages $L(H^B_{p_1,\sigma})$ and $L(H^B_{p_2,\sigma})$, $p_1, p_2 \in [q]_B$, $p_1 \neq p_2$, are disjoint. This means that for $w \in Q_A^*$, C has at most one accepting computation on w and C is an unambiguous NFA (with multiple initial states). ■

Please note that Lemma 7 gives a method to prove a lower bound for the number of the horizontal states without the influence of the state trade-offs.

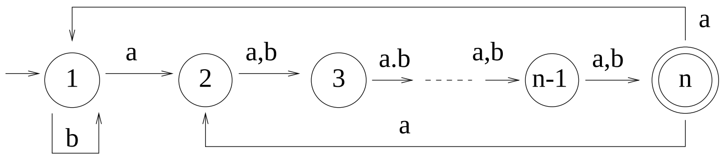

Figure 2.6: An NFA for the language L_{Moore}.

Lemma 7 gives a lower bound condition for the number horizontal states in terms of the size of a smallest unambiguous NFA for the horizontal language. A useful lower bound condition for the size of unambiguous NFAs has been given by [58], for a good presentation of this result see also [32]. We denote the *rank* of a (square) matrix M as $\text{rank}(M)$.

Theorem 8. [58] *Given a regular language L and strings x_i, y_i for $i = 1, \ldots, n$, let M be an $n \times n$ matrix such that $M[x_i, y_i] = 1$ if $x_i y_i \in L$, and 0 otherwise. Then any UFA for L has at least rank(M) states.*

2.5 Auxiliary results on regular string languages

We recall here some properties of finite automata on strings and establish a technical property of a particular unary regular language. The properties will be used in Section 4.2.2.

It is well known that for each $n \geq 1$ there exists an NFA with n states such that the minimal equivalent DFA needs 2^n states [38, 64].

Example 9. [38] *Let L_{Moore} be the language defined by the NFA in Figure 2.6. It is shown already by Moore [38] that any DFA for the language L_{Moore} needs at least 2^n states.*

We recall some notation concerning unary DFAs. An arbitrary unary DFA recognizing an infinite language can be written as a tuple

$$A = (Q, \{a\}, \delta, q_0, F) \tag{2.5}$$

where $Q = \{q_0, \ldots, q_{h+k-1}\}$, $h \geq 0$, $k \geq 1$, $F \subseteq Q$ and $\delta(q_i, a) = q_{i+1}$, $0 \leq i < h+k-1$, $\delta(q_{h+k-1}, a) = q_h$. The states $q_0, \ldots, q_{h-1}$ are called the *tail* of A and the states $q_h, \ldots, q_{h+k-1}$ are called the *cycle* of A.

Lemma 10. *Let p_1, …, p_n be the first n primes. Suppose that R_1 and R_2 are a partition of $\{1, \ldots, n\}$. Define*

$$L_0 = \{\ a^j \ \mid\ [(\forall r \in R_1)\, j \equiv 0 \pmod{p_r}] \ \textit{and}\ [(\forall r' \in R_2)\, j \not\equiv 0 \pmod{p_{r'}}]\ \}.$$

Assume L_1 is a regular infinite subset of L_0. If A is a DFA recognizing L_1 then the cycle of A has length at least $\Pi_{i=1}^n p_i$.

Proof. We use for A notations as in (2.5). Now the claim can be written as $k \geq \Pi_{i=1}^n p_i$.

Since L_1 is infinite, the cycle of A has an accepting state q_{h-1+x}, $1 \leq x \leq k$.

First consider an arbitrary $r \in R_1$. Now $a^{h-1+x} \in L_1$ and $a^{h-1+x+k} \in L_1$. Hence $h-1+x \equiv 0 \pmod{p_r}$ and $h-1+x+k \equiv 0 \pmod{p_r}$, which implies that p_r divides k.

Next consider an arbitrary $r' \in R_2$ and for the sake of contradiction assume that $k \not\equiv 0 \pmod{p_{r'}}$. This means that the equation $u \cdot k \equiv 1 \pmod{p_{r'}}$ has a solution u_0 for u. Choose $y \in \{0, \ldots, p_{r'} - 1\}$ such that $h - 1 + x \equiv y \pmod{p_{r'}}$.

Since $a^{h-1+x} \in L_1$ and k is the length of the cycle of A, it follows that also $a^{h-1+x+(p_{r'}-y)u_0 k} \in L_1$. This is a contradiction because, by the choice of u_0 and y, we have $h - 1 + x + (p_{r'} - y)u_0 k \equiv 0 \pmod{p_{r'}}$, and by the definition of L_0 the prime $p_{r'}$

should not divide the length of a word in $L_1 \subseteq L_0$.

We have shown that each p_i, $1 \leq i \leq n$, divides k and the claim follows. ■

Chapter 3

State trade-offs in unranked tree automata

It is known that a regular tree language does not, in general, have a unique minimal DTA(DFA) [35], and in particular, the DTA(DFA) with the smallest total number of states does not need to have the smallest possible number of vertical states. This is illustrated in the following example by giving a regular language L_0 with state complexity 256 such that L_0 is a disjoint union of eight languages requiring a total of 176 states. Thus, by replacing one vertical state that has horizontal language L_0 by eight distinct vertical states (that could be equivalent in the bottom-up computation) we could reduce the total size of A.

Example 11. *Recall that we denote the size of a minimal DFA for a regular language L, or the state complexity of L, as* $\mathrm{sc}(L)$*. Consider the language*

$$L_0 = (a + b)^* b(a + b)^7.$$

The minimal DFA for L_0 has 256 states and L_0 can be represented as a disjoint

Table 3.1: State complexity of disjoint union

Language (L_i)	*sc*(L_i)	*Language* (L_i)	*sc*(L_i)
$L_1 = (a+b)^*bbbb(a+b)^4$	*29*	$L_2 = (a+b)^*babb(a+b)^4$	*23*
$L_3 = (a+b)^*bbab(a+b)^4$	*23*	$L_4 = (a+b)^*bbba(a+b)^4$	*19*
$L_5 = (a+b)^*baab(a+b)^4$	*22*	$L_6 = (a+b)^*baba(a+b)^4$	*22*
$L_7 = (a+b)^*bbaa(a+b)^4$	*19*	$L_8 = (a+b)^*baaa(a+b)^4$	*19*

union of languages L_i*,* $1 \leq i \leq 8$*, that are listed in Table 3.1 together with the state complexity of each language. From the table we see that* $\sum_{i=1}^{8} \mathrm{sc}(L_i) = 176$*.*

On the other hand, by defining an equivalence relation for the set of vertical states (as in the case for tree automata on ranked trees) it is easy to see that any regular tree language has a unique DTA(DFA) with the smallest number of vertical states, as stated in Proposition 6. By a *state trade-off* we mean a situation where we add to a DTA(DFA) additional vertical states in a way that reduces the total number of states. Since minimization of the total number of states of a DTA(DFA) is NP-complete [35], questions of state trade-offs can be expected to be hard.

We establish upper bounds for the maximal state trade-offs in DTA(DFA)s and give exponential lower bounds for the state trade-offs. Also by relying on nondeterministic state complexity of regular languages [19] we show that for DTA(NFA)s there can be no trade-offs between the number of vertical and horizontal states, that is, any regular tree language has a minimal DTA(NFA) that has also the smallest possible number of vertical states. However, this does not mean that a minimal DTA(NFA) would be unique, because it is well known that a minimal NFA for a regular language need not be unique [60, 64].

We also consider corresponding state trade-offs for nondeterministic tree automata.

Here the situation becomes more involved due to the fact that also the bottom-up computations with a minimal number of vertical states can be constructed in very different ways.

3.1 Maximal trade-off in DTA(DFA)

In this section, we first investigate the upper bounds on the state trade-offs in a DTA(DFA). After that we give two lower bound results, with a fixed size and a non-fixed size alphabet, respectively.

Lemma 12. *Let $A = (Q, \Sigma, \delta, F)$ be an arbitrary DTA(DFA) with n vertical states, $Q = \{1, \ldots, n\}$, and for each $\sigma \in \Sigma$, let $L_{i,\sigma}$, $1 \leq i \leq n$ be the horizontal language associated with σ and state i. Assume that $L_{i,\sigma} = \bigoplus_{j=1}^{k_i} L_{i,\sigma,j}$, $k_i \geq 1$, where $\bigoplus$ denotes disjoint union.*

Then there exists a DTA(DFA) B equivalent to A where state i is replaced by k_i equivalent states and the transitions of B associated with symbol σ need at most $\sum_{i=1}^{n} \sum_{j=1}^{k_i} sc(L_{i,\sigma,j})$ horizontal states.

Proof. Let $B = (Q', \Sigma, \delta', F')$, where $Q' = \{(i, j) \mid 1 \leq i \leq n, 1 \leq j \leq k_i\}$, and the δ'-transitions are defined for each $\sigma \in \Sigma$ and $(i, j) \in Q'$ by setting

$$\delta'((i, j), \sigma) = \{ \ (i_1, j_1)(i_2, j_2) \cdots (i_m, j_m) \ \mid \ m \geq 0, \ i_1 i_2 \cdots i_m \in L_{i,\sigma,j} \ \}.$$

A state (i, j) is in F' if $i \in F$. Since by our assumption the languages L_{i,σ,j_1} and L_{i,σ,j_2} are always disjoint when $j_1 \neq j_2$, the tree automaton B is deterministic.

The total number of horizontal states needed for transitions associated with σ is $\sum_{i=1}^{n} \sum_{j=1}^{k_i} sc(L_{i,\sigma,j})$. ■

Since the state complexity of a regular language $L_{i,\sigma}$ may be considerably larger than $\sum_{j=1}^{k_i} sc(L_{i,\sigma,j})$, Lemma 12 gives a method to reduce the total number of states in a DTA(DFA). In order to establish an upper bound for the worst-case state trade-off, we first observe that any trade-off for a v-minimal DTA(DFA) has to be based on a construction where a given vertical state and its horizontal language are replaced by equivalent vertical states with disjoint horizontal languages.

Suppose that A is the unique v-minimal DTA(DFA) (as given by Proposition 6), and B is any DTA(DFA) that is equivalent to A. By considering the standard equivalence relation among vertical states of B (as defined in the the proof of Proposition 6), it is easy to see that B is a "refinement" of A where each vertical state of A has been replaced by one or more equivalent vertical states. Thus, when considering the trade-off between the number of vertical and horizontal states, it is sufficient to restrict consideration to situations where in a v-minimal DTA(DFA) we replace each vertical state by a number of states that are equivalent (in terms of the vertical computation), as described in the statement of Lemma 12.

In the following we want to identify the maximal reduction of the total number of states that may result when we replace one vertical state i of a v-minimal DTA(DFA) A by k_i vertical states. Recalling the state complexity of (disjoint) union [64] and using the notations of Lemma 12, the maximal number of horizontal states corresponding to i and $\sigma \in \Sigma$ in A is $\Pi_{j=1}^{k_i}\mathrm{sc}(L_{i,\sigma,j})$ and in the modified DTA(DFA) B these are replaced by $\sum_{j=1}^{k_i}\mathrm{sc}(L_{i,\sigma,j})$ horizontal states and $k_i - 1$ additional vertical states. Thus the question of finding the maximal trade-off amounts to finding $k_i \geq 1$ and horizontal languages $L_{i,\sigma,j}$, $1 \leq j \leq k_i$, such that the

$$\text{value } \Pi_{j=1}^{k_i}\mathrm{sc}(L_{i,\sigma,j}) \text{ is maximized as a function of } \sum_{j=1}^{k_i}(\mathrm{sc}(L_{i,\sigma,j}) + 1). \tag{3.1}$$

We note that Krause [29] considers a related, but different, problem of maximizing the product of integers $\Pi_{j=1}^{k} d_j$ as a function of their sum $\sum_{j=1}^{k} d_j$. Our solution is inspired by the solution given in [29], but the case analysis for our problem turns out to be a bit more complicated.

In order to solve (3.1), define $\{x_1, \ldots, x_k\}$, $x_i \in \mathbb{N}$, $i = 1, \ldots, k$, $k \geq 1$ to be a *partition* of $s \in \mathbb{N}$ if $s = \sum_{i=1}^{k}(x_i + 1)$. A partition is *a winning partition* if $\prod_{i=1}^{k} x_i$ is maximal among all partitions of s. We observe that the following properties hold for a winning partition.

1. No winning partition can contain a one (except when $s = 2$). If $x_i = 1$, we just take it out and add two to some other x_j.

2. No winning partition needs to contain a six because a six can be replaced by a two and a three. (Note an extra vertical state is needed.)

3. No x_i can be greater than six. If $x_i > 6$ is even, then $x/2(x/2 - 1) > x$. If $x_i > 6$ is odd, then $(x-1)/2 \cdot (x-1)/2 > x$. In both cases P is not a winning partition.

4. P cannot contain a two and a five because they could be replaced by a three and a four.

5. P cannot contain a two and a four because they could be replaced by two three's.

6. P cannot contain two two's because they could be replaced by a five.

7. P cannot contain a two, and two three's because they could be replaced by a four and a five.

Table 3.2: Winning partitions for $s \leq 11$

s	vertical states	horizontal states (P)
1	-	-
2	1	1
3	1	2
4	1	3
5	1	4
6	1	5
7	2	$2, 3$
8	2	$3, 3$
9	2	$3, 4$
10	2	$4, 4$
11	2	$4, 5$

The above means that $x_i = 2$ can only occur in P when $P = \{2, 3\}$ and $s = 7$, or $P = \{2\}$ and $s = 3$.

The winning partitions for small values of s are listed in Table 3.2. In the following we restrict consideration to cases $s \geq 12$. Up to now we have concluded that the only numbers that can occur in P are three, four and five.

8. P cannot contain two five's because they could be replaced by three three's.

9. P cannot contain a three and a five because they could be replaced by two four's.

Now we know that either P contains only four's and one five, or only three's and four's.

10. P cannot contain two four's and a five because they could be replaced by four three's.

Thus, the number five can occur only in cases $s = 11$ and $s = 6$.

11. P cannot contain five three's because they could be replaced by four four's.

Thus, we know that when $s \geq 12$, P contains only three's and four's, and at most four three's.

Clearly if P is a winning partition, the sum of $(x_i + 1)$'s is exactly s (because otherwise we could add one to one of the x_i's).

Putting the above together means that writing $s = \sum_{j=1}^{k}(\mathrm{sc}(L_{\sigma,j})+1)$, $s \geq 12$, and $t = \Pi_{j=1}^{k}\mathrm{sc}(L_{\sigma,j})$, we have exactly one possible solution for (3.1) that is determined as follows:

- If $s \equiv 0 (\mathrm{mod}\ 5)$, then t is maximized to $4^{s/5}$ when each $\mathrm{sc}(L_{\sigma,j}) = 4$,
- If $s \equiv 1 (\mathrm{mod}\ 5)$, then t is maximized to $3^4 \cdot 4^{(s-16)/5}$ when four of the $\mathrm{sc}(L_{\sigma,j})$'s are equal to 3, and the rest of the $\mathrm{sc}(L_{\sigma,j})$'s are equal to 4,
- If $s \equiv 2 (\mathrm{mod}\ 5)$, then t is maximized to $3^3 \cdot 4^{(s-12)/5}$ when three of the $\mathrm{sc}(L_{\sigma,j})$'s are equal to 3, and the rest of the $\mathrm{sc}(L_{\sigma,j})$'s are equal to 4,
- If $s \equiv 3 (\mathrm{mod}\ 5)$, then t is maximized to $3^2 \cdot 4^{(s-8)/5}$ when two of the $\mathrm{sc}(L_{\sigma,j})$'s are equal to 3, and the rest of the $\mathrm{sc}(L_{\sigma,j})$'s are equal to 4,
- If $s \equiv 4 (\mathrm{mod}\ 5)$, then t is maximized to $3 \cdot 4^{(s-4)/5}$ when one of the $\mathrm{sc}(L_{\sigma,j})$'s is equal to 3, and the rest of the $\mathrm{sc}(L_{\sigma,j})$'s are equal to 4.

The above analysis allows us to give an upper bound for the worst-case trade-off that can be obtained based on the method of Lemma 12.

Theorem 13. *Using the notations of Lemma 12, let* $s_i = \sum_{j=1}^{k_i}(\mathrm{sc}(L_{i,\sigma,j}) + 1)$, $1 \leq i \leq n$,[1] *and assume that* $s_i \geq 12$. *Let* $X, Y, Z, U, V \subseteq \{1, 2, \ldots, n\}$ *be the sets of* s_i*'s*

[1]To be completely general the values s_i depend, in addition to $i \in Q$ also on a symbol $\sigma \in \Sigma$. We have omitted σ in the notation to avoid making the formulas even more complicated.

defined by the conditions

- $s_x \equiv 0 \pmod 5$, $x \in X$, $\quad s_y \equiv 1 \pmod 5$, $y \in Y$, $\quad s_z \equiv 2 \pmod 5$, $z \in Z$,
- $s_u \equiv 3 \pmod 5$, $u \in U$, $\quad s_v \equiv 4 \pmod 5$, $v \in V$.

The maximal trade-off (corresponding to (3.1)) occurs when the tree automata A and B (as in Lemma 12) have the following numbers, respectively, of vertical and horizontal states:

$$size(A) = [n; \quad \textstyle\sum_{x\in X} 4^{s_x/5} + \sum_{y\in Y} 3^4 \cdot 4^{(s_y-16)/5} + \sum_{z\in Z} 3^3 \cdot 4^{(s_z-12)/5} + \sum_{u\in U} 3^2 \cdot 4^{(s_u-8)/5} + \sum_{v\in V} 3 \cdot 4^{(s_v-4)/5}]$$

and

$$size(B) = [\textstyle\sum_{x\in X} s_x/5 + \sum_{y\in Y}(4+(s_y-16)/5) + \sum_{z\in Z}(3+(s_z-12)/5) + \sum_{u\in U}(2+(s_u-8)/5) + \sum_{v\in V}(1+(s_v-4)/5); \quad \sum_{i=1}^{n} s_i'],$$

where

$$s_i' = \begin{cases} \frac{4}{5}s_i \text{ if } s_i \equiv 0 \pmod 5, \\ \frac{4}{5}(s_i-16)+12 \text{ if } s_i \equiv 1 \pmod 5, \\ \frac{4}{5}(s_i-12)+9 \text{ if } s_i \equiv 2 \pmod 5, \\ \frac{4}{5}(s_i-8)+6 \text{ if } s_i \equiv 3 \pmod 5, \\ \frac{4}{5}(s_i-4)+3 \text{ if } s_i \equiv 4 \pmod 5. \end{cases} \qquad 1 \le i \le n.$$

For $s_i < 12$, the maximal trade-offs are listed in Table 3.3.

3.1.1 Lower bound results

We first present a lower bound result for an alphabet of fixed size and afterwards give a better lower bound result on an alphabet depending on n. Recall that the size of a DTA(DFA) A is defined in (2.2), and the tsize of a DTA(DFA) is defined in (2.3).

Table 3.3: Maximum trade-offs when $s_i \leq 11$

	A		B		
s_i	vertical states	horizontal states	vertical states	horizontal states	trade-offs
2	1	1	1	1	0
3	1	2	1	2	0
4	1	3	1	3	0
5	1	4	1	4	0
6	1	5	1	5	0
7	1	6	2	$2, 3$	0
8	1	9	2	$3, 3$	2
9	1	12	2	$3, 4$	4
10	1	16	2	$4, 4$	7
11	1	20	2	$4, 5$	10

Lemma 14. *There exists a tree language T over alphabet $\Sigma = \{a, b, c\}$ where any v-minimal DTA(DFA) for T has at least three vertical states and at least 2^n, $n \geq 2$ horizontal states, and T is recognized by a DTA(DFA) B with size$(B) = [n+1; n^2+n]$.*

Proof. We define the tree language $T = \{c^i(w) \mid i \geq 1, w \in L\}$, where L is the union of the languages accepted by DFAs $A_1, A_2, \ldots, A_{n-1}$ shown in Figure 3.1.

Let $A = (Q, \Sigma, \delta, F)$ be a v-minimal DTA(DFA) with three vertical states that

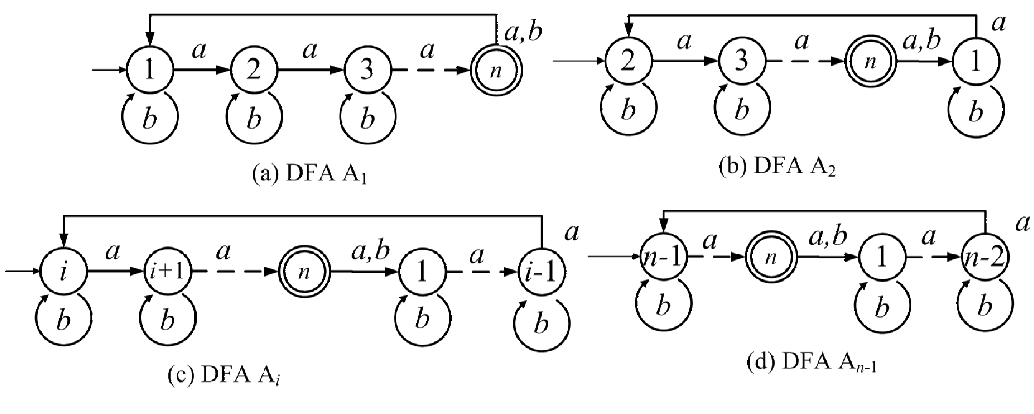

Figure 3.1: DFAs A_i

recognizes T. Clearly A must assign different states q_a and q_b to leaves labeled by a and b, respectively. Assume that A assigns q_a to the root of a tree $c(w)$, $w \in \{a,b\}^*$. This means that A would have to accept trees obtained from $t \in T$ by replacing the leaves labeled by a with $c(w)$, and we conclude that A would not recognize T. In exactly the same way we see that A cannot assign q_b to the root of any tree $c(w)$, $w \in \{a,b\}^*$. Thus, A has only one state q such that $\delta(q,c) \neq \emptyset$. This argument also establishes that any DTA(DFA) for T needs at least three vertical states. Since A recognizes T, we conclude that $\delta(q,c) \cap Q^+ = q + \{w^A \mid w \in \cup_{i=1}^{n-1} L(A_i)\}$. The horizontal language $\{w^A \mid w \in \cup_{i=1}^{n-1} L(A_i)\}$ can be accepted by a $(n-1)$-entry DFA, and any equivalent DFA has at least $2^n - 2$ states according to Lemma 3 in [23]. Then a DFA needs at least 2^n states to recognize $\delta(q,c)$. Thus, we have that A has at least three vertical states and 2^n horizontal states.

Let $B = (Q', \{a,b,c\}, \delta', F')$, where $Q' = \{p_1, p_2, \ldots, p_{n-1}, p_a, p_b\}$, $\delta'(p_a, a) = \epsilon$, $\delta'(p_b, b) = \epsilon$, $\delta'(p_i, c) = p_i + \{s^B \mid s \in L(A_i)\}$, $F' = \{p_1, p_2, \ldots, p_{n-1}\}$. It is obvious that $L(B) = T$. Since $L(A_i) \cap L(A_j) = \emptyset$ when $i \neq j$, the tree automaton B is a DTA(DFA). Each horizontal DFA needs at least $n+2$ states to recognize $\delta'(p_i, c)$, and two horizontal states are needed in total to recognize $\delta'(p_a, a)$ and $\delta'(p_b, b)$. We have $size(B) = [n+1; n^2+n]$. Although all the vertical states $\{p_1, p_2, \ldots, p_{n-1}\}$ are equivalent in B, B has a total size that is much smaller than A. ■

The upper bound in Theorem 13 and the lower bound in Lemma 14 do not match. We can get a better lower bound using an alphabet that depends on n.

Lemma 15. *Let $\Sigma = \{a_0, a_1, \ldots, a_n\}$, $n \geq 1$. There exists a tree language T over Σ such that any v-minimal DTA(DFA) A for T has $n+1$ vertical states and at least $2^{2n+1} - 3^n - 1 + n$ horizontal states, and T has a DTA(DFA) B such that*

$size(B) = [2n; 8n]$.

Proof. We define the tree language $T = \{a_0^i(w) \mid i \geq 1, w \in L\}$, where L is the union of the languages $L_i = \{wa_i \in \{a_1, \ldots, a_n\}^* \mid |w|_{a_i} \equiv 0 \bmod 4\}$, $1 \leq i \leq n$.

Let $A = (Q, \Sigma, \delta, F)$ be a v-minimal DTA(DFA) with $n + 1$ vertical states that recognizes T. Clearly A must assign different states q_i, $1 \leq i \leq n$ to leaves labeled by a_i. Assume that A assigns q_i to the root of a tree $a_0(w)$, $w \in \{a_1, \ldots, a_n\}^*$. This means that A would have to accept trees obtained from $t \in T$ by replacing the leaves labeled by a_i with $a_0(w)$, and we conclude that A would not recognize T. Thus, A has only one state q_0 such that $\delta(q_0, a_0) \neq \emptyset$. This argument also establishes that any DTA(DFA) for T needs at least $n + 1$ vertical states. Since A recognizes T, we conclude that $\delta(q_0, a_0) = q_0 + \{w^A \mid w \in \cup_{i=1}^n L_i\}$. The minimal DFA H for the horizontal language $\{w^A \mid w \in \cup_{i=1}^n L_i\}$ has $2^{2n+1} - 3^n - 1$ states. H has non-final states $(j_1, ..., j_n)$ where each $j_i \in \{0, 1, 2, 3\}$ keeps track of the number of symbols a_i mod 4, and the last seen symbol was some a_k where $j_k \neq 1$. There are $4^n - 1$ such states (since the state $(1, 1, \ldots, 1)$ is unreachable). H has final states $(j_1, ..., j_n, f)$ where each j_i keeps track of the number of symbols a_i mod 4, and the last seen symbol was some a_k where $j_k = 1$. The last component f is used just to differentiate the state from the corresponding non-final state $(j_1, ..., j_n)$. There are $4^n - 3^n$ such states. It is easy to see that all states of H are pairwise inequivalent. Since A needs a singleton DFA for the horizontal language associated with q_i, $1 \leq i \leq n$, A has in total at least $2^{2n+1} - 3^n - 1 + n$ horizontal states.

Let $B = (Q', \Sigma, \delta', F')$, where $Q' = \{p_1, p_2, \ldots, p_n, h_1, \ldots, h_n\}$, $\delta(h_i, a_i) = \epsilon$ for $1 \leq i \leq n$ and $\delta'(p_i, a_0) = p_i + \{s^B \mid s \in L_i\}$, $F' = \{p_1, p_2, \ldots, p_n\}$. It is obvious that $L(B) = T$. Since $L_i \cap L_j = \emptyset$ when $i \neq j$, the tree automaton B is a DTA(DFA). We have $size(B) = [2n; 8n]$. ■

The lower bound construction of Lemma 15 can be compared with the worst case upper bound of Theorem 13 as follows. Consider A and B as in the statement of Theorem 13. Denote by n the number of vertical states of A and suppose that the number of vertical states of B is $c \cdot n$ (in the construction of Lemma 15 we have $c = 2$). In this case the difference between the numbers of horizontal states, respectively, of A and of B would be at most constant times 4^n.

Thus, at least for the particular numbers of vertical states given by the construction of Lemma 15, the lower bound is reasonably close to the worst-case upper bound. However, Lemma 15 uses an alphabet of size n and for a fixed sized alphabet in Lemma 14 the conversion from A to B increases the number of vertical states by a non-constant factor while the number of horizontal states of A is only $2^{\sqrt{m}}$ where m stands for the number of horizontal states of B.

3.2 Results for DTA(NFA)

Compared with a DTA(DFA), it is interesting that it turns out that there are no trade-offs between the numbers of vertical and horizontal states in a DTA(NFA). For any regular tree language there exists a v-minimal DTA(NFA) that is also a t-minimal DTA(NFA). Recall that nsc(L) denotes the number of the states of a minimal NFA recognizing the regular language L, or the nondeterministic state complexity of L, and the tsize of an unranked tree automaton is defined in (2.3).

Theorem 16. *For any DTA(NFA) A, there always exists an equivalent DTA(NFA) B such that*

(1) B has a minimal number of vertical states,

(2) $tsize(B) \leq tsize(A)$.

Proof. Let T be an arbitrary regular tree language, and $A = \{Q, \Sigma, \delta, F\}$ be a DTA(NFA) recognizing T such that the number of vertical states in A is not minimal. (B could just be equal to A if A is v-minimal.) Since the number of vertical states in A is not minimal, there exist at least two states $q, q' \in Q$ that are equivalent [9] (as defined in the proof of Proposition 6). Without loss of generality, we assume that for $\sigma \in \Sigma$, $\delta(q, \sigma) = L_1$ and $\delta(q', \sigma) = L_2$. Now construct a DTA(NFA) $B = \{Q', \Sigma, \delta', F'\}$, where $Q' = Q - \{q, q'\} + \{p\}$. The transition function δ' is defined as same as δ except that

1) transition rules $\delta(q, \sigma) = L_1$ and $\delta(q', \sigma) = L_2$ are replaced by $\delta'(p, \sigma) = L_1 \cup L_2$,

2) all occurrences of q, q' in δ are replaced by p.

The final set of states $F' = F$ if $q, q' \notin F$, and $F' = F - \{q, q'\} + \{p\}$ if $q, q' \in F$. (Note that since q, q' are equivalent states, either both of them are final or neither of them is final.) It is easy to see that A is equivalent to B. According to the results in [19], we know that

$$nsc(L_1 \cup L_2) \leq nsc(L_1) + nsc(L_2) + 1.$$

Now we have that

$$tsize(A) - tsize(B) = |Q| + nsc(L_1) + nsc(L_2) - (|Q| - 1 + nsc(L_1 \cup L_2)) \geq 0.$$

We repeat the above process if the number of vertical states in B is not minimal. ■

The following corollary is immediate.

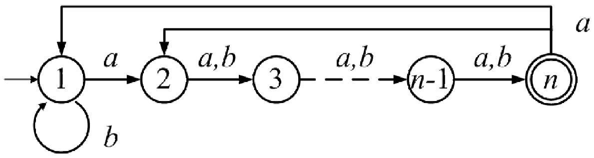

Figure 3.2: NFA A_0

Corollary 17. *Any regular tree language has a DTA(NFA) A that is both v-minimal and t-minimal, and A can be effectively constructed.*

3.3 Results for NTA(DFA)

We state a lower bound construction, where we reduce the number of horizontal states exponentially by adding only one vertical state. Recall that as defined in Section 2.3, for a string w and an automaton A, w^A denotes the string of states that A reaches at leaves labeled by elements of the string w.

Example 18. *Let* $\Sigma = \{a, b, c\}$. *Consider a tree language* $T = \{c^i(w) \mid 1 \leq i \leq m-2, m \geq 3, w \in L\}$ *where* L *is the language recognized by an NFA* A_0 *shown in Figure 3.2.*

T has an NTA(DFA) $M = (\{q_a, q_b, q_0, q_1, \ldots, q_{m-3}\}, \Sigma, \delta, \{q_0, q_1, \ldots, q_{m-3}\})$ *where*

- $\delta(q_a, a) = \epsilon$, $\delta(q_b, b) = \epsilon$, $\delta(q_{m-3}, c) = \{s^M \mid s \in L\}$,
- $\delta(q_{j-1}, c) = q_j$, *where* $1 \leq j \leq m-3$.

M is v-minimal. The horizontal language $\delta(q_{m-3}, c)$ *is recognized by an NFA* A_1 *shown in Figure 3.3. According to [38], any DFA equivalent to* A_1 *needs at least* 2^n *states. Thus, we have* $size(M) = [m, 2^n + 2m - 4]$.

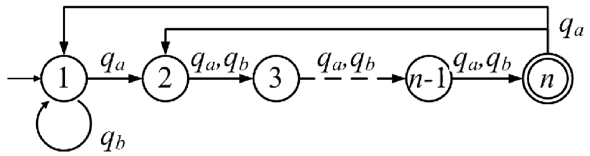

Figure 3.3: The NFA A_1 for the horizontal language $\delta(q_{m-3}, c)$

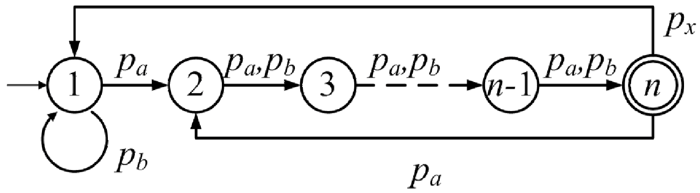

Figure 3.4: The DFA for the horizontal language $\delta'(p_{m-3}, c)$

The tree language T has also the following NTA(DFA). Define

$$N = (\{p_a, p_b, p_x, p_0, p_1, \ldots, p_{m-3}\}, \Sigma, \delta', \{p_0, p_1, \ldots, p_{m-3}\})$$

where

- $\delta'(p_a, a) = \epsilon, \quad \delta'(p_b, b) = \epsilon, \quad \delta'(p_x, a) = \epsilon, \quad \delta'(p_{m-3}, c) = \{s^N \mid s \in L\},$
- $\delta'(p_{j-1}, c) = p_j$, *where* $1 \leq j \leq m-3$.

The horizontal language $\delta'(p_{m-3}, c)$ is recognized by a DFA shown in Figure 3.4. Thus, we have size$(N) = [m+1; n+2m-3]$.

Using the fact that the minimal syntactically deterministic bottom-up automaton is unique [10], and the conversions between NTA(DFA)s and syntactically deterministic automata (which will be discussed in Chapter 4), we can get an upper bound for the trade-off that is exponential in the total number of states (vertical and horizontal states). Note that the syntactically deterministic automata of [10] are called "strongly deterministic" in the thesis. Since also the nondeterministic vertical computation of an NTA(DFA) (or NTA(NFA)) with a smallest number of vertical states can be constructed in very different ways, we do not have a good upper bound for the trade-offs in these cases.

3.4 Results for NTA(NFA)

In the same way as described at the end of the previous section, we can get a computable upper bound for state trade-offs in NTA(NFA)s, but again we do not have a good upper bound. Also finding lower bounds turns out to be more difficult in the case of NTA(NFA)s. Note that the construction used in Example 18 relies essentially on the property that the horizontal languages are recognized by a DFA and similar constructions do not seem to work for NTA(NFA)s.

In Example 19 below, we give a construction that reduces the number of horizontal states from $O(n^2)$ to $O(n)$ by adding only one vertical state.

Example 19. *Let $\Sigma = \{a, c, d\}$ be the alphabet, $p_1, \ldots, p_n$ be any integers and $P = p_1 \cdot \ldots \cdot p_n$. Define tree language T to consist of trees*

$$\{a^i(d^m) \mid 1 \leq i \leq n,\ m \equiv 0(\text{mod } p_i)\} \cup \{a^i c(d^m) \mid 1 \leq i \leq n,\ m \equiv 0(\text{mod } P)\}.$$

T can be recognized by an NTA(NFA) $A = (\{q_1, \ldots, q_n, q_d\}, \Sigma, \delta, \{q_1\})$, where δ-transition is defined as:

- $\delta(q_d, d) = \epsilon,$ $\quad\quad$ $\delta(q_i, a) = (q_d^{p_i})^* + q_{i+1}$, *for* $1 \leq i \leq n-1$,
- $\delta(q_n, a) = (q_d^{p_n})^*,$ $\quad$ $\delta(q_i, c) = (q_d^{P})^*$, *for* $1 \leq i \leq n$,

It is easy to verify that A has the smallest number of vertical states. To count the number of a's, A needs at least n states. (The number of a's is from 1 to n.) A needs a state for leaf nodes labeled by d. If A assigned any other state in $\{q_1, \ldots, q_n\}$

to a node labeled by d, the automaton will accept some trees that are not in T. For instance, if A assigns q_2 to a leaf node labeled by d, A will accepts tree $a(d)$, which is not in T. (Node labeled by d is assigned with state q_2 and according to $\delta(q_1, a) = q_2$, a final state is assigned to the root of $a(d)$.) Any NFA for $\delta(q_i, a) = (q_d^{p_i})^* + q_{i+1}$ needs at least $p_i + 2$ states. Thus, the size of A is

$$size(A) = [n+1;\ nP + \sum_{i=1}^{n} p_i + 2n - 1].$$

By adding one more vertical state q_c to A, we can construct an equivalent NTA(NFA) $B = (\{q_1, \dots, q_n, q_d, q_c\}, \Sigma, \delta', \{q_1\})$, where δ'-transition is defined as:

- $\delta'(q_d, d) = \epsilon$, $\delta'(q_i, a) = (q_d^{p_i})^* + q_{i+1} + q_c$, *for* $1 \leq i \leq n-1$,
- $\delta'(q_n, a) = (q_d^{p_n})^* + q_c$, $\delta'(q_c, c) = (q_d^{P})^*$.

The size of B is

$$size(B) = [n+2;\ P + \sum_{i=1}^{n} p_i + 2n + 1].$$

By choosing $n = P$, we have that $tsize(A) = O(n^2)$ and $tsize(B) = O(n)$, which means that in the worst case the number of horizontal states can be reduced to the square root by adding one more vertical state.

Chapter 4

Conversions between variants of unranked tree automata

We study the state complexity of determinizing different variants of nondeterministic tree automata. That is, we develop upper and lower bounds for the size of deterministic tree automata that are equivalent to given nondeterministic automata.

Also, we study the state complexity of the conversions between the strongly deterministic tree automata and DTA(DFA)s. Although the former model can be viewed to be more restricted, there exist tree languages for which the size of a strongly deterministic automaton is smaller than the size of the minimal DTA(DFA)s. It turns out to be more difficult to establish lower bounds for the size of DTA(DFA)s than is the case for strongly deterministic automata. Naturally, this can be expected due to the intractability of the minimization of DTA(DFA)s [35].

In this chapter, we make the following notational convention that allows us to use symbols of Σ in the definition of horizontal languages. Unless otherwise mentioned, we assume that a tree automaton always assigns to each leaf symbol labeled σ a state $\overline{\sigma}$ that is not used anywhere else in the computation. That is, for $\sigma \in \Sigma$ and $q \in Q$, $\varepsilon \in \delta(q, \sigma)$ only if $q = \overline{\sigma}$, $\delta(\overline{\sigma}, \sigma) = \{\varepsilon\}$ and $\delta(\overline{\tau}, \sigma) = \emptyset$ for all $\sigma, \tau \in \Sigma$, $\sigma \neq \tau$. When there is no confusion, we denote also $\overline{\sigma}$ simply by σ. When the alphabet Σ is fixed, there is only a constant number of the special states $\overline{\sigma}$ and since, furthermore, the special states have the same function in all types of tree automata, for simplicity, we do not include them when counting the number of vertical states. The purpose of this convention is to improve readability: many of our constructions become more transparent when alphabet symbols can be used explicitly to define horizontal languages. The convention does not change our state complexity bounds that are generally given within a multiplicative constant.

4.1 Size comparison of the strongly deterministic tree automata and DTA(DFA)s

Here we give upper and lower bounds for the size of a DTA(DFA) simulating a strongly deterministic one (an SDTA), and vice versa. The computation of a DTA(DFA) can, in some sense, nondeterministically choose which of the horizontal DFAs it uses at each transition. An SDTA does not have this capability and it can be expected that, in the worst case, an SDTA may need considerably more states than an equivalent DTA(DFA). However, there exist also tree languages for which an SDTA can be considerably more succinct than a DTA(DFA).

4.1.1 Converting an SDTA to a DTA(DFA)

We show that an SDTA can be quadratically smaller than a DTA(DFA). This can be compared with [35] where it was shown that deterministic stepwise tree automata can be quadratically smaller than SDTA's (that are called dPUTAs in [35]).

The upper bound for the conversion is as expected but we include a short proof. As introduced in Section 2.3, for an SDTA A we denote the deterministic automata for the corresponding horizontal languages by H_σ^A, $\sigma \in \Sigma$.

Lemma 20. *Let $A = (Q, \Sigma, \delta, F)$ be an arbitrary SDTA. We can construct an equivalent DTA(DFA) A' where*

$$\text{size}(A') \leq \left[\ |Q|;\ |Q| \times \sum_{\sigma \in \Sigma} \text{size}(H_\sigma^A)\ \right]. \tag{4.1}$$

Proof. For $\sigma \in \Sigma$ denote the components of H_σ^A as in (2.1). Construct an equivalent DTA(DFA) $A' = (Q, \Sigma, \delta', F)$, where for each $\sigma \in \Sigma$, $q \in Q$, $\delta'(q, \sigma) = \{w \in Q^* \mid \lambda_\sigma(\gamma_\sigma(s_\sigma^0, w)) = q\}$. The languages $\delta'(q_1, \sigma)$ and $\delta'(q_2, \sigma)$, $q_1 \neq q_2$ are always disjoint, and $\delta'(q, \sigma)$ is recognized by a DFA obtained from H_σ^A by choosing as the set of final states $\lambda_\sigma^{-1}(q)$, $q \in Q$, $\sigma \in \Sigma$. The construction does not change the number of vertical states and (4.1) holds. ∎

Next we give a lower bound for the conversion. The construction is based simply on the idea that an SDTA B uses a single DFA H_σ^B (equipped with an output mapping) to compute all transitions associated with $\sigma \in \Sigma$, while a DTA(DFA) needs a separate DFA corresponding to each vertical state for the transition associated with σ. Thus, we can use a construction where a large prefix of the horizontal languages corresponding to different vertical states is identical, and H_σ^B distinguishes the state to be output based on a suffix of logarithmic length.

Lemma 21. *Let $n, z \in \mathbb{N}$ and choose $\Sigma = \{a, b, 0, 1\}$. There exists an SDTA B with input alphabet Σ, n vertical states and $z + 4n$ horizontal states, such that any DTA(DFA) for the tree language $L(B)$ has at least n vertical states and $n \cdot (\lfloor \log n \rfloor + 2 + z)$ horizontal states.*

Proof. Let $n, z \geq 1$ be arbitrary but fixed. For $1 \leq i \leq n$, $y_i \in \{0,1\}^{\lfloor \log n \rfloor + 1}$ is the binary representation of i, possibly containing leading zeros. Define $L = \{a^i(b^z y_i) \mid n \geq i \geq 1\}$.

The tree language L is recognized by an SDTA $B = (Q, \Sigma, \delta, F)$, where $Q = \{q_1, \ldots, q_n\}$, $F = \{q_1\}$, $\delta(a, q_i) = b^z \cdot y_i + q_{i+1}$, when $1 \leq i \leq n-1$, and, $\delta(a, q_n) = b^z \cdot y_n$.[1]

Clearly the bottom-up computations of B recognize the tree language L and it remains to estimate the size of the DFA with output H_a^B that defines transitions at a-labeled nodes. The DFA needs $z+1$ states to process the prefix b^z and at most $2^{(2+\lfloor \log n \rfloor)} - 1$ states to remember the suffix of length $1 + \lfloor \log n \rfloor$ and output the correct vertical state using the λ-function. Thus, B can be constructed with $z + 4n$ horizontal states.

Consider an arbitrary DTA(DFA) $B' = (Q', \Sigma, \delta', F')$ accepting L. First using Lemma 4, we see that $|Q'| \geq n$. Choose $R = \{a(b^z y_i) \mid 1 \leq i \leq n\} \cup \{a(b^{z+1})\}$. Clearly for any $t_1, t_2 \in R$ there exists $t \in T_\Sigma[x]$ such that $t(x \leftarrow t_1) \in L$ if and only if $t(x \leftarrow t_2) \notin L$. Note that $a(b^{z+1})$ corresponds to the dead state and $t(x \rightarrow a(b^{z+1})) \notin L$ for any $t \in T_\Sigma[x]$.

Denote by p_i the state that B' assigns to the root of $a(b^z y_i)$, $1 \leq i \leq n$. It is easy

[1]As explained at the beginning of the current chapter, we use notation where an alphabet symbol σ occurring in strings of a horizontal language is interpreted as a leaf node labeled by σ. We use $\cdot$ and $+$ to denote concatenation and union in regular expressions.

to verify (as in Lemma 4) that $p_i \neq p_j$ when $i \neq j$. Now $\delta'(p_i, a) \cap \Sigma^* = \{b^z y_i\}$. (If $\delta'(p_i, a)$ were to contain some other string over Σ, B' would accept trees not in L.) Thus the DFA $H^{B'}_{p_i,a}$ recognizing $\delta'(p_i, a)$ has at least $|b^z y_i| + 1 = z + \lfloor \log n \rfloor + 2$ states.

Since the above holds for all $1 \leq i \leq n$, a lower bound for the numbers of vertical and horizontal states of B' is size$(B') \geq [\ n;\ n \cdot (\lfloor \log n \rfloor + 2 + z)\ ]$. ∎

Using Lemma 21 with $z = n - \lfloor \log n \rfloor$, we see that the upper bound of Lemma 20 is tight within a multiplicative constant. This is stated as:

Theorem 22. *An SDTA with n vertical and m horizontal states can be simulated by a DTA(DFA) having n vertical and $n \cdot m$ horizontal states.*

For $n \geq 1$, there exists a tree language L_n recognized by an SDTA with n vertical and $O(n)$ horizontal states such that any DTA(DFA) recognizing L_n has n vertical and $\Omega(n^2)$ horizontal states.

It can be viewed as expected that in the conversion of Theorem 22 the number of vertical states does not change. However as discussed in the previous chapter, in general, for a DTA(DFA) it may be possible to reduce the number of horizontal states by increasing the number of vertical states. As observed in [35], this is the reason why a DTA(DFA) with a minimal total number of states is not unique.

4.1.2 Converting a DTA(DFA) to an SDTA

We now consider the equivalent SDTA of a DTA(DFA). Again we give first an upper bound for the construction. It is known from ([35] Proposition 24) that the equivalent SDTA does not increase the number of vertical states.

Lemma 23. *Let $B = (Q, \Sigma, \delta, F)$ be an arbitrary DTA(DFA), where $|Q| = n$. Let $H^B_{q,\sigma} = (S_{q,\sigma}, Q, \gamma_{q,\sigma}, s^0_{q,\sigma}, F_{q,\sigma})$ be a DFA for the horizontal language $\delta(q, \sigma)$, $q \in Q$, $\sigma \in \Sigma$.*

We can construct an equivalent SDTA B' where

$$\text{size}(B') \leq \left[\; |Q|; \quad \sum_{\sigma\in\Sigma}(\prod_{q\in Q}(|S_{q,\sigma}| - |F_{q,\sigma}|) + \sum_{q\in Q}|F_{q,\sigma}| \cdot \prod_{p\in Q, p\neq q}(|S_{p,\sigma}| - |F_{p,\sigma}|)) \;\right].$$

Proof. We construct an equivalent SDTA $B' = (Q, \Sigma, \delta', F)$ as follows. Denote $Q = \{q_1, \ldots, q_n\}$. For each $\sigma \in \Sigma$, we define a DFA with output

$$H^{B'}_\sigma = (\prod_{q\in Q} S_{q,\sigma}, Q, \Delta_\sigma, (s^0_{q_1,\sigma}, \ldots, s^0_{q_n,\sigma}), E_\sigma, \lambda_\sigma),$$

where for $r_i \in S_{q_i,\sigma}$, $1 \leq i \leq n$,

$$\Delta_\sigma((r_1, r_2, \ldots, r_n), q) = (\gamma_{q_1,\sigma}(r_1, q), \ldots, \gamma_{q_n,\sigma}(r_n, q)),$$

$$\lambda_\sigma((r_1, \ldots, r_n)) = \begin{cases} q_j \text{ if } \min\{k \mid r_k \in F_{q_k,\sigma}\} = j \geq 1, \\ \text{undefined, otherwise;} \end{cases}$$

and E_σ consists of elements of $\prod_{q\in Q} S_{q,\sigma}$ for which λ_σ is defined. The output function λ_σ assigns to a tuple $(r_1, \ldots, r_n) \in E_\sigma$ the vertical state q_j where j is the smallest index such that r_j is a final state of the DFA $H^B_{q_j,\sigma}$. The choice may seem arbitrary, however, the construction works because, since B is a DTA(DFA) the horizontal languages $\delta(q_{j_1}, \sigma)$ and $\delta(q_{j_2}, \sigma)$, $j_1 \neq j_2$, are always disjoint and hence in any tuple $(r_1, \ldots, r_n) \in \prod_{q\in Q} S_{q,\sigma}$ that the computation of $H^{B'}_\sigma$ may actually reach, at most one of the components r_i can be a final state of the corresponding horizontal DFA $H^B_{q_i,\sigma}$.

Above we have noted that computations of $H^{B'}_\sigma$ use as states only tuples $(r_1, \ldots, r_n)$ where either, for all $1 \leq j \leq n$, $r_j \in S_{q_j,\sigma} - F_{q_j,\sigma}$, or there exists exactly one $1 \leq j \leq n$, such that $r_j \in F_{q_j,\sigma}$. Eliminating the unnecessary tuples and taking the sum over all $\sigma \in \Sigma$, gives for the total number of horizontal states of B', $\sum_{\sigma\in\Sigma} \text{size}(H_\sigma^{B'})$, the upper bound claimed in the statement of the lemma. The number of vertical states of B' is n. ■

If B has m horizontal states, Lemma 23 gives for the number of horizontal states of B' a worst-case upper bound that is less than 2^m but is not polynomial in m.

Next we give a lower bound construction. It can be noted that also [34] contains a claim that a minimal SDTA may be exponentially larger than an equivalent DTA(DFA) (see the discussion on Proposition 10.29 of [34] on page 211).

Lemma 24. *Let* $\Sigma = \{a, b, 0, 1\}$. *For any* $m \in \mathbb{N}$ *and relatively prime numbers* $2 \leq k_1 < k_2 < \ldots < k_m$, *there exists a tree language* L *over* Σ *recognized by a DTA(DFA)* B *with* $\text{size}(B) = [\ m;\ \ \sum_{i=1}^{m} k_i + O(m \log m)\ ]$ *such that any SDTA recognizing* L *has at least* m *vertical states and* $\Pi_{i=1}^{m} k_i$ *horizontal states.*

Proof. Let $y_i \in \{0, 1\}^*$ be the binary representation of $i \geq 1$. Choose $L = \bigcup_{1\leq i\leq m} a^i((b^{k_i})^* y_i)$.

We define for L a DTA(DFA) $B = (Q, \Sigma, \delta, F)$, where $Q = \{q_1, \ldots, q_m\}$, $F = \{q_1\}$, $\delta(a, q_i) = (b^{k_i})^* \cdot y_i + q_{i+1}$, for $1 \leq i \leq m-1$, and $\delta(a, q_m) = (b^{k_m})^* \cdot y_m$. Note that the bottom-up computation of B is deterministic because different horizontal languages are marked by distinct binary strings y_i.

Each horizontal language $(b^{k_i})^* \cdot y_i + q_{i+1}$ can be recognized by a DFA with $k_i + \lfloor \log i \rfloor + 3$ states, and in total B has $\sum_{i=1}^{m} k_i + \sum_{i=1}^{m}(\lfloor \log i \rfloor) + 3m$ horizontal states (and m vertical states).

Let $B' = (Q', \Sigma, \delta', F')$ be an arbitrary SDTA recognizing L. By choosing $R = \{a(b^{k_i} y_i) \mid 1 \leq i \leq m\} \cup \{a(b)\}$, Lemma 4 gives $|Q'| \geq m$.

We show that the DFA $H_a^{B'}$, with notations as in (2.1), defining transitions corresponding to symbol a needs at least $\prod_{i=1}^{m} k_i$ states. Suppose that $H_a^{B'}$ has fewer than $\prod_{i=1}^{m} k_i$ states. Then there exist $0 \leq j < s < \prod_{i=1}^{m} k_i$ such that $H_a^{B'}$ reaches the same state after reading strings b^j and b^s, respectively. There must exist $1 \leq r \leq m$ such that k_r does not divide $s - j$. Let $z = j + ((k_r - j) \bmod k_r)$. Since $H_a^{B'}$ reaches the same state on b^j and b^s, it follows that $H_a^{B'}$ reaches the same state also on $b^z \cdot y_r$ and $b^{z+s-j} \cdot y_r$, respectively. This means that $a^{k_r}(b^z y_r)$ is accepted by B' if and only if $a^{k_r}(b^{z+s-j} \cdot y_r)$ is accepted by B', which is a contradiction because k_r divides z and does not divide $z + s - j$. ■

In the above proof, using a more detailed analysis it could be shown that $H_a^{B'}$ needs $\Omega(m \cdot \log m)$ additional states to process the strings y_i, however, this would not change the worst-case lower bound.

Now we establish that the upper and lower bounds for the DTA(DFA)-to-SDTA conversion are within a multiplicative constant, at least when the sizes of the horizontal DFAs are large compared to the number of vertical states.

Theorem 25. *An arbitrary DTA(DFA) $B = (Q, \Sigma, \delta, F)$ has an equivalent SDTA B' with*

$$\text{size}(B') \leq \left[\,|Q|;\ \sum_{\sigma \in \Sigma} \prod_{q \in Q} \text{size}(H^B_{q,\sigma}) \right], \tag{4.2}$$

and, for an arbitrary $m \geq 1$ there exists a DTA(DFA) $B = (Q, \Sigma, \delta, F)$ with $|Q| = m$ such that for any equivalent SDTA B' the size of B' has a lower bound within a multiplicative constant of (4.2).

Proof. The upper bound follows from Lemma 23. We get the lower bound from Lemma 24 by choosing each k_i to be at least $m \cdot \log m$, $i = 1, \ldots, m$. ■

We note that when converting a DTA(DFA) $B = (Q, \Sigma, \delta, F)$ to an equivalent SDTA A, for each $\sigma \in \Sigma$ the horizontal DFA H_σ^A needs at least as many states as a DFA recognizing $L_{B,\sigma} = \bigcup_{q \in Q} \delta(q, \sigma)$. Note that from H_σ^A we obtain a DFA for $L_{B,\sigma}$ simply by ignoring the output function. However, H_σ^A needs to provide more detailed information for a given input string than a DFA simply recognizing $L_{B,\sigma}$, and in fact H_σ^A recognizes the marked union, as formalized below, of the languages $\delta(q, \sigma)$.

We say that a DFA $A = (Q, \Sigma, \gamma, s_0, F)$ equipped with an output function $\lambda : F \to \{1, \ldots, m\}$ recognizes the *marked union* of pairwise disjoint regular languages $L_1, \ldots, L_m$, if $L_i = \{w \in \Sigma^* \mid \lambda(\gamma(s_0, w)) = i\}$, $i = 1, \ldots, m$. The following result establishes that the state complexity of marked union may be arbitrarily much larger than the state complexity of union.

Proposition 26. *Let* $A = (Q, \Sigma, \gamma, s_0, F, \lambda)$ *be a DFA with output function* $\lambda : F \to \{1, \ldots m\}$ *that recognizes the marked union of disjoint languages* L_i*,* $i = 1, \ldots, m$*, and let* B *be the minimal DFA for* $\bigcup_{i=1}^m L_i$*.*

Then $\text{size}(A) \geq \text{size}(B)$*, and for any* $m \geq 1$ *there exist disjoint regular languages* L_i*,* $1 \leq i \leq m$*, such that* $\text{size}(B) = 1$ *and the* $\text{size}(A) \geq m$*.*

Proof. The inequality $\text{size}(A) \geq \text{size}(B)$ follows from the observation that we obtain B from A simply by ignoring the output function.

Let L_i, $i = 1, \ldots, m$, be regular languages that form a partition of Σ^*. The union of the languages L_i, $1 \leq i \leq m$, can be recognized by a DFA with one state but a DFA recognizing their marked union needs at least as many states as the number of components. ■

4.2 Converting nondeterministic tree automata to deterministic automata

In this section we consider conversions of different variants of nondeterministic automata into equivalent strongly deterministic automata and DTA(DFA)s.

4.2.1 Converting a nondeterministic automaton to an SDTA

When determinizing an NTA(NFA) A with set of vertical states Q, the set of vertical states becomes $\mathcal{P}(Q)$ as in the usual subset construction. This means that the horizontal languages will be defined over subsets of Q and a horizontal DFA H^B_σ associated with $\sigma \in \Sigma$ has to determine for a string $P_1 \cdots P_m$, $P_i \in \mathcal{P}(Q)$, $1 \leq i \leq m$, all possible states of Q that A may reach at a node labeled with σ and having m children, when the ith child is labeled by some state in P_i, $1 \leq i \leq m$.

Lemma 27. *Let $A = (Q, \Sigma, \delta, F)$ be an NTA(NFA) and for $q \in Q$, $\sigma \in \Sigma$ denote* $\text{size}(H^A_{q,\sigma}) = m_{q,\sigma}$.

(i) *We can construct an equivalent SDTA B where*

$$\text{size}(B) \leq \left[2^{|Q|};\ \sum_{\sigma \in \Sigma} 2^{\left(\sum\limits_{q \in Q} m_{q,\sigma}\right)} \right]. \tag{4.3}$$

(ii) *If A is a DTA(NFA), in the upper bound (4.3) the number of vertical states is at most $|Q|$.*

Proof. First we consider the case (i) where A is an arbitrary NTA(NFA). Denote $Q = \{q_1, \ldots, q_n\}$ and $H^A_{q_i,a} = (C_{a,i}, Q, \gamma_{a,i}, q^0_{a,i}, F_{a,i})$ is the horizontal NFA corresponding to $q_i \in Q$ and $a \in \Sigma$.

We define an SDTA $B = (\mathcal{P}(Q), \Sigma, \eta, F_B)$ where $F_B = \{X \subseteq Q \mid X \cap F \neq \emptyset\}$ and η-transitions corresponding to $a \in \Sigma$ are determined by a DFA with output function:

$$H_a^B = (\mathcal{P}(C_{a,1}) \times \cdots \times \mathcal{P}(C_{a,n}), \mathcal{P}(Q), \mu_a, (\{q_{a,1}^0\}, \ldots, \{q_{a,n}^0\}), E_a, \lambda_a), \qquad (4.4)$$

where μ_a and λ_a are defined below, and E_a consists of all tuples $(Y_1, \ldots, Y_n)$, $Y_i \subseteq C_{a,i}$, $i = 1, \ldots, n$, such that $\lambda_a(Y_1, \ldots, Y_n) \neq \emptyset$. Note that since B uses $\mathcal{P}(Q)$ as the set of states, this is also the input alphabet for H_a^B. For $X \subseteq Q$ and $Y_i \subseteq C_{a,i}$, $i = 1, \ldots, n$,

$$\mu_a((Y_1, \ldots, Y_n), X) = (\bigcup_{x \in X} \gamma_{a,1}(Y_1, x), \ldots, \bigcup_{x \in X} \gamma_{a,n}(Y_n, x)).$$

Here $\gamma_{a,i}(Y_i, x)$ stands for $\bigcup_{z \in Y_i} \gamma_{a,i}(z, x)$. For $Y_i \subseteq C_{a,i}$, $i = 1, \ldots, n$,

$$\lambda_a((Y_1, \ldots, Y_n)) = \{q_i \mid Y_i \cap F_{a,i} \neq \emptyset\}.$$

The computation of H_a^B on a string $w = w_1 \cdots w_k$, $w_j \in \mathcal{P}(Q)$, $j = 1, \ldots, k$, roughly speaking, simulates the computation of each NFA $H_{q_i,a}^A$, $1 \leq i \leq n$, on each string $u = u_1 \cdots u_k \in Q^*$, $u_i \in w_i$, $i = 1, \ldots, k$. With above notations, we say that $u \in Q^*$ is a *projection* of the string $w \in (\mathcal{P}(Q))^*$. Assume that the computation of H_a^B on w reaches a state $(Y_1, \ldots, Y_n)$, $Y_i \subseteq C_{a,i}$, $i = 1, \ldots, n$. Then λ_a maps $(Y_1, \ldots, Y_n)$ to the set P $(\subseteq Q)$ consisting of exactly those elements $q \in Q$ such that $H_{q,a}^A$ accepts some projection of the string w. These conditions guarantee that if the nondeterministic computation of A can reach the children of a node v labeled by a in states determined by the string $w \in (\mathcal{P}(Q))^*$, the possible states at node v are exactly the states of P. Note that this property relies strongly on the fact that computations in subtrees corresponding to different children of v are independent. It follows that $L(B) = L(A)$. The SDTA B has 2^n vertical states and $\sum_{a \in \Sigma} 2^{\sum_{i=1}^n |C_{a,i}|}$ horizontal states.

(ii) Now assume that A is a DTA(NFA). Analogously, as in (i) above we define an equivalent SDTA $B = (Q, \Sigma, \eta, F_B)$. Now for the horizontal DFA H_a^B the input alphabet is just Q. However, because the automata $H_{q_i,a}^A$ remain nondeterministic the set of states of H_a^B is, in general, as in (4.4) and the upper bound for the total number of horizontal states is the same as in (i). ■

We do not require the deterministic tree automata to be complete, that is, for a sequence of states labeling children of a node u, the automaton need not associate any state to u. This means that in the subset construction we can naturally omit the empty set and in (4.3) the number of vertical states of B could be reduced to $2^{|Q|} - 1$. A similar small improvement could be made to the number of horizontal states, but it would make the formula look rather complicated.

Also, in Lemma 27 (ii) the upper bound for the number of horizontal states could be slightly reduced using a more detailed analysis, as in the proof of Lemma 23, taking into account that, in no situation, two distinct NFAs defining the horizontal languages associated with a fixed input symbol σ can accept simultaneously.

Lemma 27 did not discuss the case where the bottom-up computation is nondeterministic but the horizontal languages are represented in terms of DFAs. We note that for an NTA(DFA) $A = (Q, \Sigma, \delta, F)$ the construction used in the proof of Lemma 27 gives for the size of an equivalent SDTA only the upper bound (4.3). Although the horizontal languages of A are defined using DFAs, the horizontal languages of the equivalent SDTA B are over the alphabet $\mathcal{P}(Q)$, and this means that, at least when using the straightforward construction, the upper bound for the number of horizontal states would not be improved. With the notation used in the proof, when reading a string w in $\mathcal{P}(Q)^*$, for each $q_i \in Q$, the horizontal DFA H_a^B would need to keep track of a subset of states that the DFA $H_{q_i,a}^A$ can reach on any projection of w.

Next we state two lower bound results.

Lemma 28. *Let $\Sigma = \{a, b\}$. For any relatively prime numbers $m_1, m_2, ..., m_n$, there exists a tree language L over Σ such that L is recognized by an NTA(DFA) A with* $\text{size}(A) \leq [\ n;\ (\sum_{i=1}^{n} m_i) + 2n - 2\ ]$, *and any SDTA for L needs at least $2^n - 1$ vertical states and $(\prod_{i=1}^{n} m_i) - 1$ horizontal states.*

Proof. We choose $L = \{a^i((b^{m_i})^*) \mid 1 \leq i \leq n\}$.

The tree language L is accepted by an NTA(DFA) $A = (Q, \{a, b\}, \delta, \{q_1\})$, where $Q = \{q_1, q_2, ..., q_n\}$, $\delta(a, q_i) = (b^{m_i})^* + q_{i+1}$, $1 \leq i \leq n-1$, $\delta(a, q_n) = (b^{m_n})^*$. Each horizontal language $\delta(a, q_i)$ can be recognized by a DFA with $m_i + 2$ states, $1 \leq i \leq n-1$, and $\delta(a, q_n)$ is recognized by a DFA with m_n states. Note that since A is an NTA(DFA) there is no requirement that different horizontal languages associated with a would need to be disjoint.

Let $B = (Q', \Sigma, \delta', F')$ be an arbitrary SDTA recognizing L. For $r \subseteq \{1, \ldots, n\}$, define $s_r = a(b^{\Pi_{i \in r} m_i})$. Denote

$$R = \{s_r \mid \emptyset \neq r \subseteq \{1, \ldots, n\}\ \} \cup \{a(b^{(\Pi_{i=1}^{n} m_i)+1})\}.$$

We show that R satisfies the conditions of Lemma 4. First consider any two distinct nonempty sets $r_1, r_2 \subseteq \{1, \ldots, n\}$. Choose $k \in r_1 - r_2$. The other case where $r_1 - r_2 \neq \emptyset$ is completely symmetric. Now for $t = a^{k-1}(x) \in T_\Sigma[x]$, $t(x \leftarrow s_{r_1}) \in L$ and $t(x \leftarrow s_{r_2}) \notin L$.

Second, for any $\emptyset \neq r$ there exists $t \in T_\Sigma[x]$ such that $t(x \leftarrow s_r) \in L$. On the other hand, for any $t \in T_\Sigma[x]$, $t(x \leftarrow a(b^{(\Pi_{i=1}^{n} m_i)+1}) \notin L$ because no m_i, $1 \leq i \leq n$, can divide $(\Pi_{i=1}^{n} m_i) + 1$.

Thus, we have verified that the set R satisfies the conditions of the statement of

Lemma 4. Since $|R| = 2^n$ it follows that B needs at least $2^n - 1$ vertical states.

It remains to establish the lower bound for the number of horizontal states of B. Let $K = \Pi_{i=1}^{n} m_i$ and define $S = \{b^j \mid 1 \leq j \leq K\}$. [2] Consider any distinct integers $1 \leq x < y \leq K$. Since all the m_j's are pairwise relatively prime, there exists $1 \leq i \leq n$ such that m_i does not divide $y - x$. Choose $0 \leq z < m_i$ such that $y + z \equiv 0 \pmod{m_i}$. Also let $t = a^{i-1}[x] \in T_\Sigma[x]$. Now $t(x \leftarrow a(b^{y+z})) \in L$ and $t(x \leftarrow a(b^{x+z})) \notin L$. Note that because m_i divides $y+z$ and m_i does not divide $y - x$, m_i does not divide $x + z$. According to Lemma 5, B needs at least $|S| - 1$ horizontal states. ■

Lemma 29. *For each $n \geq 1$, there exists a tree language L_n recognized by a DTA(NFA) A with n vertical and $O(n)$ horizontal states such that for any SDTA B for L_n,* size$(B) \geq [\ n;\ 2^n\ ]$.

Proof. Choose $\Sigma = \{a, b, c\}$. The construction is, essentially, based on using an NFA of size n for which the minimal equivalent DFA has 2^n states.

Let L_{Moore} be the language defined from Example 9 Define $T = \{c^n(w) \mid w \in L_{\text{Moore}}\}$. The tree language T is recognized by a DTA(NFA) $A = (Q, \Sigma, \delta, \{q_1\})$, where $Q = \{q_1, q_2, \ldots, q_n\}$, $\delta(c, q_i) = \{q_{i+1}\}$, $1 \leq i \leq n-1$, $\delta(c, q_n) = L_{\text{Moore}}$. Each $\delta(c, q_i)$ can be recognized by an NFA with 2 states, $1 \leq i \leq n-1$, and from Example 9 we know that $\delta(c, q_n) = L_{\text{Moore}}$ is recognized by an NFA with n states.

Let $B = \{Q', \Sigma, \delta', F'\}$ be an arbitrary SDTA recognizing T. Define $R = \{c^i(a^{n-1}) \mid 1 \leq i \leq n-1\} \cup \{c(b)\}$. Let $t \in T_\Sigma[x]$ be arbitrary. For $1 \leq i \leq n-1$, $t(x \leftarrow c(a^{n-1})) \in T$ if and only if $t = c^{n-i}(x)$, and $t(x \leftarrow c(b))$ is never in T. Thus, R satisfies the conditions of Lemma 4, and it follows that $|Q'| \geq n$.

[2]To be consistent with the notation of Lemma 5, we view the elements of S as tuples of trees each having one node.

Let $H_c^B = (S_c, Q', s_c^0, F_c, \gamma_c, \lambda_c)$ be the DFA with output function that determines the transitions of B at symbol c.[3] Denote $P = \{q \in Q' \mid B \text{ accepts } c^{n-1}(q)\}$. Define $E = (S_c, Q', s_c^0, \lambda_c^{-1}(P), \gamma_c)$. Now $L(E) = L_0$ and we know by Example 9 that $\text{size}(E) \geq 2^n$, which means that also H_c^B has at least 2^n states. ■

The lower bounds given by Lemma 28 and 29 for the resulting number of horizontal states when converting an NTA(DFA) or a DTA(NFA), respectively, into an SDTA is far away from the corresponding upper bound given in Lemma 27. Furthermore, even for general NTA(NFA)s we do not have a worst-case construction that provably would give an essentially better lower bound than the one obtained for NTA(DFA)s in Lemma 28.

4.2.2 Converting a nondeterministic automaton to a DTA(DFA)

Lemma 30. *Consider an NTA(NFA) $A = (Q, \Sigma, \delta, F)$. For $q \in Q$, $\sigma \in \Sigma$ we denote* $\text{size}(H_{q,\sigma}^A) = m_{q,\sigma}$.

(i) *There exists a DTA(DFA) B equivalent to A where*

$$\text{size}(B) \leq \left[2^{|Q|};\ 2^{|Q|} \cdot \left(\sum_{\sigma \in \Sigma} 2^{(\Sigma_{q \in Q} m_{q,\sigma})}\right) \right]. \tag{4.5}$$

(ii) *If A is a DTA(NFA), it has an equivalent DTA(DFA) B where*

$$\text{size}(B) \leq \left[|Q|;\ \sum_{q \in Q} \sum_{\sigma \in \Sigma} 2^{m_{q,\sigma}} \right].$$

Proof. Let $Q = \{q_1, \ldots, q_n\}$ and as usual denote by $H_{q,\sigma}^A = (S_{q,\sigma}, Q, \gamma_{q,\sigma}, s_{q,\sigma}^0, F_{q,\sigma})$ the horizontal DFA corresponding to $q \in Q$ and $\sigma \in \Sigma$. We note that the construction below has similarities with the proof of Lemma 27. At first sight it might seem that

[3] Recall that according to our notational conventions elements of Σ are used also as states of Q'.

while in Lemma 27 the horizontal automaton of the SDTA needed to simulate all horizontal automata corresponding to the same input symbol, here for a horizontal DFA associated with $P \subseteq Q$ it would be sufficient to simulate only the horizontal NFAs of A that are associated with states in P. However, in order to guarantee the disjointness of the horizontal languages in the current construction the horizontal DFA associated with P needs to simulate also the horizontal NFAs associated with states in $Q - P$. The difference in the upper bound is caused by the fact that a DTA(DFA) needs a separate horizontal automaton for each state.

We use the following notation. For $P = \{q_{i_1}, \ldots, q_{i_m}\} \subseteq Q$, $1 \le i_1 < i_2 < \cdots < i_m \le n$, we denote by $I_P = \{i_1, \ldots, i_m\}$ the *index set* of P. We define a DTA(DFA) $B = (\mathcal{P}(Q), \Sigma, \eta, F_B)$ where $F_B = \{X \subseteq Q \mid X \cap F \neq \emptyset\}$ and for $P \subseteq Q$ and $\sigma \in \Sigma$, the horizontal language $\eta(P, \sigma)$ is recognized by a DFA

$$H^B_{P,\sigma} = (\mathcal{P}(S_{q_1,\sigma}) \times \cdots \times \mathcal{P}(S_{q_n,\sigma}),\ \mathcal{P}(Q), \beta_\sigma,\ (s^0_{q_1,\sigma}, \ldots, s^0_{q_n,\sigma}), \mathcal{H}_P),$$

where the set of final states is

$$\mathcal{H}_P = \{(X_1, \ldots, X_n) \ \mid \ X_i \subseteq S_{q_i,\sigma}, i = 1, \ldots, m, (\forall i \in I_P)\ X_i \cap F_{p_i,\sigma} \neq \emptyset$$
$$\text{and } (\forall i \in \{1, \ldots, n\} - I_P)\ X_i \cap F_{p_i,\sigma} = \emptyset\},$$

and the transitions are defined by setting for $X_i \subseteq S_{q_i,\sigma}$, $1 \le i \le n$, $Y \subseteq Q$,

$$\beta_\sigma((X_1, \ldots, X_n), Y) = (\bigcup_{y \in Y} \gamma_{q_1,\sigma}(X_1, y), \ldots, \bigcup_{y \in Y} \gamma_{q_n,\sigma}(X_m, y)).$$

The above construction and the argument justifying that B correctly simulates A are similar to ones used in the proof of Lemma 27.

We note here just the following. In the above construction $H^B_{P,\sigma}$ simulates, besides the NFAs $H^A_{q,\sigma}$, $q \in P$, also the computation of each NFA $H^A_{q,\sigma}$, $q \in Q - P$. This is necessary,[4] in order to guarantee that the bottom-up computation of B is deterministic, i.e., that all horizontal languages $\eta(P, \sigma)$, $P \subseteq Q$, are pairwise disjoint. Note that $H^B_{P,\sigma}$ accepts the strings $w \in (\mathcal{P}(Q))^*$ such that the set of all states that A can reach at a node labeled σ with leaves labeled by some projection[5] of w is exactly P. This means that automatically $L(H^B_{P_1,\sigma}) \cap L(H^B_{P_2,\sigma}) = \emptyset$, for $P_1 \neq P_2$, $\sigma \in \Sigma$.

In fact, we can note that the definition of the DFA $H^B_{P,\sigma}$ depends on P only in the choice of the set of final states $\mathcal{H}_P$ and the sets of final states corresponding to distinct sets P_1 and P_2 are disjoint.

The above construction of the DFAs $H^B_{P,\sigma}$, $P \subseteq Q$, $\sigma \in \Sigma$, gives the upper bound for the number of horizontal states in (i).

Finally, the upper bound for (ii) follows from the observation that it is sufficient determinize the NFA $H^A_{q,\sigma}$ separately for each $q \in Q$ and $\sigma \in \Sigma$. ■

Roughly speaking, the simulation uses a standard subset construction [64] for the set of vertical states, and in order to guarantee that the bottom-up computation remains deterministic the DFA for the horizontal language corresponding to $P \subseteq Q$, $\sigma \in \Sigma$, needs to simulate each horizontal NFA of A corresponding to σ. In the case where A is an NTA(DFA) we do not have a significantly better upper bound than (4.5), because the horizontal languages of the DTA(DFA) will consist of strings of subsets of Q, which means that we again have to simulate multiple computations of each horizontal DFA of A. In the lower bound construction of Theorem 31 below we, in fact, use an NTA(DFA).

We do not have a lower bound that would match the bound of Lemma 30. Recall that strongly deterministic automata can be minimized efficiently and the minimal

[4]That is, at least there seems to be no obvious way to avoid simulating all the NFAs $H^A_{q,\sigma}$.
[5]A projection of a string over $\mathcal{P}(Q)$ is defined as in the proof of Lemma 27.

automaton is unique [10], however, minimal DTA(DFA)s are, in general, not unique and minimization is intractable [35]. When trying to establish lower bounds for the size of a DTA(DFA) $A = (Q, \Sigma, \delta, F)$ there is the difficulty that by adding more vertical states, and hence more horizontal languages, it may still be possible that the total number of horizontal states is reduced, which was illustrated in Example 11.

Below in Theorem 31 we give a lower bound construction for the NTA(DFA) to DTA(DFA) transformation that relies on particular types of unary horizontal languages that are constructed in a way that prevents a trade-off between vertical and horizontal states. After that we give a more general lower bound that relies on Lemma 7. The lower bound result of Theorem 31 below relies on an ad hoc proof.

Let $\Sigma = \{a, b\}$ and let $p_1, \ldots, p_n$ be the first n primes, $n \geq 1$. Define the tree language

$$T_n = \{a^i(b^k) \mid i \geq 1,\ k \geq 0,\ (\exists 1 \leq j \leq n)[k \equiv 0 \pmod{p_j} \text{ and } i \equiv j \pmod{n}]\}.$$

Theorem 31. *The tree language T_n can be recognized by an NTA(DFA) A with*

$$\text{size}(A) = \left[\ n;\ (\sum_{i=1}^{n} p_i) + 2n\ \right],$$

and for any DTA(DFA) B recognizing T_n,

$$\text{size}(B) \geq \left[\ 2^n - 1;\ (2^n - 1) \cdot \prod_{i=1}^{n} p_i\ \right].$$

Proof. The tree language T_n can be recognized by an NTA(DFA) $A = (Q, \{a, b\}, \delta, \{q_1\})$, where $Q = \{q_1, q_2, ..., q_n\}$, $\delta(a, q_i) = (b^{p_i})^* + q_{i+1}$, $1 \leq i \leq n-1$, $\delta(a, q_n) = (b^{p_n})^* + q_1$. Clearly B accepts only trees of the form $a^i(b^k)$, $i \geq 1$, $k \geq 0$. On a tree $a^i(b^k)$, B can nondeterministically assign a state q_j, $1 \leq j \leq n$, to the node of height one provided that k is a multiple of p_j. After this the vertical computation checks deterministically that $j \equiv i \pmod{n}$.

The NTA(DFA) B has n vertical states and each language $\delta(a, q_i)$ can be recognized by a DFA with $p_i + 2$ states, $i = 1, \ldots, n$.

Let $B = (R, \Sigma, \delta_B, R_F)$ be an arbitrary DTA(DFA) for T_n. Recall that for a Σ-tree t we denote by t^B the state of B reached at the root of t.

We establish a lower bound for the number of vertical and of horizontal states of B. It would be fairly easy to establish directly that $|R| \geq 2^n - 1$. We derive this as a consequence of the more general Claim 32 that is useful for a lower bound for the total number of horizontal states.

Define $H_1 = \{a(b^m) \mid m \geq 0\}$. For $t = a(b^m) \in H_1$ we define

$$\mathrm{PRIMES}_t = \{1 \leq j \leq n \mid m \equiv 0 \pmod{p_j}\} \ \ (\subseteq \{1, \ldots, n\}).$$

Then for $S \subseteq \{1, \ldots, n\}$ we define

$$\mathrm{TREES}_S = \{t \in H_1 \mid \mathrm{PRIMES}_t = S\}.$$

TREES_S consists of elements of H_1 where the number of leaves labeled by b is divided by exactly those p_j's where $j \in S$. Furthermore, we define

$$(\mathrm{TREES}_S)_B = \{t^B \mid t \in \mathrm{TREES}_S\}.$$

$(\mathrm{TREES}_S)_B$ consists of states that B reaches at roots of elements of TREES_S.

Claim 32. *For any $S_1, S_2 \subseteq \{1, \ldots, n\}$, $S_1 \neq S_2$, we have*

$$(\mathrm{TREES}_{S_1})_B \cap (\mathrm{TREES}_{S_2})_B = \emptyset.$$

Proof of the claim. For the sake of contradiction assume that

$$r \in (\mathrm{TREES}_{S_1})_B \cap (\mathrm{TREES}_{S_2})_B. \tag{4.6}$$

Without loss of generality, we can choose $j \in S_1 - S_2$. The other possibility where $S_2 - S_1 \neq \emptyset$ is symmetric.

By (8.2), there exist $t_i = a(b^{m_i}) \in \text{TREES}_{S_i}$, $i = 1, 2$, such that $(t_1)^B = (t_2)^B = r$. By the definition of the sets TREES_{S_i} it follows that $m_1 \equiv 0 \pmod{p_j}$ and $m_2 \not\equiv 0 \pmod{p_j}$.

Choose $u = a^{j-1}(x) \in T_\Sigma[x]$. Since B reaches the same state at the roots of t_1 and t_2, respectively, it follows that

$$u(x \leftarrow t_1) \text{ is accepted by } B \text{ if and only if } u(x \leftarrow t_2) \text{ is accepted by } B.$$

We have derived a contradiction because $u(x \leftarrow t_1) \in T_n$ and $u(x \leftarrow t_2) \notin T_n$. This concludes the proof of the claim.

As a consequence of Claim 32 it follows that R contains $2^n - 1$ nonempty disjoint sets of states, each of which consists of states reached at the root of elements of TREES_S for some $\emptyset \neq S \subseteq \{1, \ldots, n\}$. In particular, $|R| \geq 2^n - 1$.

For $S \subseteq \{1, \ldots, n\}$ define the unary language

$$\text{UNARY}_S = \{b^m \mid a(b^m) \in \text{TREES}_S\}.$$

By Claim 32,

$$\text{UNARY}_S = \bigcup_{r \in (\text{TREES}_S)_B} \delta_B(r, a). \tag{4.7}$$

If $(\text{TREES}_S)_B = \{r_0\}$ is a singleton set, the minimal DFA C for the horizontal language $\delta_B(r_0, a) = \text{UNARY}_S$ has exactly $\Pi_{i=1}^n p_i$ states. Note that on an input b^m, the DFA C has to verify for each p_i, $1 \leq i \leq n$, that p_i divides m if and only if $i \in S$.

More generally, UNARY_S can be a finite union of horizontal languages $\delta_B(r, a)$, where $r \in (\text{TREES}_S)_B$ as in (8.3). Now we apply Lemma 10 to UNARY_S by selecting (in the notations of Lemma 10) $R_1 = S$ and $R_2 = \{1, \ldots, n\} - S$. Since UNARY_S is infinite, we know that one of the languages $\delta_B(r_1, a)$, $r_1 \in (\text{TREES}_S)_B$ has to be infinite, and by Lemma 10, the minimal DFA for $\delta_B(r_1, a)$ has a cycle of length at

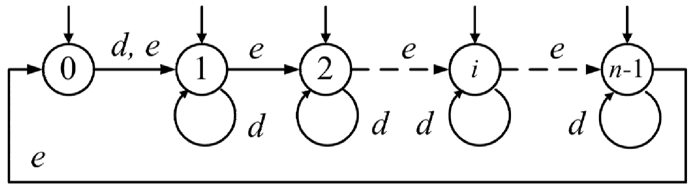

Figure 4.1: The NFA M recognizing L_{Leung}

least $\Pi_{i=1}^{n} p_i$.

In all cases, for any nonempty set S ($\subseteq \{1, \ldots, n\}$) the DFAs for the horizontal languages $\delta_B(r, a)$, $r \in (\text{TREES}_S)_B$, have in total at least $\Pi_{i=1}^{n} p_i$ states. Since S is an arbitrary nonempty subset of $\{1, \ldots, n\}$, we have established the required lower bound for the number of horizontal states of any DTA(DFA) for T_n. ■

Theorem 31 gives a construction where converting an NTA(DFA) to a DTA(DFA) causes an exponential blow-up in the number of vertical states, and additionally the size of each of the (exponentially many) horizontal DFAs is considerably larger than the original DFA. However, the size blow-up of the horizontal DFAs does not match the upper bound of Lemma 30. In the proof of Theorem 31, roughly speaking, we use a particular type of unary horizontal language in order to be able to (provably) establish that there cannot be a trade-off between the numbers of vertical and horizontal states, and with this type of construction it seems difficult to approach the worst-case size blow-up of Lemma 30.

Next we give another lower bound result for converting an NTA(NFA) to a DTA(DFA) that relies on Lemma 7.

Let L_{Leung} be the language recognized by the NFA M shown in Figure 4.1. The final states in M consists of all the odd numbered states, i.e. $\{1, 3, 5, \ldots\}$. Define an

NTA(NFA)

$$A = (\{q_1, \ldots, q_m, q_d, q_e\}, \{a, b, c, d, e\}, \delta_A, \{q_m\}), \tag{4.8}$$

where

- $\delta_A(q_d, d) = \epsilon, \quad \delta_A(q_e, e) = \epsilon, \quad \delta_A(q_1, c) = w^A$, where $w \in L_{Leung}$
- $\delta_A(q_1, b) = q_1, \quad \delta_A(q_1, a) = q_m, \quad \delta_A(q_2, a) = q_1 + q_m$
- $\delta_A(q_{i+1}, b) = q_i$ for $2 \leq i \leq m-1$
- $\delta_A(q_{i+1}, a) = q_i$ for $2 \leq i \leq m-1$

In the tree language $L(A)$, symbol c always labels a node of height one. The leaves below c spell out a word recognized by the NFA of Figure 4.1. On top of the node labeled by c there is a unary branch belonging to the language L_{Moore}. L_{Moore} can be recognized by an NFA shown in Figure 2.6. In this case the NFA has m states instead of n.

Theorem 33. *There exists a tree language L that is recognized by an NTA(NFA) with $m+2$ vertical and n+$4m$ horizontal states such that any equivalent DTA(DFA) $B = (P, \{a, b, c, d, e\}, \delta_B, F_B)$ needs at least 2^m vertical and $2^n - 1$ horizontal states.*

Proof. We choose $L = L(A)$ where A is as in (4.8). It is obvious that c is a *unique height-one label* of $L(A)$. According to Lemma 7, the total size of all horizontal DFAs associated with c in B ($\sum_{p\in P} |H^B_{p,c}|$) is greater or equal than the size of the smallest unambiguous NFA for the horizontal language $\delta_A(q_1, c)$, which is equal to L_{Leung} when we identify symbol e (respectively, d) with state q_e (respectively, q_d). Using Theorem 8, Leung [32] shows that any unambiguous NFA recognizing L_{Leung} has at least $2^n - 1$ states. This means the number of the horizontal states of B is at least $2^n - 1$, and this number cannot be reduced by introducing state trade-offs.

The vertical branch above the node labeled by c belongs to the language L_{Moore}. Since any DFA for L_{Moore} has at least 2^m states, where m is the number of states of the minimal NFA recognizing L_{Moore}, any deterministic tree automaton recognizing $L(A)$ has at least 2^m vertical states. ■

The lower bound above gives an exponential blow-up on both the number of vertical states and horizontal states. Furthermore, the exponential blow-up on the number of horizontal states cannot be reduced by any state trade-offs.

Chapter 5

Operations on tree languages

In this chapter, we extend the operations on strings to tree languages. Some operations such as union, intersection and complementation are similarly performed on trees as on strings. Thus, here we give the definitions of the operations that cannot obviously be extended to tree languages, such as concatenation and Kleene-star.

In the following chapters we will study the state complexity of operations on tree languages. Interestingly it turns out that the state complexity of many basic operations on trees is essentially different than the corresponding operational state complexity results for regular string languages.

An operation such as concatenation of strings can be extended either as a sequential or a parallel operation to trees, and the same holds naturally for iterated concatenation (or Kleene-star). In this chapter we develop a systematic notation for some basic operations on trees, such as concatenation, iterated concatenation and quotient. Variants of tree concatenation (or substitution) operations have been used in different places in the literature [14].

When defining concatenation and iterated concatenation, for the sake of convenience our notations use ranked trees (partly because for these operations the main state complexity results can also be formulated in terms of ranked trees). The definitions are extended in the obvious way for unranked trees.

5.1 Concatenation and iterated concatenation of tree languages

Concatenation of strings can be extended to trees as a sequential operation, where one occurrence of a leaf with a given label is replaced by a tree, or as a parallel operation, where all occurrences of a leaf with a given label are replaced.

For $\sigma \in \Sigma_0$ and $T_1 \subseteq F_\Sigma$, $t_2 \in F_\Sigma$, we define their *sequential σ-concatenation*

$$T_1 \cdot_\sigma^s t_2 = \{ \ t_2(u \leftarrow t_1) \mid u \in \text{leaf}(t_2, \sigma), t_1 \in T_1 \ \}. \tag{5.1}$$

That is, $T_1 \cdot_\sigma^s t_2$ is the set of trees obtained from t_2 by replacing one occurrence of a leaf labeled by σ with some tree of T_1. In order to get concatenation of individual trees we can choose T_1 as a singleton set.[1]

The *parallel σ-concatenation* of T_1 and t_2 is

$$T_1 \cdot_\sigma^p t_2 = t_2(\text{leaf}(t_2, \sigma) \leftarrow T_1). \tag{5.2}$$

Note that when $T_1 = \{t_1\}$ consists of one tree, $t_1 \cdot_\sigma^p t_2$ is an individual tree while $t_1 \cdot_\sigma^s t_2$ is a set of trees. In the case where no leaf of t_2 is labeled by σ, $t_1 \cdot_\sigma^s t_2 = \emptyset$ and $t_1 \cdot_\sigma^p t_2 = t_2$. The example below illustrates the difference between parallel and sequential concatenation.

[1]The first argument in (5.1) and (5.2) is a set of trees because otherwise the extension of parallel concatenation to tree languages would be somewhat cumbersome.

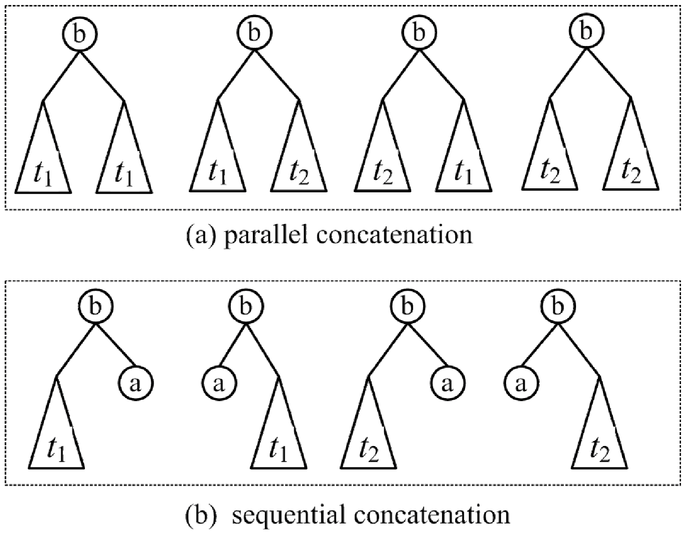

Figure 5.1: Parallel and sequential concatenation for t_0 and T_1

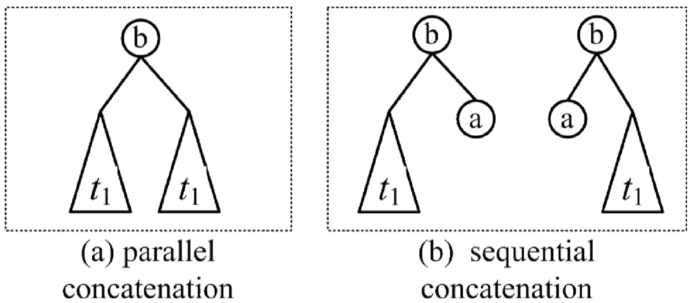

Figure 5.2: Parallel and sequential concatenation for t_0 and T_1'

Example 34. Let $\Sigma = \{a, b\}$, $t_0 = b(a, a)$ and $T_1 = \{t_1, t_2\} \subset F_\Sigma$. The resulting trees of $T_1 \cdot^p_a t_0$ and $T_1 \cdot^s_a t_0$ are shown in Figure 5.1 (a) and (b), respectively. Note that if $T_1' = \{t_1\}$ is a singleton, $T_1' \cdot^p_a t_0$ is an individual tree and $T_1' \cdot^s_a t_0$ is a set of two trees. The resulting trees are shown in Figure 5.2 (a) and (b), respectively.

In the natural way we extend $\circ \in \{\cdot^s_\sigma, \cdot^p_\sigma\}$ for tree languages $T_1, T_2 \subseteq F_\Sigma$ by setting

$$T_1 \circ T_2 = \bigcup_{t_2 \in T_2} T_1 \circ t_2.$$

The parallel concatenation $T_1 \cdot^p_\sigma T_2$ is called the σ-product of T_1 and T_2 in [14]. When

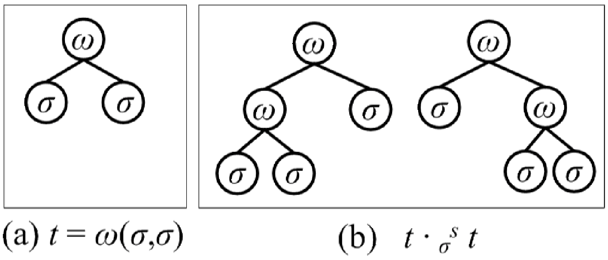

Figure 5.3: $t = \omega(\sigma, \sigma)$ in (a) and sequential concatenation $t \cdot_\sigma^s t$ in (b)

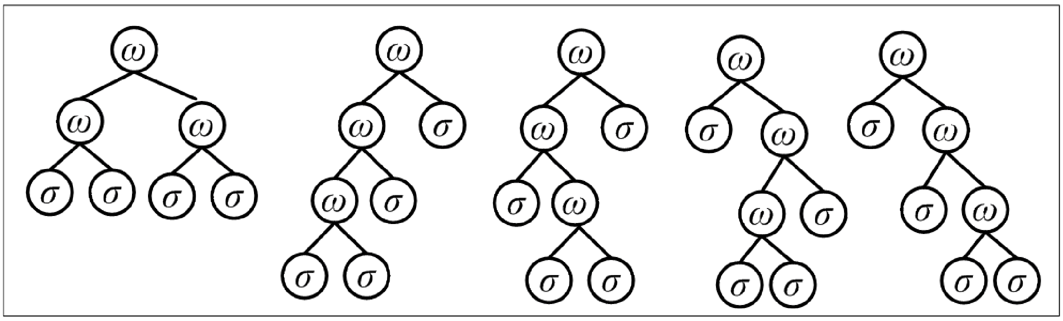

Figure 5.4: Trees in $t \cdot_\sigma^s (t \cdot_\sigma^s t)$

considering bottom-up tree automata operating on unary trees, the above definition of $T_1 \circ T_2$ reduces to the usual concatenation of string languages: the automaton reads first an element of T_1 and then an element of T_2.

The parallel concatenation operation is associative, however, sequential concatenation is non-associative.

Example 35. It is easy to see that sequential concatenation is non-associative. Consider a ranked alphabet Σ determined by $\Sigma_2 = \{\omega\}$, $\Sigma_0 = \{\sigma\}$ and let $t = \omega(\sigma, \sigma)$ which is shown in Figure 5.3 (a). Now $t \cdot_\sigma^s t = \{\omega(\omega(\sigma, \sigma), \sigma), \omega(\sigma, \omega(\sigma, \sigma))\}$ which is shown in Figure 5.3 (b) and $t_1 = \omega(\omega(\sigma, \sigma), \omega(\sigma, \sigma)) \in t \cdot_\sigma^s (t \cdot_\sigma^s t)$ but, on the other hand, $t_1 \notin (t \cdot_\sigma^s t) \cdot_\sigma^s t$. All trees in $(t \cdot_\sigma^s t) \cdot_\sigma^s t$ have height three. The resulting trees of $t \cdot_\sigma^s (t \cdot_\sigma^s t)$ and $(t \cdot_\sigma^s t) \cdot_\sigma^s t$ are shown in Figure 5.4 and Figure 5.5, respectively.

Finally, we note that instead of (sequential or parallel) concatenation where we

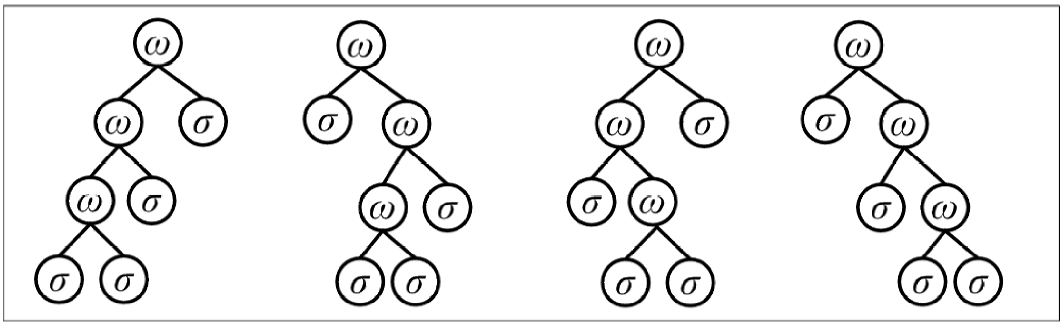

Figure 5.5: Trees in $(t \cdot_\sigma^s t) \cdot_\sigma^s t$

replace occurrences of $\sigma \in \Sigma_0$, we could define a more general substitution operation where a subtree with root labeled by $\sigma \in \Sigma_m$, $m \geq 0$, can be replaced by another tree. In the parallel case the selected nodes labeled by $\sigma \in \Sigma_m$ should be independent, and there is more than one way to define the parallel operation. The state complexity of such generalized operations is the same as the state complexity of tree concatenation considered here; see Remark 39 in Section 6.1.

5.1.1 Iterated concatenation of trees

We can define powers of a tree language based on sequential or parallel concatenation in the natural way, and then define a Kleene-star operation by setting T_σ^* to be the infinite union of all the ith powers ($i \geq 0$) of T. Since sequential concatenation is non-associative, as observed in Example 35, there will be two different ways to define an iterated version of sequential concatenation.

For $\sigma \in \Sigma$ and $T \subseteq F_\Sigma$, we define the *kth sequential top-down σ-power of T*, $k \geq 0$, by setting $T_\sigma^{s,t,0} = \{\sigma\}$, and $T_\sigma^{s,t,k} = T \cdot_\sigma^s T_\sigma^{s,t,k-1}$, when $k \geq 1$. The *sequential top-down σ-star of T* is then

$$T_\sigma^{s,t,*} = \bigcup_{k \geq 0} T_\sigma^{s,t,k}.$$

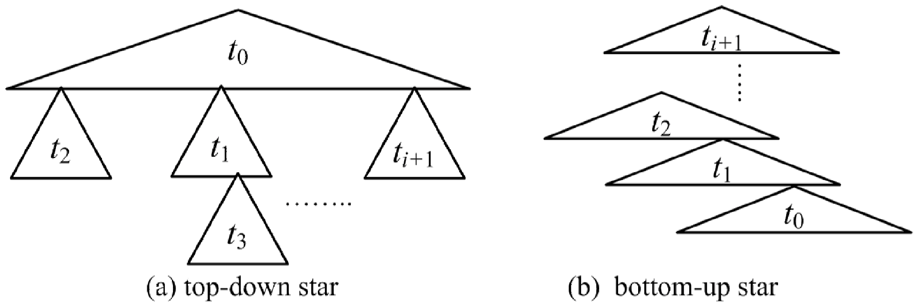

Figure 5.6: A tree in $T_\sigma^{s,t,*}$ (a) and in $T_\sigma^{s,b,*}$ (b). Here $t_0, t_1, \ldots t_{i+1}$ are trees in T.

Similarly, the *kth sequential bottom-up σ-power of T*, is defined by setting $T_\sigma^{s,b,0} = \{\sigma\}$, $T_\sigma^{s,b,1} = T$ and $T_\sigma^{s,b,k} = T_\sigma^{s,b,k-1} \cdot_\sigma^s T$, when $k \geq 2$. The *sequential bottom-up σ-star of* T is

$$T_\sigma^{s,b,*} = \bigcup_{k\geq 0} T_\sigma^{s,b,k}.$$

Note that the definition of bottom-up σ-powers explicitly sets $T_\sigma^{s,b,1}$ to be equal to T. This is done because $T_\sigma^{s,b,0} \cdot_\sigma^s T$ can be a strict subset of T if some trees of T contain no occurrences of σ. Figure 5.6 illustrates the definitions of top-down star and bottom-up star.

To illustrate the difference of top-down and bottom-up star, respectively, consider a ranked alphabet Σ determined by $\Sigma_2 = \{\omega\}$, $\Sigma_0 = \{\sigma\}$ and let $T = \{\omega(\sigma, \sigma)\}$. We note that $T_\sigma^{s,t,*} = F_\Sigma$ and

$$T_\sigma^{s,b,*} = \{r \in F_\Sigma \mid \text{ each non-leaf node of } r \text{ has at least one leaf as a child } \}.$$

Note that with $T = \{\omega(\sigma, \sigma)\}$, $T_\sigma^{s,b,k}$, $k \geq 0$, consists of trees of height (exactly) k. The trees of $T_\sigma^{s,b,*}$ all consist of a path labeled by binary symbols ω and all children of nodes of the path that "diverge" from the path are labeled by the leaf symbol σ.

The following characterization of bottom-up σ-star as the smallest set closed under concatenation with T from the right follows directly from the definition of bottom-up star. The characterization will be used in the next section.

Lemma 36. *For $\sigma \in \Sigma_0$ and $T \subseteq F_\Sigma$, define* $\mathrm{cl}_\sigma(T)$ *as the smallest set $S \subseteq F_\Sigma$ such that (i) $T \cup \{\sigma\} \subseteq S$, and (ii) $t_1 \cdot_\sigma^s t_2 \in S$ for every $t_2 \in T$ and $t_1 \in S$. Then* $\mathrm{cl}_\sigma(T) = T^{s,b,*}$.

Completely analogously we can define, for $T \subseteq F_\Sigma$, the parallel σ-star of T, denoted $T_\sigma^{p,*}$. Since parallel concatenation is associative, we do not need to distinguish the bottom-up and top-down variants. However, we note that with $T = \{\omega(\sigma,\sigma)\}$, $T_\sigma^{p,*}$ consists of all balanced trees over the ranked alphabet Σ, where $\Sigma_2 = \{\omega\}$, $\Sigma_0 = \{\sigma\}$.

Since iterated parallel concatenation does not preserve regularity, we will consider only the sequential variant of iterated concatenation. The top-down (respectively, bottom-up) σ-powers and σ-star of a tree language T are in the following denoted $T_\sigma^{t,k}$, $(k \geq 0)$, and $T_\sigma^{t,*}$ (respectively, $T_\sigma^{b,k}$ and $T_\sigma^{b,*}$), that is, we drop the superscript "s" in the notation.

Finally, we note that it would be possible to define a regularity-preserving iterated version of the parallel concatenation, by defining the kth $(k \geq 1)$ power of T by parallel-concatenating T with the union of all the ith powers of T, $0 \leq i \leq k-1$. It is easy to verify that this definition of a parallel star operation would coincide with the sequential top-down star defined above.

Here the definitions are on ranked trees. Concatenation and Kleene-star can be defined on unranked trees in exactly the same way except that we do not consider the rank of the alphabet.

5.2 Projection

To begin with we recall the projection operation on strings as studied in [28, 63].

For an alphabet Σ, $\Sigma_P \subseteq \Sigma$, and a string $w = \sigma_0\sigma_1 \ldots \sigma_n \in \Sigma^*$, a natural projection P is defined as: $P_{\Sigma\to\Sigma_P}(w) = \sigma'_0\sigma'_1 \ldots \sigma'_n$ such that

$$\sigma'_i = \begin{cases} \sigma_i, & \text{if } \sigma_i \in \Sigma_P; \\ \epsilon, & \text{otherwise.} \end{cases}$$

Now we define projection operation on trees.

For alphabets Σ and $\Sigma_P \subseteq \Sigma$, all Σ-labeled (Σ_P-labeled respectively) trees are denoted T_Σ (T_{Σ_P} respectively). Symbols in Σ/Σ_P are called unobservable symbols. A projection on trees is a mapping $\mathcal{M}_{\Sigma\to\Sigma_P}$: $T_\Sigma \to T_{\Sigma_P}$. When Σ and Σ_P are understood from the context, we denote $\mathcal{M}_{\Sigma\to\Sigma_P}$ simply by $\mathcal{M}$. $\mathcal{M}$ is inductively defined as:

- for a leaf node u labeled by σ,

$$\mathcal{M}(u) = \begin{cases} u, & \text{if } \sigma \in \Sigma_P; \\ \epsilon, & \text{otherwise.} \end{cases}$$

- for a tree $t = \sigma(t_0, t_1, \ldots, t_n)$,

$$\mathcal{M}(t) = \begin{cases} \sigma(\mathcal{M}(t_0), \mathcal{M}(t_1), \ldots, \mathcal{M}(t_n)), & \text{if } \sigma \in \Sigma_P; \\ \epsilon, & \text{otherwise.} \end{cases}$$

By an (ϵ, σ)-substitution, $\sigma \in \Sigma$, we mean a (finite) substitution that maps σ to $\{\epsilon, \sigma\}$ and maps each $\gamma \neq \sigma$ to itself. Furthermore, by an (ϵ, Δ)-substitution, $\Delta \subseteq \Sigma$, we mean a (finite) substitution that maps each $\sigma \in \Delta$ to $\{\epsilon, \sigma\}$ and maps each $\gamma \notin \Delta$ to itself. The result of applying the (ϵ, Δ)-substitution on a string s is denoted as $\xi_\Delta(s)$. The (ϵ, Δ)-substitution is extended to a regular language L as:

$$\xi_\Delta(L) = \{\xi_\Delta(s) \mid s \in L\}.$$

We noted that it is also possible to define the projection on trees in another way as follows, denoted by $\mathcal{M}'_{\Sigma\to\Sigma_P}$.

- for a leaf node u labeled by σ,

$$\mathcal{M}'(u) = \begin{cases} u, & \text{if } \sigma \in \Sigma_P; \\ \epsilon, & \text{otherwise.} \end{cases}$$

- for a tree $t = \sigma(t_0, t_1, \ldots, t_n)$,

$$\mathcal{M}'(t) = \begin{cases} \sigma(\mathcal{M}'(t_0), \mathcal{M}'(t_1), \ldots, \mathcal{M}'(t_n)), & \text{if } \sigma \in \Sigma_P; \\ \mathcal{M}'(t_0), \mathcal{M}'(t_1), \ldots, \mathcal{M}'(t_n), & \text{otherwise.} \end{cases}$$

When the root node of t is labeled by a symbol that is not in Σ_P, by the definition of $\mathcal{M}'_{\Sigma\to\Sigma_P}$, the root node labeled by σ of t is deleted and the children of t are kept as a sequence of trees $\mathcal{M}'(t_0), \mathcal{M}'(t_1), \ldots, \mathcal{M}'(t_n)$. This sequence of trees become the children of t's parent, and are listed between the left sibling and the right sibling of t. However, this definition does not preserve regularity as shown in Example 37.

Example 37. *Consider* $\Sigma = \{a, b, c, d\}$*,* $\Sigma_P = \{a, b, d\}$*, and* $T = \bigcup_{i\geq 0} f_i$ *is inductively defined as,*

- $t_0 = c(a, c, b)$*,*
- *for* $i \geq 1$*,* $t_{i+1} = c(a, t_i, b)$*,*
- *for* $i \geq 0$*,* $f_i = d(t_i)$*.*

We get $\mathcal{M}'_{\Sigma\to\Sigma_P}(T) = \{d(a^i, b^i) \mid i \geq 1\}$*, which is not regular.*

5.3 Quotient

Concatenation on unranked trees can be exactly defined as on ranked trees except that we do not consider the rank of the alphabet. The operation of concatenation is already defined in detail in Section 5.1. Here we just briefly define the sequential concatenation on unranked trees which will be used to define quotient operations on unranked trees.

For $t, t' \in T_\Sigma$ and $\sigma \in \Sigma$, we denote by $t \cdot_\sigma^s t'$ the set of trees that are obtained from t' by replacing one leaf labeled by σ by the tree t. The σ-concatenation operation (sequential concatenation) is extended in the natural way to sets of trees L_1, L_2:

$$L_1 \cdot_\sigma^s L_2 = \bigcup_{t \in L_1, t' \in L_2} t \cdot_\sigma^s t'.$$

The quotient operation on unranked trees is defined as follows.

Definition 1. *Let $\sigma \in \Sigma$. The σ-top-quotient of a tree language T with respect to a tree language T' is defined as:*

$$T' \top_\sigma T = \{t \mid \exists t' \in T', t \cdot_\sigma^s t' \in T\}.$$

Definition 2. *Let $\sigma \in \Sigma$. The σ-bottom-quotient of a tree language T with respect to a tree language T' is defined as:*

$$T \bot_\sigma T' = \{t \mid \exists t' \in T', t' \cdot_\sigma^s t \in T\}.$$

When considering computations that process a tree in the bottom-up direction, the top-quotient can be viewed as an extension of right-quotient from strings to trees, and similarly, the bottom-quotient extends the left-quotient operation from strings to trees. When $\sigma \in \Sigma$ is understood from the context, we simply call the operations top-quotient and bottom-quotient and write $T' \top T$ (respectively $T \bot T'$) in place of

$T' \top_\sigma T$ (respectively $T \bot_\sigma T'$).

Chapter 6

State complexity of concatenation-like operations

The results of this chapter are summarized at the beginning of Sections 6.1 and 6.2. We begin here with some comments on why we are using automata on ranked trees, instead of the more general model of unranked tree automata. The main justification is that, while the notations needed for automata operating on ranked trees are much simpler, already using tree languages over a ranked alphabet we can construct worst-case examples that match the general upper bound for the number of vertical states for the sequential concatenation of unranked tree languages. In the case of the Kleene-star operations, the worst-case state complexity bounds for the numbers of vertical states can be reached using just binary trees. The bounds are of a different order of magnitude than the known state complexity of concatenation and Kleene-star of regular string languages.

The general upper bound construction for concatenation and Kleene-star of unranked tree languages is given in Appendix A. While the idea is similar to the one used in Lemma 38 and Lemma 44 in this chapter, the notations are considerably more complicated with unranked tree automata. On the other hand, establishing lower bounds for the sizes of horizontal DFAs in unranked tree automata is a challenging question, and a topic for further research.

6.1 State Complexity of Concatenation

As we have seen in Section 5.1, it is possible to extend the concatenation operation from strings to trees either as a sequential or parallel concatenation operation. In the sequential (respectively, parallel) concatenation of trees t_1 and t_2 one occurrence (respectively, all occurrences) of leaves of t_2 having a designated label are replaced by t_1. The operations were extended in the natural way for sets of trees. In order to keep the connection with string operations more transparent, we defined the substitution operation by replacing a leaf (or leaves) of t_2 by t_1. In the context of trees one could define more general substitutions where a node (or nodes) of t_2 with a given label are replaced by t_1, however, this would not change the worst-case state complexity bounds for the sequential or parallel substitutions, respectively.

We consider the state complexity of concatenation operations for regular tree languages, that is, the question how many states are sufficient, and necessary in the worst-case, to recognize the concatenation of tree languages recognized by deterministic bottom-up tree automata with m and n states, respectively. We give tight state complexity bounds both for sequential and parallel concatenation. Interestingly, the state complexity of sequential concatenation of tree languages turns out to be of a different order of magnitude than the corresponding bound for regular string languages. The results for parallel concatenation are similar to the string case.

6.1.1 State complexity of sequential concatenation

Note that for DTAs A_1 and A_2, the difficulty in constructing a DTA B for the (sequential) concatenation of A_1 and A_2 is caused by the fact that B has no way to "know" where a substitution may have occurred, and consequently B has to simulate multiple computations in its state. It turns out that for sequential concatenation of tree languages, the size blow-up of B differs by an order of magnitude from the known state complexity of concatenation of regular string languages [64].

First we give an upper bound for the state complexity of sequential concatenation. As mentioned at the begin of the chapter, for ease of presentation we restrict consideration to automata operating on ranked trees.

Lemma 38. *Let A_i be a DTA with m_i states, $i = 1, 2$. For $\sigma \in \Sigma_0$, the tree language $L(A_1) \cdot_\sigma^s L(A_2)$ can be recognized by a DTA with*

$$(m_2 + 1) \cdot (m_1 \cdot 2^{m_2} + 2^{m_2-1}) - 1$$

states.

Proof. Denote $A_i = (\Sigma, Q_i, Q_{i,F}, g_i)$, and let $Q'_i = Q_i \cup \{\text{dead}\}$, $i = 1, 2$. The symbol "dead" will be used to denote a simulated computation that is undefined[1]. Without loss of generality we assume that σ_{g_2} is defined. Note that otherwise trees of $L(A_2)$ cannot contain leaves labeled by σ and $L(A_1) \cdot_\sigma^s L(A_2) = \emptyset$.

We define $B = (\Sigma, Q_B, Q_{B,F}, g_B)$ where

$$Q_B = Q'_2 \times 2^{Q_2} \times Q'_1, \quad Q_{B,F} = \{q \in Q_B \mid \pi_2(q) \cap Q_{2,F} \neq \emptyset\},$$

and the transitions of g_B are determined below. For $\tau \in \Sigma_m$, $m \geq 0$, $q_1, \ldots, q_m \in Q_i$,

[1] We use a new symbol "dead" (instead of $\emptyset$) to make a more transparent distinction between components of B that are, respectively, a state or a set of states of A_i, $i = 1, 2$.

$i = 1, 2$, we denote

$$\overline{\tau_{g_i}(q_1, \ldots, q_m)} = \begin{cases} \tau_{g_i}(q_1, \ldots, q_m) & \text{if } \tau_{g_i}(q_1, \ldots, q_m) \text{ is defined,} \\ \text{dead,} & \text{otherwise.} \end{cases} \tag{6.1}$$

For $\tau \in \Sigma_0$, define

$$\tau_{g_B} = \begin{cases} (\overline{\tau_{g_2}}, \{\sigma_{g_2}\}, \tau_{g_1}) & \text{if } \tau_{g_1} \in Q_{1,F}, \\ (\overline{\tau_{g_2}}, \emptyset, \overline{\tau_{g_1}}) & \text{if } \tau_{g_2} \text{ or } \tau_{g_1} \text{ is defined, } \tau_{g_1} \notin Q_{1,F}, \\ \text{undefined,} & \text{if } \tau_{g_2} \text{ and } \tau_{g_1} \text{ are both undefined.} \end{cases} \tag{6.2}$$

For $\tau \in \Sigma_m$, $m \geq 1$, and $(p_i, P_i, q_i) \in Q_B$, $p_i \in Q'_2$, $P_i \subseteq Q_2$, $q_i \in Q'_1$, $i = 1, \ldots, m$, define

$$\tau_{g_B}((p_1, P_1, q_1), \ldots, (p_m, P_m, q_m)) \tag{6.3}$$

to be equal to

(i) $(\overline{\tau_{g_2}(p_1, \ldots, p_m)}, X, \tau_{g_1}(q_1, \ldots, q_m))$ if $\tau_{g_1}(q_1, \ldots, q_m) \in Q_{1,F}$, where

$$X = \bigcup_{i=1}^{m} \left(\bigcup_{x \in P_i} \tau_{g_2}(p_1, \ldots, p_{i-1}, x, p_{i+1}, \ldots, p_m) \right) \cup \{\sigma_{g_2}\}.$$

(ii) $(\overline{\tau_{g_2}(p_1, \ldots, p_m)}, Y, \overline{\tau_{g_1}(q_1, \ldots, q_m)})$ if $\tau_{g_1}(q_1, \ldots, q_m) \notin Q_{1,F}$ and $[\tau_{g_2}(p_1, \ldots, p_m)$ or $\tau_{g_1}(q_1, \ldots, q_m)$ is defined, or $Y \neq \emptyset]$. Here

$$Y = \bigcup_{i=1}^{m} \left(\bigcup_{x \in P_i} \tau_{g_2}(p_1, \ldots, p_{i-1}, x, p_{i+1}, \ldots, p_m) \right).$$

(iii) undefined, otherwise.

The computation of B operates as follows. The first component of the state simulates the computation of A_2, assuming that no σ-substitution has occurred below the current node. Similarly the third component of the state simulates the computation of A_1 on the current subtree.

Finally, the second component of the state of B consists of the set of states $S \subseteq Q_2$ that A_2 could be in assuming a σ-substitution has been done below the current node, that is, S consists of all states that A_2 would reach if exactly one subtree of the current node belonging to $L(A_1)$ is replaced by a leaf labeled by σ. Both rules (6.6) and (6.3) add the state σ_{g_2} to the second component exactly in case when the current subtree is in $L(A_1)$. The rules 6.3 (i) and (ii) simulate all computations where for exactly one $1 \leq i \leq m$ we take a state of A_2 corresponding to a computation where a σ-substitution was done below the current node and for all $j \neq i$ we take the state of A_2 that corresponds to a computation where no substitution has occurred.

Thus, by induction on the height of an input tree $t = \tau(t_1, \ldots, t_m)$ it follows that assuming that B reaches the root of t_i in a state (p_i, P_i, q_i) where P_i consists of all states that A_2 can reach assuming that in t_i exactly one subtree belonging to $L(A_1)$ would be replaced by a leaf labeled by σ (and p_i, q_i are as described above), then the second component of the state (6.3) again consists of all states that A_2 can reach at the root of t assuming exactly one subtree of t belonging to $L(A_1)$ had been replaced by the symbol σ.

A state of B is final exactly in case the second component contains a final state of A_2. This means that B accepts exactly the trees that are obtained from some tree of $L(A_2)$ by replacing exactly one σ-labeled leaf by a tree of $L(A_1)$.

We note that $|Q_B| = (m_2 + 1) \cdot 2^{m_2} \cdot (m_1 + 1)$, however, not all states of Q_B are reachable. According to the definition of g_B, a state (p, P, q) where $q \in Q_{1,F}$ and $\sigma_{g2} \notin P$ cannot be reached in any computation of B and, furthermore, the state $(\text{dead}, \emptyset, \text{dead})$ is omitted as the sink state. Thus, Q_B has (at least) $(m_2+1)\cdot 2^{m_2-1}+1$ unreachable states. Subtracting this number from $|Q_B|$ gives the upper bound for the size of B given in the statement of the lemma. ∎

Remark 39. *Suppose that instead of tree concatenation, where the substitutions occur only at leaves, we consider a more general sequential tree substitution $t_1 \diamond_\sigma^s t_2$ that substitutes in t_2 the subtree at some node labeled by $\sigma \in \Sigma_m$, $m \geq 0$, by the tree t_1. For $\sigma \in \Sigma_m$ and DTAs A_1 and A_2, the σ-substitution of $L(A_1)$ into $L(A_2)$, $L(A_1) \diamond_\sigma^s L(A_2)$, could be recognized by a DTA C with the same set of states as the DTA B in the proof of Lemma 38, however, the transitions of C would be defined slightly differently. In (6.6) and (6.3) (i) always when the third component is a final state of A_1, the transition would add to the second component all states that A_2 may reach at the root of an arbitrary tree with the root labeled by σ.*

On the other hand, since tree concatenation (as considered here) is a special case of σ-substitution, the state complexity lower bound established below applies also for σ-substitution, for an m-ary symbol σ. Hence the state complexity of σ-substitution for an m-ary symbol σ, $m \geq 0$, coincides with state complexity of tree concatenation.

Similarly, the upper bound construction of Theorem 43 could be modified, without changing the set of states, for a parallel substitution operation that replaces subtrees at nodes labeled by an m-ary symbol σ by another tree. There are various ways to define a parallel substitution operation and we leave the details to the interested reader.

The upper bound of Lemma 38 is of a different order of magnitude than the tight state complexity bound for concatenation of string languages [64], and it remains to be verified that there exists a worst-case example matching the upper bound.

For our lower bound construction we use tree languages consisting, roughly speaking, of trees where each branch belongs to the worst-case languages $L(A)$ and $L(B)$ for string concatenation [64] and, furthermore, the DFA A (or B, respectively) reaches

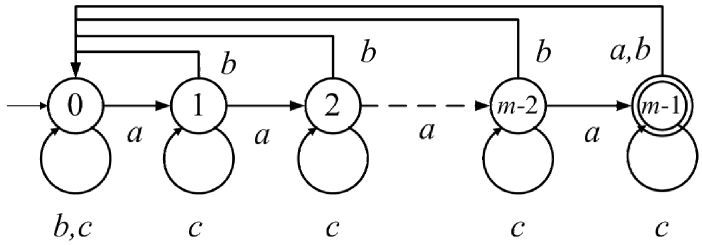

Figure 6.1: The DFA A from [64]

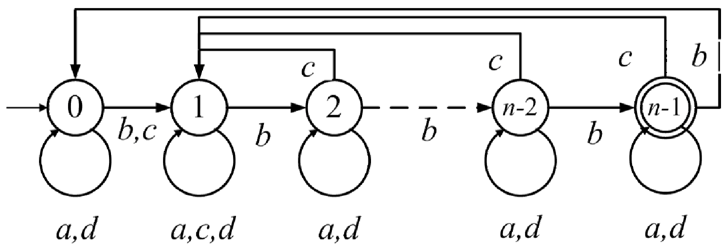

Figure 6.2: The DFA B from [64] with added d-transitions

the same state at an arbitrary node u in computations starting from any two leaves below u. Although the construction is based on the worst-case string languages, the extension is non-trivial and additional technical modifications are required in order to establish a lower bound matching the upper bound of Lemma 38.

Let A and B be the DFAs from Figures 6.1 and 6.2, respectively. Note that A and B are modified variants of the automata used for the worst-case construction for concatenation in [64]. In the DFA B we have added a new alphabet symbol d and a self-loop on d for each state of B. With the modified alphabet, A is an incomplete DFA where the d-transition is undefined in each state.

We choose ranked alphabet $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$ where $\Sigma_0 = \{e\}$, $\Sigma_1 = \{a, b, c, d\}$ and $\Sigma_2 = \{a_2, b_2, c_2, d_2\}$. Denote by h_Σ the morphism $(\Sigma_1 \cup \Sigma_2)^* \to \{a, b, c, d\}^*$ defined by the conditions $h_\Sigma(z_2) = h_\Sigma(z) = z$, $z \in \{a, b, c, d\}$, that is, the morphism h_Σ simply erases the subscript from elements of Σ_2.

Using the DFAs A and B of Figures 6.1 and 6.2 we define tree languages $T_A, T_B \subseteq F_\Sigma$. First, T_B is defined to consist of all Σ-trees t such that

(i) The following holds for any $u \in \mathrm{dom}(t)$ and any nodes v_1 and v_2 of height one below u. If w_i is the string of symbols labeling the path from v_i to u, $i = 1, 2$, then B reaches the same state after reading the strings $h_\Sigma(w_1)$ and $h_\Sigma(w_2)$. Furthermore, if $u = \varepsilon$, B accepts the strings $h_\Sigma(w_1)$ and $h_\Sigma(w_2)$.

(ii) Suppose that $u \in \mathrm{dom}(t)$ is labeled by a_2, b_2 or d_2 and u has a child that is a leaf (labeled by e) and another child u' that is not a leaf, and w is the string of symbols labeling a path of symbols from a node of height one below u' to u'. Then B reaches the state 0 after reading the input $h_\Sigma(w)$.

Intuitively, the condition (i) means that for a tree in T_B, when the DFA B reads strings of symbols (with subscripts omitted) labeling paths starting from nodes of height one upwards, the computations corresponding to different paths "agree" at each node, and the computations accept at the root. The technical condition (ii) is just used to simplify the definition of the DTA M_B below.

Note that the simulated computations of B on path of the tree are started from the nodes of height one and they ignore the leaf symbols. This is done for technical reasons because in tree concatenation a leaf symbol is replaced by a tree, i.e., the original symbol labeling the leaf will not appear in the resulting tree.

The tree language T_B is recognized by a DTA $M_B = (\Sigma, Q_B, Q_{B,F}, g_B)$ where $Q_B = \{0, 1, \ldots, n-1\}$ $Q_{B,F} = \{n-1\}$ and the transition function is defined by setting:

- $e_{g_B} = 0$,

- $a_{g_B}(i) = (a_2)_{g_B}(i,i) = i$, $0 \leq i \leq n-1$,
- $b_{g_B}(i) = (b_2)_{g_B}(i,i) = i+1$, $0 \leq i \leq n-2$, and $b_{g_B}(n-1) = (b_2)_{g_B}(n-1,n-1) = 0$.
- $c_{g_B}(i) = (c_2)_{g_B}(i,j) = 1$, $0 \leq i,j \leq n-1$,
- $d_{g_B}(i) = (d_2)_{g_B}(i,i) = i$, $0 \leq i \leq n-1$.

Note that the transitions of g_B on the binary symbol c_2 allow different values for the arguments while this is not the case in the transitions on a_2, b_2 and d_2. The reason is that the transition function of B on each of the symbols a, b, d is injective while the transition function on c is not. It is clear that M_B recognizes the tree language T_B defined previously. Note that the condition (ii) implies that for $t \in T_B$, if a node $u \in \text{dom}(t)$ is labeled by a_2, b_2 or d_2 and u has a child $u \cdot i$, $1 \leq i \leq 2$, that is a leaf, the computation of B started from a node of height one below $u \cdot j$, $j \neq i$, arrives at node $u \cdot j$ in state 0, which is the state assigned by M_B to the leaf node $u \cdot i$. Thus, the transitions of M_B for a symbol of rank two (labeling u) continue to correctly simulate the computation of B on one path of the tree.

The tree language T_A and a DTA M_A, with m states, recognizing T_A are defined completely analogously based on the DFA A from Figure 6.1. Note that since all d-transitions of A are undefined, trees of T_A have no nodes labeled by d or d_2 and all d- and d_2-transitions of M_A are undefined.

In the following we establish that the DTA constructed from M_A and M_B to recognize the sequential concatenation of T_A and T_B is minimal, and thus gives a worst-case example that matches the upper bound of Lemma 38. Let $M_C = (\Sigma, Q_C, Q_{C,F}, g_C)$ be the DTA for the tree language $T_A \cdot_e^s T_B$ constructed as in the proof of Lemma 38.

We make the convention that the "dead" state added to M_A (respectively, to M_B) is denoted by m (respectively, n). That is, the set of states Q_C consists of all triples

$$(q, S, p), \;\; 0 \leq q \leq n, S \subseteq \{0, 1, \ldots, n-1\}, 0 \leq p \leq m, \tag{6.4}$$

where if $p = m - 1$ then $0 \in S$, and if $S = \emptyset$ then $q \neq n$ or $p \neq m$.[2] The number of states of M_C is $(n+1)((m+1)2^n - 2^{n-1}) - 1$.

In the following two lemmas we establish that all states of M_C are reachable and pairwise inequivalent with respect to the Myhill-Nerode equivalence relation (extended to trees). We still introduce the following notation. For a unary tree where the leaf is an element of Σ_0 or a state of Q_C, $t = z_1(z_2(\ldots z_m(x)\ldots)) \in F_\Sigma[Q_C]$, we define $\text{word}(t) = z_m z_{m-1} \ldots z_1$. Note that $\text{word}(t)$ consists of the sequence of symbols labeling the nodes of t bottom-up, and the label of the leaf is not included. In the following when we refer to $\text{word}(t)$ of a tree t, without further mention, this implies that t is a unary tree (with the leaf possibly labeled by a state of Q_C).

Lemma 40. *All states of M_C are reachable.*

Proof. Using induction on $|S|$ we establish that all the states (6.4) are reachable. The DTA M_C assigns to a leaf symbol e the state $(0, \emptyset, 0)$. When $|S| = 0$, $(i, \emptyset, j)$, $0 \leq i \leq n-1$, $0 \leq j \leq m-2$ is reachable from $(0, \emptyset, 0)$ by reading a unary tree t where $\text{word}(t) = b^i a^j$. (Note that $(i, \emptyset, m-1)$ is not a state of Q_C.) The state $(n, \emptyset, j)$, $1 \leq j \leq m-2$, is reachable by reading tree $a_2(t_1, t_2)$ where $\text{word}(t_1) = ba^{j-1}$ and $\text{word}(t_2) = b^2 a^{j-1}$ and the leaves of t_1 and t_2 are labeled by $(0, \emptyset, 0)$. The state $(n, \emptyset, 0)$ is reached by reading a unary symbol b from state $(n, \emptyset, j)$, $1 \leq j \leq m-2$. State $(i, \emptyset, m)$, $0 \leq i \leq n-1$ is reached by a tree $b_2(t_1, t_2)$ where $\text{word}(t_1) = b^{i-1}a$ and $\text{word}(t_2) = b^{i-1}a^2$ and the leaves of t_1 and t_2 are labeled by $(0, \emptyset, 0)$.

[2] As at the end of the proof of Lemma 38, we have omitted from Q_C the unreachable states.

In the following, for an integer $x \geq -n$, denote

$$\overline{x} = \begin{cases} x \text{ if } x \geq 0 \\ n + x \text{ if } x < 0 \end{cases}$$

Consider $z \geq 0$ and inductively assume that for $|S| \leq z$, all the states (i, S, j) as in (6.4), $0 \leq i \leq n$, $0 \leq j \leq m$, $S \subseteq \{0, \ldots, n-1\}$ are reachable. We will show that any state (x, S', y), $0 \leq x \leq n$, $0 \leq y \leq m$, $|S'| = z + 1$ is reachable.

First consider the case where $y \neq m - 1$. Let $s_1 > s_2 > \ldots > s_z > s_{z+1}$ be the elements in S'. Let $P = \{s_1 - s_{z+1}, s_2 - s_{z+1}, \ldots, s_z - s_{z+1}\}$.

(i) When $0 \leq x \leq n-1$, according to the inductive assumption, the state $(\overline{x - s_{z+1}}, P, 0)$, is reachable. Then the state $(\overline{x - s_{z+1}}, P \cup \{0\}, m-1)$ is reachable from $(\overline{x - s_{z+1}}, P, 0)$ by reading a sequence of unary symbols a^{m-1}. The state (x, S', y), $0 \leq y \leq m-2$ is reachable from $(\overline{x - s_{z+1}}, P \cup \{0\}, m-1)$ by reading a sequence of unary symbols $b^{s_{z+1}} a^y$. The state (x, S', m) is reachable from $(\overline{x - s_{z+1}}, P \cup \{0\}, m-1)$ by reading a sequence of unary symbols $b^{s_{z+1}} d$.

(ii) When $x = n$, according to the inductive assumption, the state $(n, P, 0)$, is reachable. Then the state $(n, P \cup \{0\}, m-1)$ is reachable from $(n, P, 0)$ by reading a sequence of unary symbols a^{m-1}. The state (n, S', y), $0 \leq y \leq m-2$ is reachable from $(n, P \cup \{0\}, m-1)$ by reading a sequence of unary symbols $b^{s_{z+1}} a^y$. The state (n, S', m) is reachable from $(n, P \cup \{0\}, m-1)$ by reading a sequence of unary symbols $b^{s_{z+1}} d$.

Finally, consider the case when $y = m-1$. According to the definition of (6.4), $0 \in S'$. By the inductive assumption, the state $(x, S' - \{0\}, m-2)$ is reachable. Then the state $(x, S', m-1)$ is reached by reading the unary symbol a.

This concludes the proof of the inductive step and the proof of the lemma. ■

It remains to establish that the DTA M_C has no two equivalent states.

Lemma 41. *All states of M_C are pairwise inequivalent.*

Proof. Let (i_1, S_1, j_1) and (i_2, S_2, j_2) be any distinct states as in (6.4). First we consider the case when $S_1 \neq S_2$ or $j_1 \neq j_2$. We get from the DTA M_C a bottom-up tree automaton recognizing the restriction of $T_A \cdot_e^s T_B$ to unary trees simply by ignoring the first component of the states, and making all transitions undefined on elements of Σ_2. Note that for unary trees $t_1 \in T_A$, $t_2 \in T_B$, word$(t_1 \cdot_e^s t_2)$ is simply the string concatenation of word(t_1) and word(t_2).

Let B' be the DFA obtained from B (in Figure 6.2) by deleting all d transitions. In [64, 66] it is established that the minimal DFA for the concatenation of the string languages $L(A)$ and $L(B')$ needs $m2^n - 2^{n-1}$ states, which means that the elements (S, i), $S \subseteq \{0, \ldots, n-1\}$, $0 \leq i \leq m-1$, where $0 \in S$ always when $i = m-1$, correspond to states of a minimal DFA for $L(A)L(B')$, and also to states of a minimal DTA for $T_A \cdot_e^s T_B$ restricted to unary trees without occurrences of the symbol d. Note that in the construction of M_C, the unary transitions operate on the second and third components in the same way as in the DFA constructed in [64, 66] to recognize $L(A)L(B')$.

This means that when $(S_1, j_1) \neq (S_2, j_2)$ and $0 \leq j_1, j_2 \leq m-1$, the states (i_1, S_1, j_1) and (i_2, S_2, j_2) can be distinguished using a unary tree. Note that $j_i = m$ $(1 \leq i \leq 2)$ corresponds to a "dead" state of M_A, and this "dead" state does not occur in the construction of [64, 66], and we need to consider the cases $j_i = m$, $1 \leq i \leq 2$, separately.

First consider the case $j_1 = m$, $0 \leq j_2 \leq m-1$ (and S_1, S_2 may or may not be equal). Choose a sequence of unary symbols $ca^{m-j_2-1}b^{n-1}$. From state (i_2, S_2, j_2), state $(1, \{1\}, j_2)$ is reached after reading c, state $(1, \{1, 0\}, m - 1)$ is reached after reading a^{m-j_2-1}, and a final state $(0, \{0, n - 1\}, 0)$ is reached after reading b^{n-1}. On the other hand, from state (i_1, S_1, m), state $(1, \{1\}, m)$ is reached after reading c, state $(1, \{1\}, m)$ is reached after reading a^{m-j_2-1}, and state $(0, \{0\}, m)$ is reached after reading b^{n-1}. The latter is not a final state.

Next consider the case where $S_1 \neq S_2$ and $j_1 = j_2 = m$. Without loss of generality choose $s \in S_1 - S_2$ (the other possibility being symmetric). Choose a sequence of unary symbols $w = b^{n-1-s}$. After reading w, M_C reaches a final state when the computation begins from state (i_1, S_1, m) while the computation beginning with (i_2, S_2, m) does not reach a final state.

So far, we have showed that any two states (i_1, S_1, j_1) and (i_2, S_2, j_2) can be distinguished when $S_1 \neq S_2$ or $j_1 \neq j_2$, $0 \leq j_1, j_2 \leq m$. It remains to consider the case when $S_1 = S_2 = S$, $j_1 = j_2 = j$ and $i_1 \neq i_2$. Since $i_1 \neq i_2$, one of i_1, i_2 has to be distinct from n and, without loss of generality, we assume that $0 \leq i_1 \leq n - 1$, $0 \leq i_2 \leq n$. In order to establish that (i_1, S, j) and (i_2, S, j) are inequivalent, it is sufficient to give a tree $t \in F_{\Sigma'}[x]$ such that the computation of M_C on $t(x \leftarrow (i_1, S, j))$ (respectively, $t(x \leftarrow (i_2, S, j))$) reaches a final state (respectively, a non-final state). Here $\Sigma'_0 = \Sigma_0 \cup Q_C$ and $\Sigma'_k = \Sigma_k$, when $k \geq 1$. Above we use the fact that by Lemma 46 all states of Q_C are reachable.

Denote $q_u = (i_1+1, \{i_1\}, j)$ and as the tree $t \in F_{\Sigma'}[x]$ we choose $t = b^{2n-2-i_1} b_2(x, q_u)$.

First consider the computation of M_C on $t(x \leftarrow (i_1, S, j))$. Since the second component of q_u contains i_1 and the first components of q_u and (i_1, S, j) are different, the computation assigns $(n, \{i_1 + 1\}, 0)$ to the root of $b_2((i_1, S, j), q_u)$. After reading the remaining b's on the unary path, the final state $(n, \{n - 1\}, 0)$ is reached.

Now consider the computation on $t(x \leftarrow (i_2, S, j))$. Denote by (y, U, z) the state assigned to the root of $b_2((i_2, S, j), q_u)$. We note that $z = 0$ and

$$y = \begin{cases} (i_1 + 2) \pmod{n} \text{ if } i_2 = i_1 + 1 \\ \qquad\qquad\qquad \text{and } i_1 + 1 \neq n, \\ n \text{ otherwise} \end{cases} \quad U = \begin{cases} \{(i_1 + 2) \pmod{n}\} \text{ if } i_1 + 1 \in S, \\ \emptyset \text{ otherwise} \end{cases}$$

Above we use $x \pmod{n}$ as an element of $\{0, 1, \ldots, n-1\}$. Recall that the number of b's after the root of $b_2((i_2, S, j), q_u)$ (the binary symbol b_2 is not counted) to the root of $t(x \leftarrow (i_2, S, j))$ is $2n - i_1 - 2$.

(i) If $y = (i_1+2) \pmod{n}$ and $U = \{k\}$, $k = (i_1+2) \pmod{n}$, the computation beginning from state (y, U, z) after reading the sequence of unary symbols b^{2n-i_1-2} reaches the state $(0, \{0\}, 0)$.

(ii) If $y = n$, the first component of the resulting state will change to n, and if $U = \emptyset$, the second component of the resulting state will change to $\emptyset$.

In all cases the computation of M_C reaches a non-final state at the root of $t(x \leftarrow (i_2, S, j))$.

This concludes the proof showing that all the states of (6.4) are pairwise inequivalent. ∎

The following is now a consequence of Lemmas 38, 46 and 41.

Theorem 42. *Suppose that A_i is a DTA with m_i states, $i = 1, 2$, and $\sigma \in \Sigma_0$. The sequential σ-concatenation $L(A_1) \cdot_\sigma^s L(A_2)$ can be recognized by a DTA with*

$$(m_2 + 1) \cdot (m_1 \cdot 2^{m_2} + 2^{m_2 - 1}) - 1 \tag{6.5}$$

states.

For any integers $m_1, m_2 \geq 2$, there exist DTAs A_i with m_i states, $1 \leq i \leq 2$, such that the minimal DTA for $L(A_1) \cdot_\sigma^s L(A_2)$ has (8.3) states.

6.1.2 State complexity of parallel concatenation

In this section we give a tight state complexity bound for the parallel concatenation of tree languages. As can perhaps be expected, the bounds are similar as for regular string languages. We give a short construction for the upper bound because we are considering incomplete automata and the bounds differ slightly for complete and incomplete DFAs, respectively. The well known state complexity bounds for concatenation of string languages are stated in terms complete DFAs [27, 36, 66]. Transition complexity of incomplete DFAs has been considered in [13].

Theorem 43. *Let A_1 and A_2 be DTAs with m and n states, respectively ($m, n \geq 2$). For $\sigma \in \Sigma_0$, the tree language $L(A_1) \cdot_\sigma^p L(A_2)$ is recognized by a DTA with $m \cdot 2^n + 2^{n-1} - 1$ states and this bound can be reached in the worst-case.*

Proof. Denote $A_i = (\Sigma, Q_i, Q_{i,F}, g_i)$, $i = 1, 2$, and let $Q_1' = Q_1 \cup \{\text{dead}\}$. Without loss of generality σ_{g_2} is defined (because otherwise $L(A_1) \cdot_\sigma^p L(A_2) = L(A_2)$). We define $D = (\Sigma, Q_D, Q_{D,F}, g_D)$ where $Q_D = 2^{Q_2} \times Q_1'$, $Q_{D,F} = \{q \in Q_D \mid \pi_1(q) \cap Q_{2,F} \neq \emptyset\}$, and the transitions of g_D are determined below. For $\tau \in \Sigma_0$, define

$$\tau_{g_D} = \begin{cases} (\{\tau_{g_2}, \sigma_{g_2}\}, \tau_{g_1}) & \text{if } \tau_{g_1} \in Q_{1,F}, \\ (\{\tau_{g_2}\}, \overline{\tau_{g_1}}) & \text{if } \tau_{g_1} \notin Q_{1,F}, \text{ and at least one of } \tau_{g_1} \text{ and } \tau_{g_2} \text{ is defined,} \\ \text{undefined,} & \text{if } \tau_{g_1} \text{ and } \tau_{g_2} \text{ are both undefined.} \end{cases}$$

Above the overline notation is as in (6.1). When τ_{g_2} is undefined, $\{\tau_{g_2}, \sigma_{g_2}\} = \{\sigma_{g_2}\}$.

For $\tau \in \Sigma_k$, $k \geq 1$, and $(P_i, q_i) \in Q_D$, $i = 1, \ldots, k$, define

$$\tau_{g_D}((P_1, q_1), \ldots, (P_k, q_k)) = (\tau_{g_2}(P_1, \ldots, P_k) \cup X,\ \ \overline{\tau_{g_1}(q_1, \ldots, q_k)}),$$

where $X = \{\sigma_{g_2}\}$ if $\tau_{g_1}(q_1, \ldots, q_k) \in Q_{1,F}$ and $X = \emptyset$ otherwise.

We leave to the reader the details of verifying that D recognizes the tree language $L(A_1) \cdot_\sigma^p L(A_2)$. Among the states $(P, q) \in Q_D$, $P \subseteq Q_2$, $q \in Q_1'$, the states where $q \in Q_{1,F}$ and $\sigma_{g_2} \notin P$ are unreachable, which gives in total at most $(m+1) \cdot 2^n - 2^{n-1}$ states. Furthermore, we can eliminate from the state set of D the sink state $(\emptyset, \text{dead})$ which gives a DTA with the claimed number of states.

To establish a matching lower bound we consider the string languages defined by the DFAs A and B of Figures 6.1 and 6.2. As above, we construct a DFA D to recognize the concatenation of $L(A)$ and $L(B)$, hence states of D are pairs (P, i), $P \subseteq \{0, \ldots, n-1\}$, $i \in \{0, \ldots, m-1\} \cup \{\text{dead}\}$.

Denote by B' the DFA obtained from B by omitting all transitions on d. The fact that A and B' are the DFAs used in [64, 66] to establish the tight lower bound for concatenation of complete DFAs[3] implies that all states of D belonging to

$$Z = \{(P, i) \mid P \subseteq \{0, \ldots, n-1\}, 0 \leq i \leq m-1, \text{ where } i = m-1 \text{ implies } 0 \in P\}$$

are reachable and pairwise inequivalent. Furthermore, each state of the form (P, dead), $\emptyset \neq P \subseteq \{0, \ldots, n-1\}$ is reachable in D from state $(P, 0)$ by reading the symbol d. (Recall that d-transitions are undefined in A.)

In order to complete the proof, it is sufficient to show that states of the form (P, dead) are all pairwise inequivalent, and no state of this form can be equivalent with a state of Z.

First consider two states (P_1, dead) and (P_2, dead), where $i \in P_1 - P_2$. Now after

[3] A and B' are complete DFAs over the input alphabet $\{a, b, c\}$.

reading b^{n-1-i} from state (P_1, dead) (respectively, from state (P_2, dead)) D reaches a final (respectively, non-final) state.

Second consider states (P_1, dead) and (P_2, i), $0 \leq i \leq m-1$ (where P_1 and P_2 are not required to be distinct). Choose $w = ca^{m-1-i}b^{n-1}$. After reading c in state (P_1, dead), D goes to state $(\{1\}, \text{dead})$ and a is the identity on states of B. After this $n-1$ symbols b give the non-final state $(0, \text{dead})$. On the other hand, the symbol c yields from state (P_2, i) the state $(\{1\}, i)$ and reading the sequence a^{m-1-i} yields then $(\{0,1\}, m-1)$. After this the sequence b^{n-1} yields the accepting state $(\{0, n-1\}, 0)$. Thus the computation of D on input w reaches a non-final and a final state from states (P_1, dead) and (P_2, i), respectively. ■

6.2 State Complexity of Kleene-star

Concatenation of tree languages can be defined either as a sequential or a parallel operation. Here we consider iterated concatenation of trees, that is, an extension of the Kleene-star operation for tree languages. We have seen in Section 5.1.1 that iterated parallel concatenation is not a regularity-preserving operation and, consequently, we will focus on iterated sequential concatenation. Since sequential concatenation of trees is non-associative, there are two essentially different ways to define the corresponding iterated operation. We have named the variants as the *bottom-up star* and the *top-down star* operations.

We give tight state complexity bounds for both bottom-up and top-down Kleene-star operations. We show that the bottom-up star of a tree language recognized by a deterministic bottom-up automaton with n states can be recognized by an automaton with $(n + \frac{3}{2}) \cdot 2^{n-1}$ states and, furthermore, there exist worst-case examples where this number of states is needed. This bound is, roughly, n times the corresponding bound for regular string languages. On the other hand, the state complexity of the top-down star operation is shown to coincide with the state complexity of Kleene-star on string languages.

6.2.1 State complexity of bottom-up star

We establish for the bottom-up star operation a tight state complexity bound that is of a different order of magnitude than the state complexity of Kleene-star for string languages. First we give an upper bound for the state complexity of bottom-up star.

Lemma 44. *Suppose that tree language L is recognized by a DTA with n states. For $\sigma \in \Sigma_0$, the tree language $L_\sigma^{b,*}$ can be recognized by a DTA with $(n+\frac{3}{2})2^{n-1}$ states.*

Proof. Let $A = (\Sigma, Q, Q_F, g_A)$ be a DTA with n states recognizing the tree language L. Without loss of generality we assume that σ_{g_A} is defined, because otherwise

$$L(A)_\sigma^{b,*} = L(A)_\sigma^{b,0} \cup L(A)_\sigma^{b,1} = \{\sigma\} \cup L(A),$$

and it is easy to construct a DTA with $n+1$ states that recognizes $L(A) \cup \{\sigma\}$.

Choose three disjoint subsets of $2^Q \times (Q \cup \{\text{dead}\})$ by setting

(i) $P_1 = \{(S, q) \mid S \in 2^Q, \{q, \sigma_{g_A}\} \subseteq S, q \in Q_F\}$,

(ii) $P_2 = \{(S, q) \mid S \in 2^Q, q \in S \cap (Q - Q_F)\}$,

(iii) $P_3 = \{(S, \text{dead}) \mid S \in 2^Q, S \neq \emptyset\}$.

Here dead is a new element not in Q. Now define a DTA $B = (\Sigma, P, P_F, g_B)$ where

$$P = P_1 \cup P_2 \cup P_3 \cup \{p_{\text{new}}\}, \quad P_F = \{(S, q) \in P \mid S \cap Q_F \neq \emptyset\} \cup \{p_{\text{new}}\}.$$

We define the transitions of B by setting, $\sigma_{g_B} = p_{\text{new}}$, and for $\tau \in \Sigma_0 - \{\sigma\}$,

$$\tau_{g_B} = \begin{cases} (\{\tau_{g_A}, \sigma_{g_A}\}, \tau_{g_A}) & \text{if } \tau_{g_A} \in Q_F, \\ (\{\tau_{g_A}\}, \tau_{g_A}) & \text{if } \tau_{g_A} \in Q - Q_F, \\ \text{undefined,} & \text{if } \tau_{g_A} \text{ is undefined.} \end{cases} \tag{6.6}$$

To define transitions on Σ_m, $m \geq 1$, we view p_{new} as the state $(\{\sigma_{g_A}\}, \sigma_{g_A})$, and hence every state of B is represented in the form (S, q), $S \subseteq Q$, $q \in Q$. (Note that p_{new} is not the same as $(\{\sigma_{g_A}\}, \sigma_{g_A})$, because the former is an accepting state and the latter need not be accepting.) For $\tau \in \Sigma_m$ and $(S_1, q_1), \ldots, (S_m, q_m) \in P$, we first denote

$$X = \bigcup_{i=1}^{m} \{\tau_{g_A}(q_1, \ldots, q_{i-1}, z, q_{i+1}, \ldots, q_m) \mid z \in S_i\}$$

Now we define

$$\tau_{g_B}((S_1, q_1), \ldots, (S_m, q_m)) \tag{6.7}$$

to be equal to

(i) $(X \cup \{\sigma_{g_A}\}, \tau_{g_A}(q_1, \ldots, q_m))$ if $\tau_{g_A}(q_1, \ldots, q_m) \in Q_F$,

(ii) $(X, \tau_{g_A}(q_1, \ldots, q_m))$ if $\tau_{g_A}(q_1, \ldots, q_m) \in Q - Q_F$,

(iii) (X, dead) if $X \neq \emptyset$ and $\tau_{g_A}(q_1, \ldots, q_m)$ is undefined.

In the remaining case, where $X = \emptyset$ and $\tau_{g_A}(q_1, \ldots, q_m)$ is undefined, also (6.7) is undefined. Note that if for some $1 \leq i \leq m$, $q_i = \text{dead}$, this implies automatically that $\tau_{g_A}(q_1, \ldots, q_m)$ is undefined.

Recall that if (S, q), $S \subseteq Q$, $q \in Q$ is a state of B then $q \in S$ and, furthermore, if $q \in Q_F$ then $\sigma_{g_A} \in S$. The transitions of g_B preserve this property and the state in (i) (in (ii), (iii), respectively) is an element of P_1 (an element of P_2, P_3, respectively).

The second component of the state of B simply simulates the computation of A on the current subtree, and goes to the state dead if the next state of A is undefined.

Intuitively, the first component of the state of B consists of all states that A could reach at the current subtree t' assuming that

$$\text{in } t' \text{ at most one subtree of } L(A)^{b,k},\ k \geq 0, \text{ has been replaced by a leaf } \sigma. \tag{6.8}$$

Inductively, assume that B assigns to the root of tree t_i a state $(S_i, (t_i)_{g_A})$ where $S_i \subseteq Q$ satisfies the property (6.8) for t_i, $i = 1, \ldots, m$. Now the rule (6.7) assigns to the root of tree $t = \tau(t_1, \ldots, t_m)$ a state (S, q) where $q = \tau_{g_A}((t_1)_{g_A}, \ldots, (t_m)_{g_A})$ and S consists of all states that A could reach at the root of t assuming the computation uses as arguments $q_1, \ldots, q_m$ where at most one of the q_i's can be replaced by an arbitrary state from S_i, $1 \leq i \leq m$. This means that the state (S, q) again satisfies the property (6.8) for the tree t.

The choice of the set of final states P_F and Lemma 36 now imply that $L(B) = L(A)^{b,*}_{\sigma}$.

It remains to estimate the worst-case size of B. We note that if $Q_F = \{\sigma_{g_A}\}$, in B only states of the form $(\{q\}, q)$, $q \in Q$, can be reachable, and p_{new} can be identified with $(\{\sigma_{g_A}\}, \sigma_{g_A})$. In this case $L(A)^{b,*}_{\sigma}$ has a DTA with n states. Thus, without loss of generality we assume that Q_F contains a final state distinct from σ_{g_A}.

We note that $|P_1| = |Q_F| \cdot 2^{n-2}$, $|P_2| = |Q - Q_F| \cdot 2^{n-1}$ and $|P_3| = 2^n - 1$. Here the estimation of the size of P_1 relies on the above observation that we can exclude the possibility $Q_F = \{\sigma_{g_A}\}$. Thus, the cardinality of $P_1 \cup P_2 \cup P_3 \cup \{p_{\text{new}}\}$ is maximized as $(n + \frac{3}{2})2^{n-1}$ when $|Q_F| = 1$. ■

The upper bound of Lemma 44 is of a different order of magnitude than the known state complexity of Kleene-star for string languages [64]. It remains to verify that the bound of Lemma 44 can be reached in the worst case.

Figure 6.3 represents a DFA A used in [64, 66] for the lower bound construction

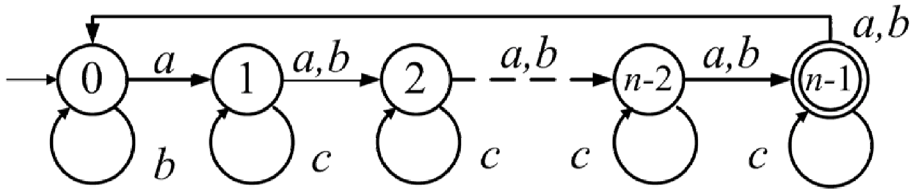

Figure 6.3: The DFA A from [64] with added c-transitions.

for Kleene-star where we have added transitions on the symbol c. Note that A is an incomplete DFA since the c transition on 0 is undefined. Based on A we define in the following a tree automaton M_A.

Choose $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$ where $\Sigma_0 = \{e\}$, $\Sigma_1 = \{a, b, c\}$ and $\Sigma_2 = \{a_2, d_2\}$. We define a DTA $M_A = (\Sigma, Q_A, Q_{A,F}, g_A)$, where $Q_A = \{0, 1, \ldots, n-1\}$, $Q_{A,F} = \{n-1\}$ and the transition function g_A is defined by setting:

(i) $e_{g_A} = 0$, $c_{g_A}(i) = i$, $1 \leq i \leq n-1$,

(ii) $a_{g_A}(i) = (a_2)_{g_A}(i, i) = i + 1$, $0 \leq i \leq n-2$,
$a_{g_A}(n-1) = (a_2)_{g_A}(n-1, n-1) = 0$,

(iii) $b_{g_A}(i) = i + 1$, $1 \leq i \leq n-2$, $b_{g_A}(j) = 0$, $j \in \{0, n-1\}$,

(iv) $(d_2)_{g_A}(0, i) = i$, $i = 0, 2, 3, \ldots, n-1$, $(d_2)_{g_A}(1, 1) = 1$.

All transitions of g_A not listed above are undefined. Intuitively, the construction of M_A can be, roughly speaking, explained as follows. Denote by T_d the subset of F_Σ consisting of trees without any occurrences of the binary symbol d_2, thus the only binary symbol in trees of T_d is a_2. On a tree $t \in T_d$, the DTA M_A simulates the computation of A on each string of symbols starting from a node of height one, where occurrences of a_2 are “interpreted” simply as a. The computations on different paths verify that for any $u \in \mathrm{dom}(t)$ labeled by a_2 and any nodes v_1 and v_2 of height one below u, the simulated computations started from v_1 and v_2 agree at u. Furthermore, if $u = \varepsilon$, the simulated computation has to accept.

Note that the original DFA has no transitions on d, and the transitions on d_2 have been added for a technical reason that will be used in the proof of Lemma 46. Also, the above intuitive description is not completely precise on how M_A operates on binary symbols a_2 where one child is a leaf (that gets assigned the state 0) and the other child is not a leaf. The following Lemmas 45 and 46 rely only on the formal definition of the transition function g_A of M_A. The above intuitive description of the operation of M_A is intended only as a guide that may be useful in understanding the operation of the DTA constructed to recognize the bottom-up e-star of $L(M_A)$. Finally, note that the d_2-transitions will be needed only to establish the reachability of one particular state, and in most of the technical constructions the above intuitive description of the operation of M_A (based on the DFA A of Figure 6.3) can be sufficient.

Using the construction of the proof of Lemma 44, based on M_A we construct a DTA $M_B = (\Sigma, Q_B, Q_{B,F}, g_B)$ that recognizes the tree language $L(M_A)_e^{b,*}$. We make the convention that the sink-state "dead" used in the proof is denoted by n. Thus the set of states Q_B consists of the special state p_{new} assigned to e and all pairs

$$(P, q),\ P \subseteq \{0, \ldots, n-1\},\ 0 \le q \le n, \tag{6.9}$$

where $0 \le q \le n-1$ implies $q \in P$, $q = n-1$ implies $0 \in P$ and $q = n$ implies $P \neq \emptyset$. The number of pairs as in (6.9) is $(n + \frac{3}{2})2^{n-1} - 1$.

In the following two lemmas we establish that M_B is a minimal DTA. That is, first we show that all states of Q_B are pairwise inequivalent with respect to the Myhill-Nerode equivalence relation extended to trees. Second we show that all states of Q_B are reachable, that is, for each $q \in Q_B$ there exists $t \in F_\Sigma$ such that $t_{g_B} = q$. The proof of our first lemma assumes that all states are reachable which will be established next in Lemma 46[4].

Lemma 45. *All states of M_B are pairwise inequivalent.*

Proof. For the sake of convenience, we assume that we have already proven that all states of M_B are reachable (Lemma 46). Thus, in order to distinguish two states with respect to the Myhill-Nerode relation, we can use an arbitrary configuration of M_B where one leaf is replaced by the given states. More formally, in order to show that two distinct states of Q_B, p_1 and p_2, are inequivalent, it is sufficient to find $t \in F_{\Sigma^{M_B}}[x]$ such that the computation of M_B started from the configuration $t(x \leftarrow p_1)$ accepts if and only if the computation started from the configuration $t(x \leftarrow p_2)$ does not accept.

We first show that any two distinct states (S_1, q_1) and (S_2, q_2) as in (6.9) are not equivalent. After that we consider the special state p_{new}. We begin by considering the case where neither of q_1 or q_2 is equal to n (which was used to denote the "dead" state of M_A).

Case $0 \leq q_1, q_2 \leq n-1$: (a) Assume $S_1 \neq S_2$ and $s \in S_1 - S_2$ (The other possibility is completely symmetric.) After reading $n-s-1$ unary symbols a, a final state is reached from state (S_1, q_1). On the other hand, since (S_2, q_2) is as in (6.9), $q_2 \neq s$. This means that the computation C that begins with (S_2, q_2) and reads $n-s-1$ unary symbols a ends with a non-final state. Note that at some point during the computation C, the second component may become $n-1$ which adds an element 0 to the first component. However, at the end of the computation C the first component cannot contain $n-1$.

[4]The proof of Lemma 46 does not rely on Lemma 45.

(b)(i) Next we consider the case $S_1 = S_2 = S$, $\{0, 1, \ldots, n-2\} \not\subseteq S$ and $q_1 \neq q_2$. According to the definition of the states (6.9), $q_1, q_2 \in S$. Choose $p \in \{0, 1, \ldots, n-2\} - S$ and consider a tree $t_1 = a^{2n-2-q_1} a_2((\{q_1, p\}, p), x) \in F_{\Sigma^{M_B}}[x]$. Since $p \in \{0, 1, \ldots, n-2\}$, $(\{q_1, p\}, p)$ is a legal state (6.9). Consider the computation of M_B on tree $t_1(x \leftarrow (S, q_1))$. Since $p \notin S$ the state $(\{q_1+1\}, n)$ is assigned to the root of the subtree $a_2((\{q_1, p\}, q_1), (S, q_1))$. (Here addition is modulo n.) After this the computation reads the $2n-2-q_1$ unary symbols a in t_1 and ends in an accepting state. On the other hand, consider the computation of M_B on $t_1(x \leftarrow (S, q_2))$. Since $p \notin S$ and $q_2 \notin \{q_1, p\}$, the transition $(a_2)_{g_B}$ on arguments $(\{q_1, p\}, p)$, $(S, q_2))$ is undefined and the computation does not accept.

(b)(ii) Consider $S = \{0, 1, \ldots, n-2\}$, and hence we know that $q_1, q_2 \neq n-1$. From state (S, q_i) by reading a unary symbol b we get (S', q_i'), where $S' = \{0, 2, \ldots, n-2, n-1\}$. Since $q_1, q_2 \neq n-1$, $q_1' \neq q_2'$ and the states (S', q_1') and (S', q_2') are distinguished as in b(i) above.

(b)(iii) Consider then the possibility $S = \{0, 1, \ldots, n-1\}$ and $q_1 \neq q_2$. If $\{q_1, q_2\} \neq \{0, n-1\}$, by reading a unary symbol b from (S, q_1) and (S, q_2), respectively, we get two states (S', q_1'), (S', q_2'), $q_1' \neq q_2'$, that are distinguished as in the previous case[5]. Next consider the case $\{q_1, q_2\} = \{0, n-1\}$, and first assume that $n \geq 3$. By reading a unary symbol a we obtain states (S, q_1+1), (S, q_2+1) where $q_1+1 \neq q_2+1$ and $q_i+1 \neq n-1$, $i = 1, 2$ (addition is modulo n). The states (S, q_1+1) and (S, q_2+1) can be distinguished as in the previous cases.

[5]The b-transitions of A violate injectivity only on states 0 and $n-1$.

Finally consider the possibility $n = 2$ and $\{q_1, q_2\} = \{0, 1\}$. From state $(\{0, 1\}, 1)$ by reading unary symbols ca, we reach the accepting state $(\{0, 1\}, 0)$. On the other hand, a computation starting from $(\{0, 1\}, 0)$ by reading the unary symbols ca reaches the non-accepting state $(\{0\}, 2)$.

Case where $q_2 = n$: First assume $q_1 \neq n$. Choose $t_2 \in F_{\Sigma^{M_B}}[x]$ by setting $t_2 = a^{n-2}a_2((\{0, 1\}, 1), b^{n-1}(x))$. Since $n-1$ consecutive b-transitions take any state of A to state 0, the computation of M_B on $t_2(x \leftarrow (S_1, q_1))$ assigns state $(\{0\}, 0)$ to the root of the subtree $b^{n-1}((S_1, q_1))$. Then the state $(\{1\}, n)$ is reached at the root of the subtree $a_2((\{0, 1\}, 1), b^{n-1}((S_1, q_1)))$. A final state $(\{n-1\}, n)$ is reached after reading further $n-2$ unary symbols a. On the other hand, in the computation of M_B on $t_2(x \leftarrow (S_2, n))$ the state $(\{0\}, n)$ is assigned to the root of the subtree $b^{n-1}((S_2, n))$. When reading the binary symbol a_2 with arguments $(\{0, 1\}, 1)$ and $(\{0\}, n)$ the computation step of M_B is undefined, and hence M_B does not accept $t_2(x \leftarrow (S_2, n))$.

Finally consider the case where also $q_1 = n$. Thus $S_1 \neq S_2$ and choose $s \in S_1 - S_2$. After reading $n - s - 1$ unary symbols a, a final state is reached from state (S_1, n), and the same computation does not reach a final state from (S_2, n).

It remains to show that p_{new} is not equivalent with any state (S, q) as in (6.9). Since p_{new} is final, it is sufficient to consider states where $n-1 \in S$. Thus, by reading a unary symbol c from state (S, q) we get a state (S', q'), where $n-1 \in S'$ and $0 \leq q' \leq n$. On the other hand, computations starting from p_{new} are identical to computations starting from $(\{0\}, 0)$ and hence a computation step with unary symbol c is undefined.

■

Before the next lemma we introduce the following notation. For a unary tree representing a configuration of M_B, $t = z_1(z_2(\ldots z_m(z_0)\ldots)) \in F_{\Sigma^{M_B}}$, we define $\text{word}(t) = z_m z_{m-1} \ldots z_1$. Note that $\text{word}(t)$ consists of the sequence of symbols labeling the nodes of t bottom-up, and the label of the leaf is not included. In the following when we refer to $\text{word}(t)$ of a tree t, without further mention, this implies that t is a unary tree.

Lemma 46. *All states of M_B are reachable.*

Proof. The transition function of M_B assigns the special state p_{new} to leaf symbol e. Recall that from p_{new} the computation of M_B continues as from $(\{0\}, 0)$. Thus, after reading $n-1$ unary symbols a we reach the state $(\{0, n-1\}, n-1)$.

Inductively, we assume that a state $(\{0, 1, 2, \ldots, k, n-1\}, n-1)$, $0 \le k < n-2$, is reachable. We show that $(\{0, 1, 2, \ldots, k+1, n-1\}, n-1)$ is also reachable. From state $(\{0, 1, 2, \ldots, k, n-1\}, n-1)$, we reach the state $Z_1 = (\{1, 2, \ldots, k+1, 0\}, 0)$ by reading a unary symbol a. By our assumption on k, $k+1 < n-1$. Thus from Z_1 we reach the state $Z_2 = (\{2, 3, \ldots, k+2, 0\}, 0)$ by reading b. Since $k < n-2$, all elements of $\{2, 3, \ldots, k+2, 0\}$ are distinct (that is, the b-transition does not take $k+1$ to 0). After reading $n-1$ symbols a, the state $(\{1, 2, \ldots, k+1, n-1, 0\}, n-1)$ is reached. The element 0 is added to the first component as the second component becomes $n-1$.

By the above inductive claim we now know that the state $(\{0, 1, \ldots, n-2, n-1\}, n-1)$ is reachable. After reading $i+1$ $a's$, state $(\{0, 1, \ldots, n-2, n-1\}, i)$ is reached, $0 \le i \le n-1$.

Inductively, assume that all states (S, j), where $|S| \ge k+1$, $1 \le k < n$ and $0 \le j \le n-1$ as in (6.9) are reachable. We show that then also states where $|S| = k$

are reachable. Let (S, s_i) where $S = \{s_1, s_2, \ldots, s_k\}$, $1 \leq i \leq k$ and $0 \leq s_1 < s_2 < \ldots < s_k \leq n-1$ be an arbitrary state where $|S| = k$. Recall that in states of M_B, when the second component is not n, it must belong to the first component.

In the cases (a) and (b) below, numbers $z \geq n$ are interpreted as the unique element of $\{0, 1, \ldots, n-1\}$ congruent to z modulo n.

(a-i) First consider the case where $s_i < n-1$. The following discussion assumes $n \geq 3$, and the case $n = 2$ is handled in case (a-ii). Since $|S| = k < n$, in the "cyclical sequence" of $s_1, \ldots, s_k$, there exist two consecutive numbers with difference at least two, where the difference between the numbers s_k and s_1 is counted modulo n. More formally, either there exists $1 \leq j \leq k-1$ such that $s_{j+1} - s_j \geq 2$ or $n + s_1 - s_k \geq 2$. In the latter case we choose $j = k$. In the following we assume that $i \leq j$. The case where $i > j$ is similar and only some notations are changed. According to the inductive assumption, the state $Z_3 = (\{0, n-1\} \cup S_1, n + s_i - s_j - 1)$ where $S_1 = \{s_{j+1} - s_j - 1, s_{j+2} - s_j - 1, \ldots, s_k - s_j - 1, n + s_1 - s_j - 1, n + s_2 - s_j - 1, \ldots, n + s_{j-1} - s_j - 1\}$ is reachable. Note that since $0 \leq s_1 < s_2 < \ldots < s_k \leq n-1$ and $s_{j+1} - s_j \geq 2$, $|S_1 \cup \{0, n-1\}| = k+1$. After reading from state Z_3 a unary symbol b, we get the state $Z_4 = (\{0\} \cup S_2, n + s_i - s_j)$ where $S_2 = \{s_{j+1} - s_j, s_{j+2} - s_j, \ldots, s_k - s_j, n + s_1 - s_j, n + s_2 - s_j, \ldots, n + s_{j-1} - s_j\}$. Since $0 \leq s_1 < s_2 < \ldots < s_k \leq n-1$, $0 \notin S_2$. From state Z_4 we reach the state $(\{s_j, s_{j+1}, s_{j+2}, \ldots, s_k, n + s_1, n + s_2, \ldots, n + s_{j-1}\}, n + s_i)$ by reading s_j symbols a. The latter state is the state (S, s_i) that we wanted.

(a-ii) Assume that $s_i < n-1$ and $n = 2$. Now $k = 1$, and the only legal state (S, s_i), $|S| = k = 1$, $0 \leq s_i < 1$, is $(\{0\}, 0)$ (because we know that $s_i \in S$). The state $(\{0\}, 0)$ is reached from state p_{new} by reading unary symbols ab.

(b) Now consider the case where $s_i = n-1$, and thus $i = k$. This implies that $0 \in S$, and we have $s_i(= s_k) = n-1$ and $s_1 = 0$. Since $k < n$, there exists $1 \le j \le k-1$ such that $s_{j+1} - s_j \ge 2$. According to the inductive assumption, the state $Z_5 = (\{0, n-1\} \cup S_3, n-2-s_j)$ is reachable, where $S_3 = \{s_{j+1} - s_j - 1, s_{j+2} - s_j - 1, \ldots, s_{k-1} - s_j - 1, n-1-s_j-1, n+0-s_j-1, n+s_2-s_j-1, \ldots, n+s_{j-1}-s_j-1\}$. Similarly as in (a) above we observe that $|S_3 \cup \{0, n-1\}| = k+1$. From state Z_5 we get the state $Z_6 = (\{s_{j+1}-s_j, s_{j+2}-s_j, \ldots, s_{k-1}-s_j, n-1-s_j, n+0-s_j, n+s_2-s_j, \ldots, n+s_{j-1}-s_j, 0\}, n-1-s_j)$ by reading a symbol b. After reading s_j symbols a, from state Z_6 we reach the state $(\{s_{j+1}, s_{j+2}, \ldots, s_{k-1}, n-1, n+0, n+s_2, \ldots, n+s_{j-1}, s_j\}, n-1)$. This means that we have reached the desired state $(S, n-1)$ with $S = \{0, s_2, \ldots, s_{k-1}, n-1\}$.

Up to now, we have shown that all that states (S, j), $S \subseteq \{0, \ldots, n-1\}$, $0 \le j \le n-1$ as in (6.9) are reachable. Next we will show that the states (S, n), $S \subset \{0, 1, \ldots, n-1\}$ are reachable.

We know that $(\{0, 1, \ldots, n-1\}, 0)$ is reachable and from this state we get $Z_7 = (\{1, \ldots, n-1\}, n)$ by reading a unary symbol c. From Z_7 we get all states (S, n), $|S| = n-1$ by cycling the elements of S using a-transitions. Now inductively, assume that all states (S, n), $n > |S| \ge k+1$, $k < n-1$ are reachable. Consider an arbitrary state (S, n) where $|S| = k$. Choose $0 \le j \le n-1$ such that $j \notin S$. By our inductive assumption the state $(S \cup \{j\}, n)$ is reachable. From this state we reach (S, n) by reading the sequence of unary symbols $a^{n-j}ca^j$. Note that transitions on a always add one modulo n to states of S and the c-transition deletes the element 0 and is the identity on all other elements.

It remains to consider the state $(\{0, 1, \ldots, n-1\}, n)$. We know that states

$(\{0,1\},0)$ and $(\{0,1,\ldots,n-1\},1)$ are reachable. According to the definition of d_2-transitions of M_A, the d_2-transition of M_B with arguments $(\{0,1\},0)$ and $(\{0,1,\ldots,n-1\},1)$ gives the state $(\{0,1,\ldots,n-1\},n)$. ■

Note that above the transitions on d_2 were needed only to establish that the state $(\{0,1,\ldots,n-1\},n)$ is reachable in M_B. The transitions of d_2 in M_A did not have a similar intuitive interpretation as the other transitions based on the DFA A, and they were introduced only for the technical purpose needed at the end of the proof of Lemma 46.

By Lemmas 44, 45 and 46 we have a tight bound for the state complexity of bottom-up star that differs by an order of magnitude from the known bound for Kleene-star of string languages [64].

Theorem 47. *If A is a DTA with n states, the bottom-up star of $L(A)$ can be recognized by a DTA with $(n+\frac{3}{2})\cdot 2^{n-1}$ states. For every $n \geq 2$, there exists an n-state DTA A and $\sigma \in \Sigma_0$ such that the minimal DTA for $L(A)^{b,*}_{\sigma}$ has $(n+\frac{3}{2})\cdot 2^{n-1}$ states.*

6.2.2 State complexity of top-down star

Here we give a tight state complexity bound for top-down star of regular tree languages. The top-down iteration of the concatenation operation allows the replacement of subtrees at arbitrary locations and, as can perhaps be expected, the state complexity is similar as for the Kleene-star of string languages. For completeness, we give a brief construction for the upper bound, because we are considering incomplete automata and the known state complexity bounds for ordinary DFAs are stated in terms of complete DFAs [64, 66]. The state complexity results for complete and incomplete DFAs, respectively, differ slightly for operations such as union [64, 13] or concatenation [64, 46].

Theorem 48. *Let A be a DTA with n states and $\sigma \in \Sigma_0$. The top-down σ-star of the tree language recognized by A, $L(A)_\sigma^{t,*}$, can be recognized by a DTA with $\frac{3}{4} \cdot 2^n$ states and this bound can be reached in the worst case.*

Proof. Denote $A = (\Sigma, Q_A, Q_{A,F}, g_A)$ and let q_{new} be a new element not in Q_A. We can assume that σ_{g_A} is defined because otherwise $L(A)_\sigma^{t,*} = L(A) \cup \{\sigma\}$.

We define $B = (\Sigma, Q_B, Q_{B,F}, g_B)$, where

$$Q_B = \{q_{\text{new}}\} \cup \{\emptyset \neq P \subseteq Q_A \mid P \cap Q_{A,F} \neq \emptyset \text{ implies } \sigma_{g_A} \in P\},$$

$$Q_{B,F} = \{q_{\text{new}}\} \cup \{P \in Q_B \mid P \cap Q_{A,F} \neq \emptyset\}.$$

The transitions of B are defined for $\tau \in \Sigma_0 - \{\sigma\}$ by setting

$$\tau_{g_B} = \begin{cases} \{\tau_{g_A}, \sigma_{g_A}\} & \text{if } \tau_{g_A} \in Q_{A,F}, \\ \{\tau_{g_A}\} & \text{if } \tau_{g_A} \in Q_A - Q_{A,F}, \\ \text{undefined} & \text{if } \tau_{g_A} \text{ is undefined.} \end{cases}$$

For the leaf symbol σ used to define the star-operation, we set $\sigma_{g_B} = q_{\text{new}}$. For $m \geq 1$, $\tau \in \Sigma_m$ and $X_1, \ldots, X_m \in Q_B$ we define $\tau_{g_B}(X_1, \ldots, X_m) = Y \cup Z$, where

$$Y = \{\tau_{g_A}(x_1, \ldots, x_m) \mid x_i \in X_i \text{ if } X_i \in 2^{Q_A},\ x_i = \sigma_{g_A} \text{ if } X_i = q_{\text{new}}, 1 \leq i \leq m\},$$

and $Z = \{\sigma_{g_A}\}$ if $Y \cap Q_{A,F} \neq \emptyset$, $Z = \emptyset$ otherwise.

The construction of B is similar as the construction used to recognize the Kleene-star of a string language. Note that the state q_{new} is used as a copy of σ_{g_A} because the latter state is not, in general, accepting. We leave to the reader the details of verifying that B recognizes $L(A)_\sigma^{t,*}$.

To get the upper bound on the number of states, we note that if $Q_{A,F} = \{\sigma_{g_A}\}$, then we can identify q_{new} and σ_{g_A} and in the resulting DTA the number of reachable states is (at most) n. Thus we can assume that $Q_{A,F} - \{\sigma_{g_A}\} \neq \emptyset$. In the case where $\sigma_{g_A} \notin Q_{A,F}$, we observe that $(2^{|Q_{A,F}|} - 1) \cdot 2^{n-|Q_{A,F}|-1}$ of the elements $P \in 2^{Q_A} - \{\emptyset\}$ contain an element of $Q_{A,F}$ and, at the same time, do not contain σ_{g_A}. Thus, by choosing $|Q_{A,F}| = 1$, the cardinality of Q_B is maximized as $|Q_B| = 2^n - 1 - (2 - 1) \cdot 2^{n-2} + 1 = \frac{3}{4} \cdot 2^n$. It is easy to verify that this bound cannot be exceeded with $\sigma_{g_A} \in Q_{A,F}$, $|Q_{A,F}| \geq 2$.

When restricted to unary trees, the top-down (or bottom) star operation coincides with Kleene-star on string languages. Theorem 5.5 of [64] gives a complete DFA C with n states such that the state complexity of the Kleene-star of $L(C)$ is $\frac{3}{4} \cdot 2^n$. Furthermore, C does not have a “dead” state, which means that the same lower bound construction works for incomplete DFAs. ■

Chapter 7

State complexity of other operations

In this chapter we consider the state complexity of union, intersection, projection and quotient operations on tree languages. We consider unranked tree automata and, for union and intersection, we give the bounds separately for strongly deterministic automata and DTA(DFA)s.

7.1 State complexity of union

We investigate the upper bounds on the numbers of vertical and horizontal states for the operation of union on SDTAs and DTA(DFA)s. The upper bounds for vertical states are tight for both SDTAs and DTA(DFA)s, and as can be expected these bounds are similar as the state complexity of union on regular strings. We also get an upper bound which is almost tight for the number of the horizontal states of SDTAs. Obtaining a matching lower bound for the horizontal states of DTA(DFA)s turns out to be very problematic. This is mainly because the minimal DTA(DFA)s may not be unique and the minimization of DTA(DFA)s is intractable [35]. Also, as we have seen in Chapter 3, the number of horizontal states of DTA(DFA)s can be reduced by adding vertical states, i.e., there can be trade-offs between vertical and horizontal states.

7.1.1 Union of strongly deterministic automata

The following lemma gives the upper bound for the operation of union for SDTAs.

Lemma 49. *Let A_1 and A_2 be two arbitrary SDTAs, $A_i = (Q_i, \Sigma, \delta_i, F_i)$, $i = 1, 2$, transition function for each $\sigma \in \Sigma$ is represented by a DFA $H_\sigma^{A_i} = (C_\sigma^i, Q_i \cup \Sigma, \gamma_\sigma^i, c_{\sigma,0}^i, E_\sigma^i)$ with an output function λ_σ^i.*

The language $L(A_1) \cup L(A_2)$ can be recognized by an SDTA $B_\cup$ with

$$\text{size}(B_\cup) \leq [\ (|Q_1| + 1) \times (|Q_2| + 1) - 1;\ \sum_{\sigma \in \Sigma}((|C_\sigma^1| + 1) \times (|C_\sigma^2| + 1) - 1)\ ].$$

Proof. Choose $B_\cup = (Q_1' \times Q_2', \Sigma, \delta, F)$, where $Q_i' = Q_i \cup \{dead\}, i = 1, 2$, $(q, p) \in Q_1' \times Q_2'$ is final if $q \in F_1$ or $p \in F_2$.

The transition function for each σ is represented by a DFA $H_\sigma^{B_\cup} = ((C_\sigma^1 \cup \{dead\}) \times (C_\sigma^2 \cup \{dead\}), (Q_1' \times Q_2') \cup \Sigma, \mu, (c_{\sigma,0}^1, c_{\sigma,0}^2), V)$ with an output function $\lambda_\sigma^{B_\cup}$, where $(c_1, c_2) \in (C_\sigma^1 \cup \{dead\}) \times (C_\sigma^2 \cup \{dead\})$ is final if $c_1 \in E_\sigma^1$ or $c_2 \in E_\sigma^2$. Transition function μ is defined as below: For any input $(q, p) \in Q_1' \times Q_2'$,

$$\mu((c_1, c_2), (q, p)) = \begin{cases} (\gamma_\sigma^1(c_1, q), \gamma_\sigma^2(c_2, p)) \text{ if } \gamma_\sigma^1(c_1, q) \text{ and } \gamma_\sigma^2(c_2, p) \text{ are both defined,} \\ (\gamma_\sigma^1(c_1, q), dead) \text{ if } \gamma_\sigma^1(c_1, q) \text{ is defined but } \gamma_\sigma^2(c_2, p) \text{ is not defined,} \\ (dead, \gamma_\sigma^2(c_2, p)) \text{ if } \gamma_\sigma^2(c_2, p) \text{ is defined but } \gamma_\sigma^1(c_1, q) \text{ is not defined,} \end{cases}$$

For any input $a \in \Sigma$,

$$\mu((c_1, c_2), a) = \begin{cases} (\gamma^1_\sigma(c_1, a), \gamma^2_\sigma(c_2, a)) \text{ if } \gamma^1_\sigma(c_1, a) \text{ and } \gamma^2_\sigma(c_2, a) \text{ are both defined,} \\ (\gamma^1_\sigma(c_1, a), \mathit{dead}) \text{ if } \gamma^1_\sigma(c_1, a) \text{ is defined but } \gamma^2_\sigma(c_2, a) \text{ is not defined,} \\ (\mathit{dead}, \gamma^2_\sigma(c_2, a)) \text{ if } \gamma^2_\sigma(c_2, a) \text{ is defined but } \gamma^1_\sigma(c_1, a) \text{ is not defined,} \end{cases}$$

For any $(c_1, c_2) \in (C^1_\sigma + \{\mathit{dead}\}) \times (C^2_\sigma + \{\mathit{dead}\})$ and $(c_1, c_2) \in V$,

$$\lambda((c_1, c_2)) = \begin{cases} (\lambda^1_\sigma(c_1), \lambda^2_\sigma(c_2)) \text{ if } c_i \neq \mathit{dead} \text{ and } c_i \in E^i_\sigma \text{ for both } i = 1, 2 \\ (\lambda^1_\sigma(c_1), \mathit{dead}) \text{ if } c_1 \neq \mathit{dead} \text{ and } c_1 \in E^1_\sigma \text{ but } c_2 = \mathit{dead} \\ (\mathit{dead}, \lambda^2_\sigma(c_2)) \text{ if } c_1 = \mathit{dead} \text{ but } c_2 \neq \mathit{dead} \text{ and } c_2 \in E^2_\sigma \end{cases}$$

State $(\mathit{dead}, \mathit{dead})$ is not used in vertical or horizontal computation. Thus, we deduct the numbers of horizontal states in each $H^{B_\cup}_\sigma$ and the vertical states both by 1. ■

Next we state a lower bound result.

Lemma 50. *For any integers m, n and relatively prime numbers $k_1 < k_2 < \ldots < k_m < k_{m+1} < \ldots < k_{m+n}$, there exist tree languages T_1 and T_2 such that T_1, T_2 can be recognized respectively by an SDTA A_1, A_2 with $\text{size}(A_1) \leq [m;\ \prod_{i=1}^{m} k_i + m]$ and $\text{size}(A_2) \leq [n;\ \prod_{i=1}^{n} k_{i+m} + n]$.*

Any SDTA recognizing $T_1 \cup T_2$ needs at least $(m+1)(n+1) - 1$ vertical states and $\prod_{i=1}^{m+n} k_i$ horizontal states.

Proof. Let $Z_1 = \prod_{i=1}^{m} k_i$, $Z_2 = \prod_{i=m+1}^{m+n} k_i$, $Z = \prod_{i=1}^{m+n} k_i$. Consider tree languages T_1 and T_2. Tree language T_1 consists of trees $t \in \{e^{i-1}(a(a^k))\}$, where

(1) $\exists i : 1 \leq i \leq m, k \equiv 0 (\text{mod } k_i)$,

(2) $\forall j : 1 \leq j \leq m, j \neq i, k_j$ does not divide k.

T_2 consists of trees $t \in b^{i-m-1}(a(a^k))$, where

(1) $\exists i : m+1 \leq i \leq m+n, k \equiv 0 (\text{mod } k_i)$,

(2) $\forall j : m+1 \leq j \leq m+n, j \neq i, k_j$ does not divide k.

Tree language T_1 can be recognized by an SDTA $A_1 = (Q_1, \{a, e\}, \delta_1, F_1)$, where $Q_1 = \{p_1, p_2, \ldots, p_m\}$, $F_1 = \{p_1\}$. The transition function associated with a is represented by a DFA $H_a^{A_1} = (C_a^1, Q_1 \cup \{a\}, \gamma_a^1, c_0^1, E_a^1)$ with an output function λ_a^1, where $C_a^1 = \{c_0^1, c_1^1, \ldots, c_{Z_1-1}^1\}$. For $0 \leq i \leq Z_1 - 2$, $\gamma_a^1(c_i^1, a) = c_{i+1}^1$, $\gamma_a^1(c_{Z_1-1}^1, a) = c_0^1$. For any $c_k^1 \in C_a^1$ where k satisfies that

$$(1) \exists j : 1 \leq j \leq m, k \equiv 0 (\text{mod } k_j),\ (2) \forall i : 1 \leq i \leq m, i \neq j, k_i \text{ does not divide } k,$$

define $c_k^1 \in E_a^1$ and $\lambda_a^1(c_k^1) = p_j$. The horizontal language associated with e is represented by a DFA $H_e^{A_1} = (D^1, Q_1, \gamma_e^1, d_0^1, E_e^1)$ with an output function λ_e^1, where $D^1 = \{d_0^1, d_1^1, d_2^1, \ldots, d_{m-1}^1\}$. For $1 \leq i \leq m-1$, the transition function is defined as $\gamma_e^1(d_0^1, p_{i+1}) = d_i^1$. The set of final states is $E_e^1 = \{d_1^1, d_2^1, \ldots, d_{m-1}^1\}$. For $1 \leq i \leq m-1$, the output function is $\lambda(d_i^1) = p_i$. The SDTA A_2 recognizing T_2 can be similarly defined.

Let $B_\cup = (Q, \{a, b, e\}, \delta, F)$ be an arbitrary SDTA recognizing $T_1 \cup T_2$. $B_\cup$ needs at least $(m+1)(n+1) - 1$ vertical states. Let $t \in T_\Sigma[x]$ be arbitrary. Choose $R = \{a(a^{Z+1})\} \cup \{a(a^k)\}$, where k can be any of the following values:

(a) $k_i \cdot k_j$ where $1 \leq i \leq m, m+1 \leq j \leq m+n$,

(b) k_i where $1 \leq i \leq m+n$.

For any two trees $w_1 = a(a^{k_{x_1} \cdot k_{y_1}})$ and $w_2 = a(a^{k_{x_2} \cdot k_{y_2}})$, where $1 \leq x_1, x_2 \leq m$, $m+1 \leq y_1, y_2 \leq m+n$, choose $t = e^{x_1-1}(x)$ if $x_1 \neq x_2$, $t = b^{y_1-m-1}(x)$ if $y_1 \neq y_2$. We have $t(x \leftarrow w_1) \in T_1 \cup T_2$ and $t(x \leftarrow w_2) \notin T_1 \cup T_2$.

For any two trees $w_1 = a(a^{k_{x_1} \cdot k_{y_1}})$ and $w_2 = a(a^{k_{x_2}})$, where $1 \leq x_1 \leq m$, $m+1 \leq y_1 \leq m+n, 1 \leq x_2 \leq m+n$, choose $t = b^{y_1-m-1}(x)$ if $1 \leq x_2 \leq m$, $t = e^{x_1-1}(x)$ if $m+1 \leq x_2 \leq m+n$. We have $t(x \leftarrow w_1) \in T_1 \cup T_2$ and $t(x \leftarrow w_2) \notin T_1 \cup T_2$.

For any two trees $w_1 = a(a^{k_{x_1}})$ and $w_2 = a(a^{k_y})$, where $1 \leq x_1, y \leq m+n$, choose $t = e^{x_1-1}(x)$ if $1 \leq x_1 \leq m$, $t = b^{x_1-m-1}(x)$ if $1+m \leq x_1 \leq m+n$. We have $t(x \leftarrow w_1) \in T_1 \cup T_2$ and $t(x \leftarrow w_2) \notin T_1 \cup T_2$.

For any two trees $w_1 = a(a^{k_{x_1}})$ and $w_2 = a(a^{Z+1})$, choose $t = e^{x_1-1}(x)$ if $1 \leq x_1 \leq m$, $t = b^{x_1-m-1}(x)$ if $1+m \leq x_1 \leq n+m$. We have $t(x \leftarrow w_1) \in T_1 \cup T_2$ and $t(x \leftarrow w_2) \notin T_1 \cup T_2$.

For any two trees $w_1 = a(a^{k_{x_1} \cdot k_y})$ and $w_2 = a(a^{Z+1})$, choose $t = e^{x_1-1}(x)$. We have $t(x \leftarrow w_1) \in T_1 \cup T_2$ and $t(x \leftarrow w_2) \notin T_1 \cup T_2$.

Since $|R| = (m+1)(n+1)$, according to Lemma 4, $B_\cup$ has at least $(m+1)(n+1)-1$ vertical states.

Now we show that $B_\cup$ needs at least Z horizontal states. Let $t \in T_\Sigma[x]$ be arbitrary. Choose $S = \{b\} \cup \{a^j \mid 1 \leq j \leq Z\}$. Consider any distinct integers $1 \leq x < y \leq Z$. Since all the k_i's are relatively prime, there exists a k_i, $1 \leq i \leq m+n$, such that k_i does not divide $y-x$. Choose $0 \leq z < k_i$ such that $y+z \equiv 0 (\text{mod } k_i)$. Let $t = e^{i-1}(x)$ if $1 \leq i \leq m$, $t = b^{i-m-1}(x)$ if $1+m \leq i \leq n+m$. Since k_i divides $y+z$ and does not divide $y-x$, k_i does not divide $z+x$. Now $t(x \leftarrow a(a^{y+z})) \in T_1 \cup T_2$ and $t(x \leftarrow a(a^{x+z})) \notin T_1 \cup T_2$. Consider any string a^j, $1 \leq j \leq Z$ and b. Let $t = e(x)$. Now we have $t(x \leftarrow a(a^{j+Z-j+k_2})) \in T_1 \cup T_2$ and $t(x \leftarrow a(ba^{Z-j+k_2})) \notin T_1 \cup T_2$. According to Lemma 5, $B_\cup$ needs at least Z horizontal states. ■

Following Lemma 49 and Lemma 50, the theorem below is immediate.

Theorem 51. *For any two arbitrary SDTAs $A_i = (Q_i, \Sigma, \delta_i, F_i)$, $i = 1, 2$, whose* *transition function associated with* σ *is represented by a DFA* $H_\sigma^{A_i} = (C_\sigma^i, Q_i \cup \Sigma, \gamma_\sigma^i, c_{\sigma,0}^i, E_\sigma^i)$ *augmented with an output function* λ_σ^i*, we have*

1 *There exists an SDTA* $B_\cup$ *recognizing* $L(A_1) \cup L(A_2)$ *such that*

$$\text{size}(B_\cup) \leq [\ (|Q_1|+1) \times (|Q_2|+1) - 1;\ \sum_{\sigma\in\Sigma}((|C_\sigma^1|+1) \times (|C_\sigma^2|+1) - 1)\ ].$$

2 *For integers* $m, n \geq 1$ *and relatively prime numbers* $k_1, k_2, \ldots, k_m, k_{m+1}, \ldots,$ k_{m+n}*, there exists tree languages* T_1 *and* T_2 *such that* T_1 *and* T_2*, respectively, can be recognized by SDTAs with* m *and* n *vertical states,* $\prod_{i=1}^{m} k_i + O(m)$ *and* $\prod_{i=1+m}^{m+n} k_i + O(n)$ *horizontal states, and any SDTA recognizing* $T_1 \cup T_2$ *has at least* $(m+1)(n+1) - 1$ *vertical states and* $\prod_{i=1}^{m+n} k_i$ *horizontal states.*

Theorem 51 shows that for the operation of union on SDTAs the upper bounds are tight for vertical states and almost tight for horizontal states.

7.1.2 Union of DTA(DFA)s

The upper bounds on the numbers of vertical and horizontal states for the operation of union on DTA(DFA)s are investigated, and followed by a matching lower bound on the number of vertical states.

Lemma 52. *Let* $A_i = (Q_i, \Sigma, \delta_i, F_i)$*,* $i = 1, 2$*, be DTA(DFA)s where each horizontal language* $\delta_i(q, \sigma)$ *is represented by a DFA* $D_{q,\sigma}^{A_i} = (C_{q,\sigma}^i, Q_i \cup \Sigma, \gamma_{q,\sigma}^i, c_{q,\sigma,0}^i, E_{q,\sigma}^i)$.

The language $L(A_1) \cup L(A_2)$ *can be recognized by a DTA(DFA)* $B_\cup$ *with*

$$\text{size}(B_\cup) \leq [\ (|Q_1|+1) \times (|Q_2|+1) - 1;$$
$$|\Sigma| \times (\textstyle\sum_{q\in Q_1, p\in Q_2} |D_{q,\sigma}^{A_1}| \times |D_{p,\sigma}^{A_2}| + \sum_{q\in Q_1} |D_{q,\sigma}^{A_1}| \times \prod_{p\in Q_2} |D_{p,\sigma}^{A_2}| + \sum_{p\in Q_2} |D_{p,\sigma}^{A_2}|$$
$$\times \textstyle\prod_{q\in Q_1} |D_{q,\sigma}^{A_1}|)\ ]$$

Proof. Denote the complement of $D^{A_i}_{q,\sigma}$ as: $\overline{D}^{A_i}_{q,\sigma} = (\overline{C}^i_{q,\sigma}, Q_i \cup \Sigma, \overline{\gamma}^i_{q,\sigma}, \overline{c}^i_{q,\sigma,0}, \overline{E}^i_{q,\sigma})$.

The language $L(A_1) \cup L(A_2)$ can be recognized by a DTA(DFA) $B_\cup = (Q'_1 \times Q'_2, \Sigma, \delta, F)$, where $Q'_i = \{Q_i \cup \{dead\}\}, i = 1, 2$, $(q, p) \in Q'_1 \times Q'_2$ is final if $q \in F_1$ or $p \in F_2$. δ is defined as below:

(1) For any $(q, p) \in Q_1 \times Q_2$, horizontal language $\delta((q, p), \sigma)$ is represented by a DFA $G_{(q,p),\sigma} = (C^1_{q,\sigma} \times C^2_{p,\sigma}), (Q'_1 \times Q'_2) \cup \Sigma, (c^1_{q,\sigma,0}, c^2_{p,\sigma,0}), \mu, V)$, where $(c^1, c^2) \in C^1_{q,\sigma} \times C^2_{p,\sigma}$ is final if $c^1 \in E^1_{q,\sigma}$ and $c^2 \in E^2_{p,\sigma}$. μ is defined as below:

   (i) For any input $a \in \Sigma$,

   $\mu((c^1, c^2), a) = (d^1, d^2)$, $d^1 = \gamma^1_{q,\sigma}(c^1, a)$, $d^2 = \gamma^2_{p,\sigma}(c^2, a)$, $d^i = dead$, $i = 1, 2$, if $\gamma^i_{x,\sigma}(c^i, a)$ is undefined, x is p or q.

   (ii) For any input $(x, y) \in Q'_1 \times Q'_2$,

   $\mu((c^1, c^2), (x, y)) = (d^1, d^2)$, $d^1 = \gamma^1_{q,\sigma}(c^1, x)$, $d^2 = \gamma^2_{p,\sigma}(c^2, y)$, $d^i = dead$, $i = 1, 2$, if $\gamma^1_{q,\sigma}(c^1, x)$ or $\gamma^2_{p,\sigma}(c^2, y)$ is undefined.

(2) Suppose $Q_2 = \{p_1, p_2, \ldots, p_k\}$. For any $(q, dead)$, $q \in Q_1$, horizontal language $\delta((q, dead), \sigma)$ is represented by a DFA $G_{(q,dead),\sigma} = (C^1_{q,\sigma} \times \overline{C}^2_{p_1,\sigma} \times \overline{C}^2_{p_2,\sigma} \times \ldots \times \overline{C}^2_{p_k,\sigma}, (Q'_1 \times Q'_2) \cup \Sigma, \mu, (c^1_{q,\sigma,0}, \overline{c}^2_{p_1,\sigma,0}, \overline{c}^2_{p_2,\sigma,0}, \ldots, \overline{c}^2_{p_k,\sigma,0}), V)$, where $(c^1, c^2_1, c^2_2, \ldots, c^2_k) \in C^1_{q,\sigma} \times \overline{C}^2_{p_1,\sigma} \times \overline{C}^2_{p_2,\sigma} \times \ldots \times \overline{C}^2_{p_k,\sigma}$ is final if $c^1 \in E^1_{q,\sigma}$ and $c^2_i \in \overline{E}^2_{p_i,\sigma}$ for all $i = 1, 2, \ldots, k$. The transition function μ is defined as below:

   (i) For any input $a \in \Sigma$,

   $\mu((c^1, c^2_1, c^2_2, \ldots, c^2_k), a) = (d^1, d^2_1, d^2_2, \ldots, d^2_k)$, $d^1 = \gamma^1_{q,\sigma}(c^1, a)$, $d^2_i = \overline{\gamma}^2_{p_i,\sigma}(c^2_i, a)$. $d^1 = dead$, if $\gamma^1_{q,\sigma}(c^1, a)$ is undefined. $d^2_i = dead$, if $\overline{\gamma}^2_{p_i,\sigma}(c^2_i, a)$ is undefined, $i = 1, 2, \ldots, k$.

(ii) For any input $(x, y) \in Q'_1 \times Q'_2$,

$\mu((c^1, c^2_1, c^2_2, \ldots, c^2_k), (x, y)) = (d^1, d^2_1, d^2_2, \ldots, d^2_k)$, $d^1 = \gamma^1_{q,\sigma}(c^1, x)$, $d^2_i = \overline{\gamma}^2_{p_i,\sigma}(c^2_i, y)$. $d^1 = dead$, if $\gamma^1_{q,\sigma}(c^1, x)$ is undefined. $d^2_i = dead$, if $\overline{\gamma}^2_{p_i,\sigma}(c^2_i, y)$ is undefined, $i = 1, 2, \ldots, k$.

(3) $\delta((dead, p), \sigma)$ is defined similarly.

In the computation of $B_\cup$, state (q, p), $p, q \neq dead$, is assigned to a node u labeled by a whose children are assigned with states $(q_1, p_1), \ldots, (q_z, p_z)$ if and only if A_1 assigns state q to u after reading sequence $q_1 \ldots q_z$ and A_2 assigns state p to u after reading sequence $p_1 \ldots p_z$. $(q, dead)$ is assigned to u, if A_1 assigns q to u and none of the horizontal languages associated with a in A_2 recognizes the sequence of states $p_1 \ldots p_z$. A similar situation happens when state $(dead, p)$ is assigned to u. The automaton $B_\cup$ accepts when either the first or the second component of the state is final. ■

Next we state a lower bound on the number of vertical states.

Lemma 53. *For any integers m, n, there exist tree languages T_1 and T_2 such that T_1, T_2 can be recognized respectively by a DTA(DFA) with at most m and n vertical states.*

Any DTA(DFA) recognizing $T_1 \cup T_2$ needs at least $(m+1)(n+1)-1$ vertical states.

Proof. Consider tree languages $T_1 = \{b^i(a(y_i\ a\ x_j)) \mid 1 \leq i \leq m, 1 \leq j \leq n+1\}$, and $T_2 = \{c^j(a(y_i\ a\ x_j)) \mid 1 \leq i \leq m+1, 1 \leq j \leq n\}$, where y_i (or x_j) is a binary expression of number i or (j).

T_1 can be recognized by a DTA(DFA) $A_1 = (Q_1, \{a, b, 1, 0\}, \delta_1, F_1)$, where $Q_1 = \{q_1, q_2, \ldots, q_m\}$, $F_1 = \{q_1\}$. For $1 \leq i \leq m$, $\delta(q_i, a) = y_i\ a\ x_j$, where $1 \leq j \leq n+1$.

For $1 \leq i \leq m-1$, $\delta(q_i, b) = q_{i+1}$. DTA(DFA) recognizing T_2 can be similarly defined.

Let $B_\cup = (Q, \{a, b, c, 1, 0\}, \delta, F)$ be an arbitrary DTA(DFA) recognizing $T_1 \cup T_2$. $B_\cup$ needs at least $(m+1)(n+1) - 1$ vertical states. Let $t \in T_\Sigma[x]$. Choose $R = \{a(y_i a x_j) \mid 1 \leq i \leq m+1, 1 \leq j \leq n+1, \text{and when } i = m+1, j \neq n+1\} \cup \{a(0aa0)\}$. Consider any two trees $w_1 = a(y_{u_1} a x_{v_1})$ and $w_2 = a(y_{u_2} a x_{v_2})$ in R. When $u_1 \neq u_2$, then u_1 and u_2 can't both be equal to $m+1$. Without loss of generality, suppose $u_1 \neq m+1$. Choose $t = b^{u_1}(x)$. We have $t(x \leftarrow w_1) \in T_1 \cup T_2$ and $t(x \leftarrow w_2) \notin T_1 \cup T_2$. When $v_1 \neq v_2$, similarly suppose $v_1 \neq n+1$. Choose $t = c^{v_1}(x)$. We have $t(x \leftarrow w_1) \in T_1 \cup T_2$ and $t(x \leftarrow w_2) \notin T_1 \cup T_2$. For any two trees $w_1 = a(y_u a x_v)$ and $w_2 = a(0aa0)$, choose $t = b^u(x)$ when $u \neq m+1$, $t = c^v(x)$ when $v \neq n+1$. We have $t(x \leftarrow w_1) \in T_1 \cup T_2$ and $t(x \leftarrow w_2) \notin T_1 \cup T_2$. Since $|R| = (m+1)(n+1)$, according to lemma 4, $B_\cup$ has at least $(m+1)(n+1) - 1$ vertical states. ■

The theorem below shows that the upper bound for the vertical states is tight.

Theorem 54. *For any two DTA(DFA)s A_1 and A_2 with m and n vertical states respectively, we have*

1. *there exists a DTA(DFA) recognizing $L(A_1) \cup L(A_2)$ with at most $(m+1)(n+1) - 1$ vertical states,*

2. *for any integers $m, n \geq 1$, there exist tree languages T_1 and T_2 such that T_1 and T_2 can be recognized by DTA(DFA)s with m and n vertical states respectively, and any DTA(DFA) recognizing $T_1 \cup T_2$ has at least $(m+1)(n+1) - 1$ vertical states.*

Proof. The upper bound follow from Lemma 52 and the lower bound is obtained from Lemma 53. ■

7.2 State complexity of intersection

We investigate the upper bounds on the numbers of vertical and horizontal states for the operation of intersection on SDTAs and DTA(DFA)s. The upper bounds for vertical states are tight for both SDTAs and DTA(DFA)s. We also get an upper bound which is almost tight for the number of the horizontal states of SDTAs.

7.2.1 Intersection of DTA(DFA)s

In this section, the upper bounds on the numbers of vertical and horizontal states for the operation of intersection on DTA(DFA)s are investigated, and followed by a matching lower bound on the number of vertical states.

Lemma 55. *Let $A_i = (Q_i, \Sigma, \delta_i, F_i)$, $i = 1, 2$, be DTA(DFA)s where each horizontal language $\delta_i(q, \sigma)$ is represented by a DFA $D^{A_i}_{q,\sigma} = (C^i_{q,\sigma}, Q_i \cup \Sigma, \gamma^i_{q,\sigma}, c^i_{q,\sigma,0}, E^i_{q,\sigma})$.*

The language $L(A_1) \cap L(A_2)$ can be recognized by a DTA(DFA) $B_\cap$ with

$$\text{size}(B_\cap) \leq [\ |Q_1| \times |Q_2|;\ |\Sigma| \times \textstyle\sum_{q \in Q_1, p \in Q_2} |D^{A_1}_{q,\sigma}| \times |D^{A_2}_{p,\sigma}|\ ].$$

Proof. Denote the complement of $D^{A_i}_{q,\sigma}$ as: $\overline{D}^{A_i}_{q,\sigma} = (\overline{C}^i_{q,\sigma}, Q_i \cup \Sigma, \overline{\gamma}^i_{q,\sigma}, \overline{c}^i_{q,\sigma,0}, \overline{E}^i_{q,\sigma})$.

Choose $B_\cap = (Q_1 \times Q_2, \Sigma, \delta, F)$, where $(q, p) \in Q_1 \times Q_2$ is final if $q \in F_1$ and $p \in F_2$.

Each horizontal language $\delta((q, p), \sigma)$ is represented by a DFA $G_{(q,p),\sigma} = ((C^1_{q,\sigma} \times C^2_{p,\sigma}), (Q_1 \times Q_2) \cup \Sigma, \mu, (c^1_{q,\sigma,0}, c^2_{p,\sigma,0}), V)$, where $(c^1, c^2) \in C^1_{q,\sigma} \times C^2_{p,\sigma}$ is final if $c^1 \in E^1_{q,\sigma}$ and $c^2 \in E^2_{p,\sigma}$. μ is defined as below:

(i) For any input $a \in \Sigma$,

$\mu((c^1, c^2), a) = (d^1, d^2)$, $d^1 = \gamma^1_{q,\sigma}(c^1, a)$, $d^2 = \gamma^2_{p,\sigma}(c^2, a)$, both $\gamma^1_{q,\sigma}(c^1, a)$ and $\gamma^2_{p,\sigma}(c^2, a)$ should be defined.

(ii) For any input $(x, y) \in Q_1 \times Q_2$,

$\mu((c^1, c^2), (x, y)) = (d^1, d^2)$, $d^1 = \gamma^1_{q,\sigma}(c^1, x)$, $d^2 = \gamma^2_{p,\sigma}(c^2, y)$, both $\gamma^1_{q,\sigma}(c^1, x)$ and $\gamma^2_{p,\sigma}(c^2, y)$ should be defined.

■

Next we state a lower bound for the number of vertical states for intersection.

Lemma 56. *For any integers m, n, there exist tree languages T_1 and T_2 such that T_1, T_2 can be recognized respectively by a DTA(DFA) with at most m and n vertical states.*

Any DTA(DFA) recognizing $T_1 \cap T_2$ needs at least mn vertical states.

Proof. Let $s \in (0+1)^*$. Consider tree languages $T_1 = \{s(a(y_i\ a\ x_j)) \mid |s|_0 = i, 1 \leq i \leq m, 1 \leq j \leq n\}$, and $T_2 = \{s(a(y_i\ a\ x_j)) \mid |s|_1 = j, 1 \leq i \leq m, 1 \leq j \leq n\}$, where y_i (or x_j) is a binary expression of number i (or j).

T_1 can be recognized by a DTA(DFA) $A_1 = (Q_1, \{a, 1, 0\}, \delta_1, F_1)$, where $Q_1 = \{q_1, q_2, \ldots, q_m\}$, $F_1 = \{q_1\}$. δ is defined as: For $1 \leq i \leq m$, $\delta(q_i, a) = y_i\ a\ x_j$, $\delta(q_i, 1) = q_i$; For $1 \leq i \leq m-1$, $\delta(q_i, 0) = q_{i+1}$. DTA(DFA) recognizing T_2 can be similarly defined with n states. $T_1 \cap T_2 = \{s(a(y_i\ a\ x_j)) \mid |s|_0 = i, |s|_1 = j, 1 \leq i \leq m, 1 \leq j \leq n\}$.

Let $B_\cap$ be an arbitrary DTA(DFA) recognizing $T_1 \cap T_2$. $B_\cap$ needs at least mn vertical states. Let $t \in T_\Sigma[x]$. Choose $R = \{a(y_i a x_j) \mid 1 \leq i \leq m, 1 \leq j \leq n\} \cup \{a(0aa0)\}$. For any two trees $w_1 = a(y_{u_1} a x_{v_1})$ and $w_2 = a(y_{u_2} a x_{v_2})$ in R, choose $t = 0^{u_1}(1^{v_1}(x))$. We have $t(x \leftarrow w_1) \in T_1 \cap T_2$ and $t(x \leftarrow w_2) \notin T_1 \cap T_2$. For any two trees $w_1 = a(y_u a x_v)$ and $w_2 = a(0aa0)$, choose $t = 0^u(1^v(x))$. We have

$t(x \leftarrow w_1) \in T_1 \cap T_2$ and $t(x \leftarrow w_2) \notin T_1 \cap T_2$. Since $|R| = mn + 1$, according to lemma 4, $B_\cap$ has at least mn vertical states. ■

The theorem below shows that the upper bound for the vertical states is tight.

Theorem 57. *For any two DTA(DFA)s A_1 and A_2 with m and n vertical states respectively, we have*

1 there exists a DTA(DFA) recognizing $L(A_1) \cap L(A_2)$ with at most mn vertical states,

2 for any integers $m, n \geq 1$, there exist tree languages T_1 and T_2 such that T_1 and T_2 can be recognized by DTA(DFA)s with m and n vertical states respectively, and any DTA(DFA) recognizing $T_1 \cap T_2$ has at least mn vertical states.

Proof. The upper bound follows from Lemma 55. The lower bound is obtained by Lemma 56. ■

7.2.2 Intersection of strongly deterministic automata

First we present upper bounds on the numbers of vertical and horizontal states for intersection of SDTAs.

Lemma 58. *Let A_1 and A_2 be two arbitrary SDTAs, $A_i = (Q_i, \Sigma, \delta_i, F_i)$, $i = 1, 2$, the transition function associated with σ is represented by a DFA $H_\sigma^{A_i} = (C_\sigma^i, Q_i \cup \Sigma, \gamma_\sigma^i, c_{\sigma,0}^i, E_\sigma^i)$ with an output function $\lambda_\sigma^{A_i}$.*

The language $L(A_1) \cap L(A_2)$ can be recognized by an SDTA $B_\cap$ with

$$\text{size}(B_\cap) \leq [\ |Q_1| \times |Q_2|;\ \sum_{\sigma \in \Sigma} |C_\sigma^1| \times |C_\sigma^2|\ ].$$

Proof. Choose $B_\cap = (Q_1 \times Q_2, \Sigma, \delta, F)$, where $(q,p) \in Q_1 \times Q_2$ is final if $q \in F_1$ and $p \in F_2$.

Horizontal language associated with each σ is represented by a DFA $H_\sigma^B = (C_\sigma^1 \times C_\sigma^2), (Q_1 \times Q_2) \cup \Sigma, \mu, (c_{\sigma,0}^1, c_{\sigma,0}^2), V)$ with an output function λ_σ^B, where $(c^1, c^2) \in C_\sigma^1 \times C_\sigma^2$ is final if $c^1 \in E_\sigma^1$ and $c^2 \in E_\sigma^2$. μ is defined as below:

(i) For any input $a \in \Sigma$,

$\mu((c^1, c^2), a) = (\gamma_\sigma^1(c^1, a), \gamma_\sigma^2(c^2, a))$, both $\gamma_\sigma^i(c^i, a)$, $i = 1, 2$, should be defined.

(ii) For any input $(x_1, x_2) \in Q_1 \times Q_2$,

$\mu((c^1, c^2), (x_1, x_2)) = (\gamma_\sigma^1(c^1, x_1), \gamma_\sigma^2(c^2, x_2))$, both $\gamma_\sigma^i(c^i, x_i)$, $i = 1, 2$, should be defined.

For any $(c^1, c^2) \in E_\sigma^1 \times E_\sigma^2$, $\lambda_\sigma^B(c^1, c^2) = (\lambda_\sigma^{A_1}(c^1), \lambda_\sigma^{A_2}(c^2))$. ■

Next we give the lower bounds on the number of vertical and horizontal states for the operation of intersection on SDTAs.

Lemma 59. *For any integers m, n and relatively prime numbers $k_1 < k_2 < \ldots < k_m < k_{m+1} < \ldots < k_{m+n}$, there exist tree languages T_1 and T_2 such that T_1, T_2 can be recognized respectively by an SDTA A_1, A_2 with* $\text{size}(A_1) \le [m;\ \prod_{i=1}^{m} k_i + 2m]$ *and* $\text{size}(A_2) \le [n;\ \prod_{i=1}^{n} k_{i+m} + 2n]$.

Any SDTA recognizing $T_1 \cap T_2$ needs at least mn vertical states and $\prod_{i=1}^{m+n} k_i$ horizontal states.

Proof. Choose $\Sigma = \{a, b, c\}$. Let $Z_1 = \prod_{i=1}^{m} k_i$, $Z = \prod_{i=1}^{m+n} k_i$. Let $s \in \Sigma^*$ and $|s|_c$ denotes the number of c's in s. We define the tree language T_1 to consist of trees $t \in \{s(b(a^k))\}$, where

(1) $\exists i : 1 \leq i \leq m, k \equiv 0 (\text{mod } k_i)$,

(2) $\forall j : 1 \leq j \leq m, j \neq i, k_j$ does not divide k,

(3) $s \in (b + c)^*, |s|_b = i - 1, |s|_c \geq 0$.

We define T_2 to consist of trees $t \in \{s(b(a^k))\}$, where

(1) $\exists j : m + 1 \leq j \leq m + n, k \equiv 0 (\text{mod } k_j)$,

(2) $\forall i : m + 1 \leq i \leq m + n, i \neq j, k_i$ does not divide k,

(3) $s \in (b + c)^*, |s|_c = j - m - 1, |s|_b \geq 0$.

The tree language T_1 can be recognized by an SDTA $A_1 = (Q_1, \{a, b, c\}, \delta_1, F_1)$, where $Q_1 = \{p_1, p_2, \ldots, p_m\}$, $F_1 = \{p_1\}$. The transition function is represented by two DFAs $H_b^{A_1}$ and $H_c^{A_1}$. $H_b^{A_1} = (C_b^1, Q_1 \cup \{a\}, \gamma_b^1, c_0^1, E_b^1)$ with an output function λ_b^1. $C_b^1 = \{c_0^1, c_1^1, \ldots, c_{Z_1}^1, d_1^1, d_2^1, \ldots, d_{m-1}^1\}$. For $0 \leq i \leq Z_1 - 1$, $\gamma_b^1(c_i^1, a) = c_{i+1}^1$, $\gamma_b^1(c_{Z_1}^1, a) = c_1^1$. For $1 \leq i \leq m - 1$, $\gamma_b^1(c_0^1, p_{i+1}) = d_i^1$. For all $1 \leq i \leq m - 1$, define $d_i^1 \in E_b^1$ and $\lambda_b^1(d_i^1) = p_i$. For k satisfies that

$$(1) \exists j : 1 \leq j \leq m, k \equiv 0 (\text{mod } k_j), \ (2) \forall i : 1 \leq i \leq m, i \neq j, k_i \text{ does not divide } k,$$

define $c_k^1 \in E_b^1$ and $\lambda_b^1(c_k^1) = p_j$.

$H_c^{A_1} = (C_c^1, Q_1, \gamma_c^1, e_0^1, E_c^1)$ with an output function λ_c^1. $C_c^1 = \{e_0^1, e_1^1, \ldots, e_m^1\}$. For $1 \leq i \leq m$, $\gamma_c^1(e_0^1, p_i) = e_i^1$. $E_c^1 = \{e_1^1, \ldots, e_m^1\}$. For $1 \leq i \leq m$, $\lambda_c(e_i^1) = p_i$.

The SDTA accepting T_2 can be defined similarly.

Let $B_\cap = (Q, \{a, b, c\}, \delta, F)$ be an arbitrary SDTA recognizing $T_1 \cap T_2$. $B_\cap$ needs at least mn vertical states. Choose $R = \{b(a^{k_x \cdot k_y}) \mid 1 \leq x \leq m, m + 1 \leq y \leq m + n\} \cup \{b(a^{Z+1})\}$. Let $t \in T_\Sigma[x]$ be arbitrary.

For any two strings $w_1 = b(a^{k_{x_1} \cdot k_{y_1}})$ and $w_2 = b(a^{k_{x_2} \cdot k_{y_2}})$ where $1 \leq x_1, x_2 \leq m, m+1 \leq y_1, y_2 \leq m+n$, choose $t = c^{y_1-m-1}b^{x_1-1}(x)$. We have $t(x \leftarrow w_1) \in T_1 \cap T_2$ and $t(x \leftarrow w_2) \notin T_1 \cap T_2$.

For any two strings $w_1 = b(a^{k_{x_1} \cdot k_y})$ and $w_2 = b(a^{Z+1})$, where $1 \leq x_1 \leq m, m+1 \leq y \leq m+n$, choose $t = c^{y-m-1}b^{x_1-1}(x)$. We have $t(x \leftarrow w_1) \in T_1 \cap T_2$ and $t(x \leftarrow w_2) \notin T_1 \cap T_2$.

Since $|R| = mn + 1$, according to Lemma 1, $B_\cap$ has at least mn vertical states.

Now we show that $B_\cap$ needs at least Z horizontal states. Choose $S = \{a^j \mid 1 \leq j \leq Z\} \cup \{bb\}$. Consider any distinct integers $1 \leq x < y \leq Z$. Since all the k_i's are relatively prime, there exists a k_i, $1 \leq i \leq m+n$, such that k_i does not divide $y - x$. Choose $0 \leq z < k_i$ such that $y + z \equiv 0 (\text{mod } k_i)$.

If $1 \leq i \leq m$, choose k_j, where $m+1 \leq j \leq m+n$. Choose $0 \leq w < k_j$ such that $w+y+z \equiv 0 (\text{mod } k_j)$. Let $t = c^{j-m-1}b^{i-1}(x)$. Now $t(x \leftarrow b(a^{y+(y+z+w)k_i-y})) \in T_1 \cap T_2$ and $t(x \leftarrow b(a^{x+(y+z+w)k_i-y})) \notin T_1 \cap T_2$. (Both k_i and k_j divide $y + (y+z+w)k_i - y$. Since k_i does not divide $x-y$, k_i does not divide $x+(y+z+w)k_i-y$. It does not matter whether k_j divides $x + (y+z+w)k_i - y$ or not.) The case when $m+1 \leq i \leq m+n$ can be similarly proved.

For any $s_1 = bb$ and $s_2 = a^j$, $1 \leq j \leq Z$, choose $t = c^{v-m-1}b^{u-1}(x)$, where $1 \leq u \leq m, m+1 \leq v \leq m+n$. We have $t(x \leftarrow b(s_2, a^{Z-j+k_u \cdot k_v})) \in T_1 \cap T_2$ and $t(x \leftarrow b(s_1, a^{Z-j+k_u \cdot k_v})) \notin T_1 \cap T_2$. According to Lemma 2, $B_\cap$ needs at least Z horizontal states. ■

The following result gives the upper bound and the lower bound for the operation of intersection for SDTAs.

Theorem 60. *For any two arbitrary SDTAs $A_i = (Q_i, \Sigma, \delta_i, F_i)$, $i = 1, 2$, whose* *transition function associated with* σ *is represented by a DFA* $H_\sigma^{A_i} = (C_\sigma^i, Q_i \cup \Sigma, \gamma_\sigma^i, c_{\sigma,0}^i, E_\sigma^i)$ *augmented with an output function* λ_σ^i*, we have*

1 *There exists an SDTA* $B_\cap$ *recognizing* $L(A_1) \cap L(A_2)$ *such that*

$$\text{size}(B_\cap) \leq [\ |Q_1| \times |Q_2|;\ \sum_{\sigma \in \Sigma} |C_\sigma^1| \times |C_\sigma^2|\ ].$$

2 *For integers* $m, n \geq 1$ *and relatively prime numbers* $k_1, k_2, \ldots, k_m, k_{m+1}, \ldots,$ k_{m+n}*, there exists tree languages* T_1 *and* T_2 *such that* T_1 *and* T_2*, respectively, can be recognized by SDTAs with* m *and* n *vertical states,* $\prod_{i=1}^{m} k_i + O(m)$ *and* $\prod_{i=1+m}^{m+n} k_i + O(n)$ *horizontal states, and any SDTA recognizing* $T_1 \cap T_2$ *has at least* mn *vertical states and* $\prod_{i=1}^{m+n} k_i$ *horizontal states.*

Proof. The upper bound follows from Lemma 58. The lower bound is obtained from Lemma 59. ■

Theorem 60 shows that for the operation of intersection on SDTAs the upper bounds are tight for vertical states and almost tight for horizontal states.

7.3 State complexity of projection

In this section, we will present a tight bound for the projection on unranked trees for the number of vertical states, which is different than the known tight bound for the projection operation on regular string languages. Before going to the upper bound constructions for the state complexity of projection, we present a technical lemma on string languages.

Lemma 61. *Let* $A = (Q, \Sigma, \delta, s, F)$ *be an arbitrary DFA, and* $\mathcal{E}(A)$ *be the set of NFAs with* ϵ*-transitions where an arbitrary number of transitions* $\delta(s_i, \sigma) = s_j$ *are replaced* *by $\delta(s_i, \sigma) = s_j$ and $\delta(s_i, \epsilon) = s_j$. An NFA in $\mathcal{E}(A)$ is obtained by adding between some, but not necessarily all, pairs of states s_i and s_j connected by a σ-transition an ϵ-transition from s_i to s_j. The corresponding σ-transition will remain also in the constructed NFA.*

Any DFA recognizing an n-state NFA in $\mathcal{E}(A)$ has at most $\frac{3}{4}2^n - 1$ states and the upper bound is tight.

Proof. Let $A \in \mathcal{E}(A)$ be an n-state NFA with only one ϵ-transition, $\delta(i, \epsilon) = j$. We can get a $(2^n - 1)$-state DFA B after the subset construction. Since the states i and j are connected by an ϵ-transition, all the strings that can reach the state i automatically reach the state j. However, all the strings that can reach the state j not necessarily reach the state i, since the ϵ-transition is going out from i to j. Thus, excluding all the states that contains i but not j, B has at most $2^n - 2^{n-2} - 1$ reachable states if we don't require the automata be complete. From the above discussion we also notice that any more ϵ-transitions added to A will result in more unreachable states in B. The above means any NFA in $\mathcal{E}(A)$ that contains more than one ϵ-transition cannot have more than $2^n - 2^{n-2} - 1$ reachable states. This means an n-state NFA $A \in \mathcal{E}(A)$ with only one ϵ-transition gives the largest state blow-up after determinizing.

Let A be the DFA shown in Figure 7.1 and A' in Figure 7.2 be the NFA with only one ϵ-transition. According to the proof given in Theorem 5 [28], any DFA for $L(A') \cap \{a, b\}^*$ needs at least $\frac{3}{4}2^n - 1$ states. The state complexity of $L(A')$ is at least the state complexity of $L(A') \cap \{a, b\}^*$. Thus, we proved what we claimed. ∎

After finishing the work, we found that [44] gives the same upper bound as Lemma 61. Also [44] gives, without proof, a conjectured lower bound construction

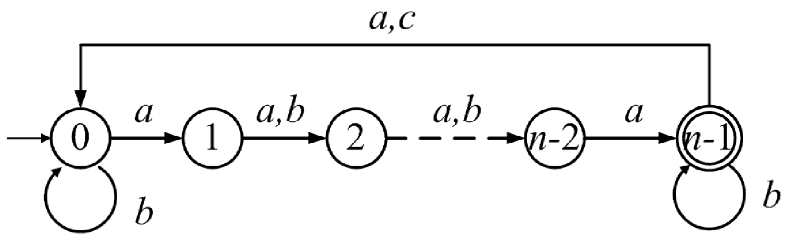

Figure 7.1: DFA A

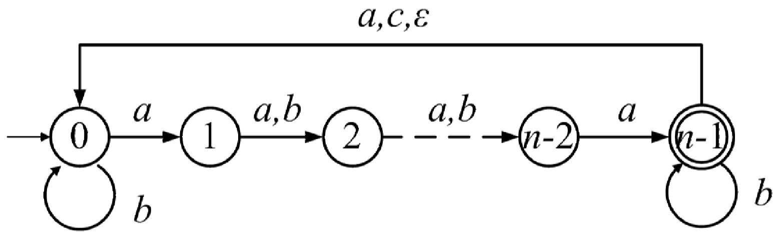

Figure 7.2: NFA A'

that uses similar (but not the same) languages as the languages used in our lower bound proof. In Lemma 61, the σ-transition from s_i to s_j is replaced by two transitions from s_i to s_j labeled by σ and ϵ, respectively, and not necessarily all pairs of states s_i and s_j connected by a σ-transition from s_i to s_j are replaced. In [44], all pairs of states s_i and s_j connected by a σ-transition are replaced by only one transition labeled by ϵ from s_i to s_j.

7.3.1 Upper bound for projection on unranked trees

We recall that the projection operation is extended to trees by deleting from a given tree all subtrees where the root is labeled by an unobservable symbol, and the definition relies on the fact that we are dealing with unranked trees. We present two upper bound constructions that give the same state complexity of projection operation on unranked trees.

Construction 1

Since the subtrees rooted at the unobservable symbols are deleted from the original trees, the states that associated only with unobservable symbols are not assigned to any node in the vertical computation. Thus, all such vertical states and the corresponding horizontal languages should be deleted after the projection. The states that can be assigned to both observable and unobservable symbols may arbitrarily be replaced by ϵ in horizontal languages, as they may correspond to nodes that are labeled by unobservable symbols and deleted after projection.

Consider a DTA(DFA) $A = (Q, \Sigma, \delta, F)$, where the horizontal language associated with $\sigma \in \Sigma$ and $q \in Q$ is recognized by a DFA $H^A_{\sigma,q}$ with size $m_{\sigma,q}$. Let $\Sigma_P \subseteq \Sigma$. We can construct a NTA(NFA) $B = (Q, \Sigma_P, \gamma, F)$ recognizing $\mathcal{M}(L(A))$ from A as follows. Recall that the (ϵ, Δ)-substitution on a regular language L, denoted as $\xi_\Delta(L)$, is defined in Section 5.2.

- Let $P \subseteq Q$ consists of all the states in A that associated with the symbols in Σ/Σ_P. That is P consists of $p \in P$ and $\delta(p, \alpha) \neq \emptyset$ for $\alpha \in \Sigma/\Sigma_P$.

- For the symbol α in Σ/Σ_P, define $\gamma(q, \alpha) = \emptyset$ for all $q \in Q$.

- For the symbol $\beta \in \Sigma_P$, define $\gamma(q, \beta) = \xi_P(\delta(q, \beta))$.

The horizontal NFA associated with q and β has the same size as $H^A_{\beta,q}$ as the construction of B adds only ϵ-transitions to $H^A_{\beta,q}$. To convert the NTA(NFA) B to a DTA(DFA) C, a powerset construction is applied to the vertical states of B and $\mathcal{P}(Q)$ is the set of states in C. As we do not require tree automata to be complete, $\emptyset$ is omitted from the set of states in C. For the horizontal states, the horizontal DFA in C must simulate all the NFAs in B to get disjoint horizontal languages. (The details of determinizing an NTA(NFA) please refer to Lemma 30 in Section 4.2.2.) According to Lemma 61, the size of a horizontal DFA associated with a state in $\mathcal{P}(Q)$ and β in C is at most $\prod_{q\in Q}(\frac{3}{4}2^{m_{\beta,q}}-1)$. Thus, the size of C is

$$size(C) \leq [\ 2^{|Q|}-1;\ 2^{|Q|}\cdot(\sum_{\beta\in\Sigma_P}\prod_{q\in Q}(\frac{3}{4}2^{m_{\beta,q}}-1))\ ].$$

Construction 2

Compared with Construction 1 where the states associated with both observable and unobservable symbols can be arbitrarily replaced by ϵ, here we give a different construction where two types of states are used in the vertical computation, the states that are assigned to only unobservable symbols and the states that are assigned to only observable ones. Then it is fine to replace all the states associated with unobservable symbols with ϵ after the projection. The construction doubles the number of the vertical states at the beginning, however, at the end it gives the same upper bound as Construction 1.

Given $\Sigma_P \subseteq \Sigma$ and a DTA(DFA) $A = (Q, \Sigma, \delta, F)$, where the horizontal language associated with $\sigma \in \Sigma$ and $q \in Q$ is recognized by a DFA $H^A_{\sigma,q}$ with size $m_{\sigma,q}$, we can construct an equivalent NTA(NFA) $B = (P, \Sigma_P, \gamma, F)$ as follows.

- For each state $q \in Q$ associated with both symbols in Σ/Σ_P and Σ_P in A, $q^{\Sigma/\Sigma_P} \in P$ and $q^{\Sigma_P} \in P$.
- For each state $q \in Q$ associated with only symbols in Σ_P in A, $q^{\Sigma_P} \in P$.

For the state q associated with both symbols in Σ/Σ_P and Σ_P in A, we replace q by two states q^{Σ/Σ_P} and q^{Σ_P} in B. For the state that associates only with Σ_P in A, we keep it the same in B. Then we modify the transition functions as follows.

- First for all horizontal languages $\delta(p, \sigma) = L$, $p \in Q$, replace each occurrence of q in L by $q^{\Sigma/\Sigma_P} + q^{\Sigma_P}$. Denote by $\delta'(p, \sigma) = L'$ the modified horizontal languages.

- Then for each the horizontal language $\delta'(p, \sigma) = L'$, where p is associated with both symbols in Σ/Σ_P and Σ_P in A, define $\gamma(p^{\Sigma/\Sigma_P}, \sigma) = L'$ and $\gamma(p^{\Sigma_P}, \sigma) = L'$.

- For each the horizontal language $\delta'(p, \sigma) = L'$, where p is associated with only symbols in Σ_P in A, define $\gamma(p^{\Sigma_P}, \sigma) = L'$.

It is easy to verify that B is equivalent to A. In B the state p^{Σ/Σ_P} is associated only with symbols in Σ/Σ_P and p^{Σ_P} is associated only with symbols in Σ_P. Thus, B at most doubles the number of the vertical states of A. The horizontal language L' can be recognized by a DFA constructed from $H^A_{\sigma,q}$ by replacing each transition labeled with q by a transition labeled by q^{Σ_P} and q^{Σ/Σ_P}. Thus, the size of the horizontal DFA associated with q^{Σ_P}, σ is the same as the DFA associated with q, σ. That is $m_{\sigma,q}$.

Now we construct an NTA(NFA) C recognizing $\mathcal{M}(L(A))$ from B as follows.

Step 1 Deleting all the horizontal languages associated with q^{Σ/Σ_P},

Step 2 Replace by ϵ each occurrence of q^{Σ/Σ_P} in the remaining horizontal languages.

Since all the horizontal languages associated with q^{Σ/Σ_P} are deleted, C has Q vertical states. Since step 2 is a natural projection on the horizontal languages, each DFA recognizing projected language can have at most $\frac{3}{4}2^{m_{\sigma,q}} - 1$ states, according to the upper bound of natural projection on string languages [28, 63].

To convert the NTA(NFA) C to a DTA(DFA) D, a powerset construction is applied to the vertical states of C and $\mathcal{P}(Q)$ is the set of states in D. As we do not require tree automata to be complete, $\emptyset$ is omitted from the set of states in D. For the horizontal states, the horizontal DFA in D must simulate all the NFAs in C to get disjoint horizontal languages. (The details of determinizing an NTA(NFA) please refer to Lemma 30 in Section 4.2.2.) The size of a horizontal DFA associated with a state in $\mathcal{P}(Q)$ and β in D is at most $\prod_{q\in Q}(\frac{3}{4}2^{m_{\beta,q}}-1)$. Thus, the size of D is at most

$$size(D) \leq [\ 2^{|Q|}-1;\ 2^{|Q|}\cdot(\sum_{\beta\in\Sigma_P}\prod_{q\in Q}(\frac{3}{4}2^{m_{\beta,q}}-1))\ ].$$

7.3.2 Lower bound on vertical states

Next we present a tight lower bound for the number of vertical states, which is different than the known tight bound for the projection operation on regular string languages.

Let $p_1,\ldots,p_n$ be first primes, $\Sigma=\{a,b,c\}$ and $\Sigma_P=\{a,b\}$. Consider the tree language T consisting of trees

$$\{a^i(c^ib^k) \mid 1\leq i\leq n,\ k\geq 0,\ (\exists 1\leq j\leq n)[k\equiv 0 \pmod{p_j} \text{ and } i\equiv j \pmod{n}]\}.$$

T can be recognized by a DTA(DFA) $A=(\{q_1,\ldots,q_n,q_c,q_b\},\Sigma,\delta,\{q_1\})$ where

- $\delta(q_c,c)=\epsilon$,
- $\delta(q_b,b)=\epsilon$,
- $\delta(q_i,a)=q_c^i(q_b^{p_i})^*$, $1\leq i\leq n$.

The prefix q_c^i in the horizontal languages are used to make the horizontal languages disjoint. After the projection, all nodes labeled by c are deleted. Then $\mathcal{M}(T)$ consists of trees

$$\{a^i(b^k) \mid i\geq 1,\ k\geq 0,\ (\exists 1\leq j\leq n)[k\equiv 0 \pmod{p_j} \text{ and } i\equiv j \pmod{n}]\}.$$

According to Theorem 31 in Section 4.2.2, any DTA(DFA) recognizing $\mathcal{M}(T)$ has at least $2^n - 1$ vertical states.

7.4 State complexity of quotient operations

In this section we establish upper bounds for the state complexity of top-quotient and bottom-quotient operations that are tight for the numbers of vertical states. The state complexity bound for bottom-quotient is of a different order of magnitude than the corresponding state complexity bound for left-quotient of string languages.

We start with top-quotient operation.

Theorem 62. *For any integer $n \geq 1$, n vertical states are necessary and sufficient in the worst case for a deterministic unranked tree automaton to accept the top-quotient of the tree language recognized by an n vertical states deterministic unranked tree automaton with respect to an arbitrary tree language T'.*

Proof. Let $A = (Q, \Sigma, \delta, F)$ where $|Q| = n$, be an arbitrary deterministic unranked tree automaton. We can construct a deterministic unranked tree automaton $B = (Q, \Sigma, \delta, E)$ recognizing $T' \top_b L(A)$, where

$$E = \{q \in Q \mid (\exists t' \in T')(\exists r \in [t'(b \leftarrow q)])\ r^A \in F\}.$$

Next we show that n states are necessary. Let $T' = \{b\}$ and $A = (\{q_1, q_2, \ldots, q_n\}, \{a\}, \delta, \{q_1\})$, where δ is defined as:

- for $1 \leq i \leq n-1$, $\delta(q_i, a) = q_{i+1}$,
- $\delta(q_n, a) = \epsilon$.

A accepts the unary tree language a^n. We have $T' \top L(A) = L(A)$. ■

We present the following theorem for bottom-quotient operation on unranked tree automata.

Theorem 63. *For any integer $n \geq 1$, $(n+1)2^n - 1$ vertical states are necessary and sufficient in the worst case for a deterministic unranked tree automaton to accept the bottom-quotient of the tree language recognized by an n vertical states deterministic unranked tree automaton with respect to an arbitrary tree language T'.*

We prove Theorem 63 by stating two lemmas. Lemma 64 establishes the upper bound and Lemma 65 establishes a lower bound that differs by the constant one. Finally, after the proof of Lemma 65 we explain how the proof of Lemma 65 can be modified to exactly reach the upper bound given in Lemma 64.

Lemma 64. *For an arbitrary deterministic unranked tree automaton A with n vertical states and an arbitrary tree language T', any deterministic unranked tree automaton recognizing $L(A) \perp_b T'$ needs at most $(n+1)2^n - 1$ vertical states.*

Proof. Let $A = (Q, \Sigma, \delta, F)$ where $|Q| = n$, be an arbitrary deterministic unranked tree automaton. Define $G = \{t'^A \mid t' \in T'\}$. G consists of all the states that are assigned to the roots of trees in T' by A. We can construct a deterministic unranked tree automaton $B = (P, \Sigma, \mu, E)$ recognizing $L(A) \perp_b T'$, where

- $P = \{(q, \mathcal{P}(Q)) \mid q \in Q \cup \{dead\}\}$,
- $(q, S) \in E$ if $S \cap F \neq \emptyset$.

B operates as follows. For a leaf node u labeled by σ,

- assign $(q, \emptyset)$ to u if $\sigma \neq b$ and $\epsilon \in \delta(q, \sigma)$,

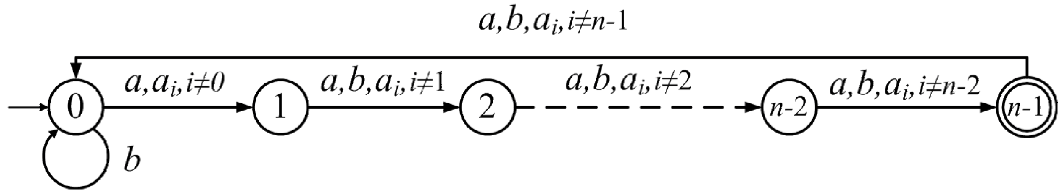

Figure 7.3: The DFA A

- assign (q, G) to u if $\sigma = b$ and $\epsilon \in \delta(q, b)$,

- assign $(dead, G)$ to u if $\sigma = b$ and $\epsilon \notin \delta(q, b)$.

For a tree $\sigma(\sigma_1, \ldots, \sigma_m)$ where the leaf nodes are assigned with a sequence of states $(q_1, S_1), \ldots, (q_m, S_m)$, the root node is assigned with state (q, S) provided that $q_1 \ldots q_m \in \delta(q, \sigma)$ and S consists of all $s \in Q$ such that for some $1 \leq i \leq m$

$$(\exists s_i \in S_i) \text{ such that } q_1 \ldots q_{i-1} s_i q_{i+1} \ldots q_m \in \delta(s, \sigma).$$

We define $q = dead$ if $q_1 \ldots q_m$ is not included in any $\delta(s, \sigma)$, $s \in Q$.

In states of B the first component directly simulates the computation of A. In a state (q, S), the second component keeps track of all states that A may be in provided that exactly one leaf labeled by b below the current node was replaced by a tree of T'.

Thus, the upper bound on the number of vertical states is $(n+1)2^n - 1$ as we do not require the tree automata to be complete. ■

Next we state a lower bound on the number of vertical states.

Lemma 65. *There exists a tree language T that can be recognized by a deterministic unranked tree automaton with n vertical states, such that any deterministic unranked tree automaton recognizing $T \perp_a T_\Sigma$ needs at least $(n+1)2^n - 2$ vertical states.*

Proof. Shown in Figure 7.3, A is the DFA we modified from the one used in Theorem 5 of [66] by adding symbols $a_0, \ldots, a_{n-1}$. Based on the DFA A we define the tree language T used in our lower bound construction. The tree language T consists of Σ-labeled trees t, $\Sigma = \{a, b, c\}$, where:

(i) All leaves are labeled by a and if a node u has a child that is a leaf, then all the children of u are leaves.

(ii) A accepts the string of symbols labeling a path from any node of height one to the root.

(iii) The following holds for any $u \in \mathrm{dom}(t)$ and any nodes v_1 and v_2 of height one below u. If w_i is the string of symbols labeling the path from v_i to u, $i = 1, 2$, then A reaches the same state after reading strings w_1 and w_2.

The computation starts from nodes of height one. This is done because when the bottom-quotient takes place, the leaf node is substituted by a tree, and it will not appear in the resulting tree.

T can be recognized by a deterministic tree automaton
$M = (Q, \{a, b, a_0, \ldots, a_{n-1}\}, \delta, F)$ where $Q = \{0, 1, \ldots, n-1\}$ and $F = \{n-1\}$. The transition function is defined as:

(1) $\delta(0, a) = \epsilon \cup (n-1)^+$,

(2) $\delta(i, a) = (i-1)^+$ for $1 \leq i \leq n-1$,

(3) $\delta(j, b) = (j-1)^+$ for $1 \leq i \leq n-1$,

(4) $\delta(0, b) = \{n-1, 0\}^+$,

(5) $\delta(i, a_j) = (i-1)^+$ for $1 \leq i \leq n-1$ and $i \neq j$,

(6) $\delta(0, a_j) = (n-1)^+$ for $j \neq n-1$.

Let $T' = T_\Sigma$ and N be the deterministic tree automaton for $L(M) \perp T'$ as considered in Lemma 64. According to the upper bound construction, the states in N are of the form $(q, S), 0 \leq q \leq n, S \subseteq \{0, 1, \ldots, n-1\}$, where $q = n$ denotes *dead* state in the construction.

First we show every state is reachable except $(n, \{0, 1, \ldots, n-1\})$. The leaf node labeled with a is assigned with $(0, \{0, 1, \ldots, n-1\})$. The state $(i, \{0, 1, \ldots, n-1\})$, $0 \leq i \leq n-1$ is reachable after reading a unary tree a^i. For $S \subseteq Q$, $0 \leq i \leq n-1$, $i \in S$, (i, S) is reachable from state (i, Q) by reading $s_0 s_{n-1} \ldots s_1$ where for each $0 \leq j \leq n-1$, $s_j = a$ if $j \in S$ and $s_j = b$, otherwise.

For $i \notin S$, we show that the state (i, S) is reachable using decreasing induction on the size of S. First we note that for $S \subseteq Q$, $0 \leq i \leq n-1$, $i \notin S$, $(i, Q - \{i\})$ is reachable from state $(i-1, Q)$ by reading a_i. Now suppose $(s_i, Q - \{s_1, \ldots, s_k\})$, $1 \leq i \leq k$ is reachable. Let $p = (s_j, Q - \{s_1, \ldots, s_{k+1}\})$, $1 \leq j \leq k+1$, $s_1 < \ldots < s_{k+1}$ and $Q - \{s_1, \ldots, s_{k+1}\} = \{r_1, \ldots, r_{n-k-1}\}$ be an arbitrary state. p is reachable from $(s_j - 1, \{r_1 - 1, r_2 - 1, \ldots, r_{n-k-1} - 1, s'\})$, by reading $a_{s'}$, where $s' \notin \{s_j - 1, r_1 - 1, r_2 - 1, \ldots, r_{n-k-1} - 1\}$.

We have already shown that $(0, \{0, 1, \ldots, n-1\})$ is reachable. State $(n, \{1, \ldots, n-1\})$ is reachable by reading a_0. State $(n, \{0, 1, \ldots, n-1\} - \{i\})$, $1 \leq i \leq n-1$ is reachable after reading i $a's$. Assume all states (n, S), $n > |S| \geq k+1$, $k < n-1$ are reachable. We will inductively show that all states where $|S| = k$ are reachable. Let (S, n) where $S = \{s_1, s_2, \ldots, s_k\}$, $0 \leq s_1 < s_2 < \ldots < s_k \leq n-1$ be an arbitrary state where $|S| = k$. According to the inductive assumption, state $(S \cup \{0\}, n)$ is reachable. State (S, n) is reachable by reading a_0. The state $(n, \emptyset)$ is reached at the root of tree $a(u, v)$ where u and v are assigned with $(1, \{1\})$ and $(2, \{2\})$, respectively.

That is all the states except $(n, \{0, 1, \ldots, n-1\})$ are reachable.

Now we show that all states are pairwise inequivalent. Given two states (i, S_1) and (j, S_2), first we consider the case when $S_1 \neq S_2$. Without loss of generality, let $s \in S_1 - S_2$. After reading unary tree a^{n-1-s}, a final state is reached from (i, S_1) and no final state is reached from (j, S_2). Now we consider the case when $i \neq j$ and $S_1 = S_2 = S$. Suppose (i, S) and (j, S) are assigned to the roots of tree t_1 and t_2, respectively. Consider tree $a^{n-2-i}(a(u, x), v)$ where u, v are assigned with states $(i, \{i\})$ and $(n, \{i+1\})$, respectively. Now $(i+1, \{i+1\})$ is assigned at the root of $a(u, x \leftarrow t_1)$ and $(n, \{i+2\})$ is assigned at the root of $a(a(u, x \leftarrow t_1), v)$. After reading the rest of a's, a final state $(n, \{n-1\})$ is reached. On the other hand, if $i \in S$, $(n, \{i+1\})$ is assigned at the root of $a(u, x \leftarrow t_2)$, $(n, \emptyset)$ if $i \notin S$. In either case $(n, \emptyset)$ is assigned at the root of $a(a(u, x \leftarrow t_1), v)$. $(n, \emptyset)$ is a dead state, which means no final state can be reached. ■

In fact, state $(n, \{0, 1, \ldots, n-1\})$ is reachable by introducing a new symbol and a few transitions. Let T be the tree language defined in Lemma 65 and M is the deterministic tree automaton constructed for T in the proof of Lemma 65. Let d be a new symbol not in Σ. Based on M we define a tree automaton $M' = (Q, \Sigma \cup \{d\}, \delta', F)$ where δ' for elements of Σ is defined as δ and

(1) $\delta(1, d) = 11$,

(2) $\delta(2, d) = 22$,

(3) $\delta(i, d) = 1i$ for $3 \leq i \leq n$.

The symbol d and the transitions associated with it could also be defined directly on the tree language T used in the proof of Lemma 65. However, this would make the definition of T very complicated and unreadable. Thus, we only add the new symbol and the transitions to the tree automaton M.

The node labeled by d has two branches. If the root of the left branch is assigned with state 1 by M', then the node labeled by d is assigned with the same state (except state 2) as the root of its right branch. If the root of the left branch is assigned with state 2 by M', then the root of its right branch must also be assigned with state 2, and so is the node labeled by d. The sequence of states 12 is not defined at the node labeled by d. Now let $T' = T_{\Sigma \cup \{d\}}$. Consider tree $t = d(t_1, t_2)$ where the roots of t_1 and t_2 are assigned with state $(1, \{1, 2\})$ and $(2, \{1, 2, \ldots, n\})$, respectively. According to the above definition, state $(n, \{0, 1, \ldots, n-1\})$ is assigned to the root of the tree t.

We get a tight bound for the state complexity of top-quotient and bottom-quotient, respectively. Note that the bound for bottom-quotient differs, roughly, by a multiplicative factor $n+1$ from the corresponding result for ordinary finite automata. The precise state complexity of left-quotient for ordinary finite automata is $2^n - 1$ [66].

Chapter 8

Tree homomorphisms for unranked trees

In this chapter we introduce tree homomorphism defined on unranked trees. Recall that, in the case of ranked trees, h_m, $m \geq 0$ is a mapping that associates a tree $h_m(\sigma)$ to each symbol $\sigma \in \Sigma_m$, and a tree homomorphism $h : F_\Sigma \to F_\Omega$ determined by h_m, $m \geq 0$ is defined as follows:

- $h(a) = h_0(a) \in F_\Omega$ for $a \in \Sigma_0$,
- $h(f(t_1, \ldots, t_m)) = h_m(f)(x_1 \leftarrow h(t_1), \ldots, x_m \leftarrow h(t_m))$.

Roughly speaking, the tree homomorphism h replaces each node u labeled by σ with $h_m(\sigma)$ where an occurrence of the variable x_i is replaced by $h(t_i)$, if t_i is the subtree corresponding to the ith child of u, $i = 1, \ldots, m$.

When dealing with unranked alphabets, the tree $t_{\sigma,m}$ with m variables that is used to replace an occurrence of a fork $\sigma(x_1, \ldots, x_m)$, $m \geq 1$, where σ is an alphabet symbol, should depend on the number of children of the node labeled by σ, that is on m, where there is no *a priori* bound on m. For technical reasons $t_{\sigma,m}$ is made to depend also on the labels of the children of the node labeled by σ, that is, the symbols labeling the roots of the trees corresponding to the variables x_i, $i = 1, \ldots, m$. The tree $t_{\sigma,m}$ is output by a string-to-tree transducer that receives as input the string consisting of the node labels of the children of a node labeled by σ.

We concentrate on mappings that preserve recognizability of tree languages and consider only linear tree homomorphisms. In the case of unranked trees additional conditions are needed to guarantee that a tree homomorphism preserves recognizability, and arbitrary permutations of the sequence of subtrees cannot be allowed (see Example 66 in Section 8.1.2). We need to introduce additional restrictions for the string-to-tree transducers in order to guarantee that the defined mappings preserve recognizability.

On the other hand, we attempt to define tree homomorphisms that are as general as possible, while the mappings preserve recognizability of tree languages. In particular, for sufficiently large m, we want that trees $t_{\sigma,m}$ can include any unranked tree, subject to the condition that any variable can occur only once.

To satisfy these conflicting requirements we introduce string-to-tree transducers that build an output tree in left-to-right depth-first order. Note that the output trees need to contain variables from a potentially infinite set, however, the transitions of the transducer need to have a finitary definition. To solve this issue, variables are stored in a finite number of memory locations during the computation, however, the transducer is not able to distinguish between the names of the variables and refers to the variable only by its location in the memory.

8.1 Definitions

Before introducing the definition of tree homomorphisms on unranked trees, we give some notations that are used only in this chapter.

In the following $X = \{x_1, x_2, x_3, \ldots\}$ is always a countable set of variable names. We denote $X_m = \{x_1, \ldots, x_m\}$, $m \geq 0$. For $n \in \mathbb{N}$, we denote $[n] = \{1, \ldots, n\}$ and $[n]_0 = \{0, 1, \ldots, n\}$.

Let Σ be a finite alphabet and S a set. The set of Σ-labeled unranked trees where leaves can be labeled also by elements of S is denoted $T_\Sigma(S)$ and $T_\Sigma = T_\Sigma(\emptyset)$.

For $t, t_1, \ldots, t_k \in T_\Sigma$, and pairwise distinct $i_1, \ldots, i_k \in \mathbb{N}$, $t[x_{i_1} \leftarrow t_1, \ldots, x_{i_k} \leftarrow t_k]$ is the tree obtained from t by replacing all occurrences of the variable x_{i_j} by t_j, $j = 1, \ldots, k$.

For $t \in T_\Sigma$ and $r \in T_\Sigma(X_m)$, we say that r is an *inside subtree* of t if there exist $r_1, \ldots, r_m \in T_\Sigma$ such that $r[x_1 \leftarrow r_1, \ldots, x_m \leftarrow r_m]$ is a subtree of t. Intuitively, an inside subtree of t consists of the "top part" of some subtree of t.

8.1.1 String-to-tree transducers

Here we define a string-to-tree transducer model that will be used to produce the trees defining the tree homomorphisms in the next section.

Definition 3. *A* deterministic string-to-tree transducer with k memory locations, *$k \geq 0$, or k-*transducer, *for short, is a tuple $M = (\Sigma, \Omega, Q, q_0, R)$, where Σ is a finite alphabet of input symbols not containing \$, Ω is a finite alphabet of output symbols, Q is a finite set of states, $q_0 \in Q$ is the start state and R has for each pair (q, η), $q \in Q$, $\eta \in \Sigma \cup \{\$\}$, at most one rule of one of the following types.*

Store move: $(q, \eta) \rightarrow (\mathtt{store}, i, q')$*, where* $q, q' \in Q$*,* $\eta \in \Sigma \cup \{\$\}$*,* $i \in [k]_0$*.*

Add leaf: $(q, \eta) \rightarrow (\mathtt{leaf}, z, q')$*, where* $q, q' \in Q$*,* $\eta \in \Sigma \cup \{\$\}$*,* $z \in \Omega \cup [k]$*.*

Down move: $(q, \eta) \rightarrow (\mathtt{down}, \sigma, q')$*, where* $q, q' \in Q$*,* $\eta \in \Sigma \cup \{\$\}$*,* $\sigma \in \Omega$*.*

Up move: $(q, \eta) \rightarrow (\mathtt{up}, q')$*, where* $q, q' \in Q$ *and* $\eta \in \Sigma \cup \{\$\}$*.*

We list a brief intuitive description for each of the rules in the above definition. The more precise meaning of the rules becomes clear later from the definition of the computations of a k-transducer. For each of the types of rules, the state changes from q to q'. The store rule consumes the current input symbol η and stores in the ith memory location the index of the current input symbol (that is, a variable corresponding to this position). If $i = 0$, then the current input symbol is simply discarded. The "add leaf" rule inserts a new node u as the left sibling of the state. If $z \in \Omega$, the node u is labeled by z. If $z \in [k]$, the node u will labeled by x_j, where $j \geq 1$ is the contents of the memory location z and the new contents of the memory location z will be 0. The *down* move replaces the state by label σ and the new state will be the only child of σ. The *up* move deletes the current state node and places the new state as the right sibling of its parent.

A k-transducer M processes an input string $b_1 b_2 \cdots b_m \$$, $b_i \in \Sigma$, $i = 1, \ldots, m$, $m \geq 0$, where, informally, the b_i's correspond to the symbols labeling the sequence of children of a node and as output M produces a tree in $T_\Omega(X_m)$. The symbol \$ is used as an end marker. The computation keeps track of the position i, $1 \leq i \leq m$ (since the output has to contain variables indexed by any $i = 1, \ldots, m$), in the input, however, the actual transitions of M do not see the index i and depend only on the label b_i. The transducer M has k memory locations, where each memory location can store an index i, $1 \leq i \leq m$, that is used to represent a variable x_i.

More formally, a *configuration* of a k-transducer $M = (\Sigma, \Omega, Q, q_0, R)$ on input $b_1 \cdots b_m$, $b_i \in \Sigma$, $i = 1, \ldots, m$, $m \geq 1$, is a triple

$$C = (t_q, \overline{s}, (b_j, j)(b_{j+1}, j+1) \cdots (b_m, m)\$), \tag{8.1}$$

where t_q is a tree of $T_\Omega(X_{j-1})$ where the rightmost leaf is labeled by an element of $q \in Q$, $\overline{s} = (s_1, \ldots, s_k)$, $s_r \in [j-1]_0$, $r = 1, \ldots, k$, $1 \leq j \leq m+1$ is the current contents of the k memory cells and $b_j \cdots b_m$ is the remaining input. Intuitively, a positive value s_r indicates that the variable x_{s_r} is stored in the rth memory location and $s_r = 0$ indicates that the rth memory location is empty. The *initial configuration* of M on input $b_1 \cdots b_m$ is $C_{\text{init}}(b_1 \cdots b_m) = (q_0, \{0\}^k, (b_1, 1) \cdots (b_m, m)\$)$.

The next configuration following C in one computation step of M, denoted $C \vdash_M C'$, is the configuration that is obtained from C by the rule of M that is determined by the pair (q, b_j). Note that since M is deterministic it has at most one rule for the pair (q, b_j); if no such rule exists the computation cannot continue from C.

(i) If the right side of the rule for (q, b_j) is (store, i, q'), $i \geq 1$, C' is

$$(t_{q'}, (s_1, \ldots s_{i-1}, j, s_{i+1}, \ldots, s_k), (b_{j+1}, j+1) \cdots (b_m, m)\$),$$

where $t_{q'}$ is obtained from t_q by relabeling the rightmost leaf with q'. This operation advances the input one step and stores the index j in the ith memory location. A possible non-zero value that existed in the ith memory location previously is discarded.

If the right side of the rule for (q, b_j) is $(\text{store}, 0, q')$, C' is the configuration $(t_{q'}, \overline{s}, (b_{j+1}, j+1) \cdots (b_m, m)\$)$, that is, the input is advanced by one step. Note that if $k = 0$, all store rules are of this form.

(ii) Suppose that the right side of the rule for (q, b_j) is (leaf, z, q'). In this case the computation step depends on the type of z.

(iia) If $z \in \Omega$, then C' is the configuration $(t_{z,q'}, \overline{s}, (b_j, j) \cdots (b_m, m)\$)$, where $t_{z,q'}$ is the tree obtained from t_q by inserting a left sibling labeled by z for the right most leaf and relabeling the rightmost leaf by q'.

(iib) If $z \in [k]$ and $s_z \geq 1$, then

$$C' = (t_{x_{s_z},q'}, (s_1, \ldots, s_{z-1}, 0, s_{z+1}, s_k), (b_j, j) \cdots (b_m, m)\$),$$

where $t_{x_{s_z}}$ is the tree obtained from t_q by inserting a left sibling labeled by x_{s_z} for the right most leaf and relabeling the rightmost leaf by q'.

(iii) If the right side of the rule for (q, b_j) is $(\text{down}, \sigma, q')$, C' is the configuration $(t_{\sigma(q')}, \overline{s}, (b_j, j) \cdots (b_m, m)\$)$, where $t_{\sigma(q')}$ is the tree obtained from t_q by replacing the rightmost leaf q by the height one subtree $\sigma(q')$.

(iv) If the right side of the rule for (q, b_j) is (up, q') the configuration C' is $(t_{q'}, \overline{s}, (b_j, j) \cdots (b_m, m)\$)$, where $t_{q'}$ is the tree obtained from t_q by deleting the leaf u labeled by q and creating a new leaf u' as the right sibling of the parent of u. In the case where the parent of u is the root of t_q (and hence the parent cannot have siblings), the operation instead relabels u by q'.

(v) If $j = m + 1$, that is the computation has reached the end-marker \$, the next configuration C' is the tree $t' \in T_\Omega(X_m)$ obtained from t_q by deleting the leaf labeled by q.

Figure 8.1 illustrates the computation steps of (iib) and (iii). In the figure, (a) is the output tree (i.e., the first component of the configuration) before the computation

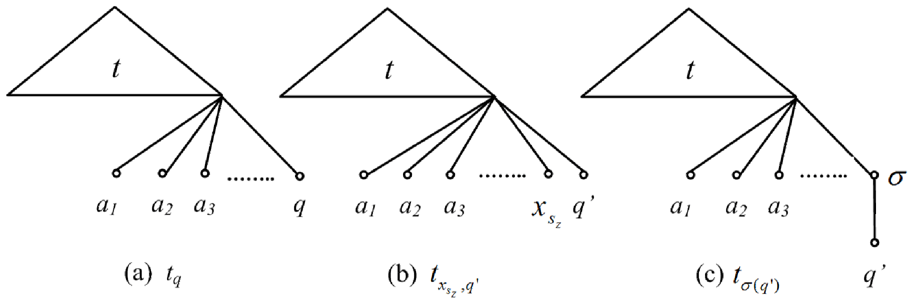

Figure 8.1: Computation steps (iib) and (iii) of a k-transducer.

step, (b) is the output tree after a step that adds a leaf labeled by a variable (as in (iib)), and (c) is the output tree after a *down* move (as in (iii)).

The *output* of M on input $b_1 \cdots b_m$ is the tree $t_{\text{out}} \in T_\Omega(X_m)$ such that $C_{\text{init}}(b_1 \cdots b_m) \vdash_M^* t_{\text{out}}$. Note that since the operation of M is deterministic, the tree t_{out} is unique, if it exists. The output of M on input $b_1 \cdots b_m$ is denoted $M(b_1 \cdots b_m)$.

Intuitively, a k-transducer produces the output tree in left-to-right depth-first order. The rules (i) consume the next input symbol and store it at one of the k memory locations. None of the other rules consume the input.

The rules (ii) add a new leaf symbol to the tree that is labeled by a symbol from Ω or by a variable taken from one of the memory locations. It should be noted that the set of variables is potentially infinite, however, the operation of M has a finitary definition – the machine M cannot distinguish between different variable names stored in the memory locations, and the rules (ii) refer to the variable only by its location in the memory. Note that in (iib) an error situation is produced if the rule refers to an empty memory location, however, this situation can be prevented by keeping track in the finite-state memory of the empty memory locations.

Finally, the down-rules (iii) label the current node by a symbol of Ω and start producing children for the current node. The up-rules (iv) move to the right-sibling of the parent of the current node and produce new siblings for the parent in left-to-right order. Since the transducer has no way of keeping track of the position of the state, representing the "output head" walking in the tree, in the special case where output head has reached the root an up-move has no effect.

In cases where the output alphabet is the same as the input alphabet, a k-transducer is denoted simply as a four-tuple (Σ, Q, q_0, R) where Σ is both the input and the output alphabet.

8.1.2 Tree homomorphisms on unranked trees

We define a *tree homomorphism* by associating to each fork $f = \sigma_0(\sigma_1, \ldots, \sigma_m)$, $m \geq 0$, a tree $t_f \in T_\Sigma(X_m)$, and then in the global tree the fork f is replaced by t_f. Note that in the usual definition of tree homomorphism for ranked trees [14], the tree t_f depends only on the symbol σ_0 labeling the root of the fork. When dealing with unranked alphabets, it is natural to require that t_f may depend on the unbounded number m of children. In our model, we define the tree t_f using a k-transducer M_{σ_0} that receives the sequence $\sigma_1 \cdots \sigma_m$ as input. Thus, the tree t_f is completely determined by the $(m+1)$-tuple of elements of Σ, $(\sigma_0, \sigma_1, \ldots, \sigma_m)$.

It is clear that in any reasonable definition of tree homomorphisms for unranked trees, t_f has to depend on m, because otherwise in the trees obtained as images of a tree homomorphism the number of children of a given node would be bounded.

Another possibility would be to define t_f as the output of a k-transducer M_{σ_0} when it receives a unary string of length m as input. We feel that this definition would be too restrictive because, when m is greater than the number of states of M_{σ_0}, the computation would necessarily enter into a loop and, consequently, M_{σ_0} could output only very particular types of trees of size greater than this constant. This can be compared with Proposition 69 below.

Definition 4. *Let Σ and Ω be finite alphabets and for each $\sigma \in \Sigma$, M_σ is a k-transducer with input alphabet Σ and output alphabet Ω, $k \geq 0$.*

*A (Σ, Ω, k)-*mapping family *is a collection of mappings $h_m : \Sigma^{m+1} \to T_\Omega(X_m)$, $m \geq 0$, defined by the k-transducers $(M_\sigma)_{\sigma \in \Sigma}$ by setting*

$$h_m(\sigma_0, \sigma_1, \ldots, \sigma_m) = M_{\sigma_0}(\sigma_1 \cdots \sigma_m), \;\; \sigma_i \in \Sigma, \; i = 0, 1, \ldots, m.$$

Definition 5. *A (Σ, Ω, k)-mapping family (h_m), $m \geq 0$, determines a k-tree homomorphism $h : T_\Sigma \to T_\Omega$ defined inductively as follows:*

(i) *If t consists of one node labeled by $\sigma \in \Sigma$, $h(t) = h_0(\sigma)$.*

(ii) *If $t = \sigma(t_1, \ldots, t_m)$, $m \geq 1$,*

$$h(t) = h_m(\sigma, t_1(\varepsilon), \ldots, t_m(\varepsilon))[x_1 \leftarrow h(t_1), \ldots, x_m \leftarrow h(t_m)].$$

The family of all k-tree homomorphisms, $k \geq 0$, is denoted by $\mathcal{HOM}_k$.

Note that in the operations of a k-transducer, variables are written on the output tree only after they are stored in a memory location. Hence for a 0-tree homomorphism h, for any $m \geq 0$ and $\overline{\omega} \in \Sigma^{m+1}$, $h_m(\overline{\omega}) \in T_\Omega$, that is, the tree $h_m(\overline{\omega})$ does not contain variable occurrences and h maps any tree t to a tree that depends only on the root label of t and the labels of the children of the root.

Similarly, for a 1-tree homomorphism h, the possible variable occurrences in $h_m(\overline{\omega})$ appear in increasing order from left-to-right based on the index of the variable. Intuitively, this means that the mapping h cannot swap the places of any two subtrees.

It is easy to see that all ordinary linear tree homomorphisms defined on a ranked alphabet Σ [14] can be realized by an $m_{\max}$-transducer, where $m_{\max}$ is the maximum rank of symbols in Σ. It follows from Lemma 67 below that the transducer needs $m_{\max}$ memory locations to realize all tree homomorphisms.

Our goal here has been to define a class of tree homomorphisms on unranked trees that preserve regularity, but otherwise the mappings are as general as possible. The k-transducers of Definition 3 have been designed with this goal in mind. The output tree produced by a k-transducer has at most one occurrence of any variable and similarly the definition does not allow expanding the output tree in parallel at more than one location which would easily define mappings that are not regularity-preserving. Furthermore, the definition uses only a bounded number of variable locations because otherwise it would be easy to define tree homomorphisms as in the example below. When dealing with unranked trees, a regularity-preserving homomorphism cannot perform an arbitrary permutation of variable locations below a given node. This applies even to very simple permutations as the one that places all odd numbered variables before the even numbered variables.

Example 66. *Let $\Sigma = \{\sigma\}$ and let $h : T_\Sigma \to T_\Sigma$ be a mapping determined by mappings h_m, $m \geq 0$, as in Definition 5.*[1] *Here h_m associates to an $(m+1)$-tuple $(\sigma, \ldots, \sigma)$ the tree*

$$\sigma(x_1, x_3, \ldots, x_{2\cdot\lceil\frac{m}{2}\rceil-1}, x_2, x_4, \ldots, x_{2\cdot\lfloor\frac{m}{2}\rfloor}).$$

It is easy to see that if, for example, L is the tree language

$$\{\sigma(t_1, \ldots t_{2n}) \mid t_i = \sigma \text{ if } i \text{ is odd, } t_i = \sigma(\sigma) \text{ if } i \text{ is even, } 1 \leq i \leq 2n,\ n \geq 1\},$$

[1]Note that for any $m \geq 2(k+1)$ the mapping h_m cannot be defined by a k-transducer as in Definition 4 and, hence, h is strictly speaking not a tree homomorphism.

then $h(L)$ *consists of all trees* $\sigma(\underbrace{\sigma, \ldots, \sigma}_{n \text{ copies}}, \underbrace{\sigma(\sigma), \ldots, \sigma(\sigma)}_{n \text{ copies}})$, $n \geq 1$.

As indicated by Example 66, the way that a regularity-preserving tree homomorphism can manipulate the order of the variables is necessarily restricted. The finite number of memory locations that a k-transducer can use to manipulate the variables restricts the possible permutations of the sequence of variables. In the next section we will see that the tree homomorphisms defined by k-transducers are guaranteed to preserve regularity.

The following lemma characterizes the permutations that can be defined by k-transducers, $k \geq 0$.

Lemma 67. *Let* $m, k \geq 0$ *be arbitrary but fixed. Consider an* $(m+1)$*-tuple of elements of* Σ, $(\sigma_0, \sigma_1, \ldots, \sigma_m)$ *and a permutation* π *of* $[m]$.

There exists a k*-transducer* M *that on input* $\sigma_1 \cdots \sigma_m$ *outputs* $\sigma_0(x_{\pi(1)}, \ldots, x_{\pi(m)})$ *if and only if for pairwise distinct* $j_1, \ldots, j_r \in [m]$ *and* $i \in [m]$ *the conditions*

$$j_1, \ldots, j_r < i, \ \text{and,} \ \ \pi(i) < \pi(j_1), \ldots, \pi(j_r), \tag{8.2}$$

imply that $r \leq k - 1$.

Proof. Suppose that there exist $j_1, \ldots, j_r, i$ as in (8.2) with $r \geq k$ and that some k-transducer M on input of length m outputs $\sigma_0(x_{\pi(1)}, \ldots, x_{\pi(m)})$. When at time t_1, M writes x_i to the output tree, the value i must be stored in one of the memory locations of M. (Recall that a k-transducer outputs only variables corresponding to some memory location.) On the other hand, since variable indices $j_1, \ldots, j_r$ occur in the input before i, and $x_{j_1}, \ldots, x_{j_r}$ are written to the output after x_i, at time t_1 also the indices $j_1, \ldots, j_r$ need to be in memory. This is impossible since M has only k memory locations.

Conversely, using induction on m it can be established that any permutation satisfying the condition in the statement of the lemma can be realized by a k-transducer. ■

As a consequence of Lemma 67 we see that $(k+1)$-transducers can always define mappings that cannot be defined by k-transducers.

Proposition 68. *For any $k \geq 0$, $\mathcal{HOM}_k \subset \mathcal{HOM}_{k+1}$.*

Proof. Choose $\Sigma = \{\sigma\}$. We observe that the tree homomorphism that on a $(k+1)$-ary node swaps the first and the last subtrees cannot be computed by a k-transducer. Denoted formally, let $\overline{\omega}_{k+2}$, be the tuple consisting of $k+2$ copies of σ and let $h_{k+1}(\overline{\omega}) = \sigma(x_{k+1}, x_2, x_3, \ldots, x_k, x_1)$. Then, by Lemma 67, h cannot be computed by a k-transducer.

On the other hand, the above mapping h can be computed by a $(k+1)$-transducer simply by storing the $k+1$ variables in memory and after that outputting them in the desired order. ■

If we disregard the variable occurrences, a k-transducer as in Definition 3 can produce all possible trees as output. In Proposition 69, for simplicity, we assume that the input alphabet contains two symbols not occurring in the output alphabet. With more effort, a transducer performing a similar computation could be constructed for any input and output alphabet, as long as the input alphabet is at least binary. (The *up* and *down* "move" instructions would be coded in binary.)

Note that in the proposition below we do not consider variables, and hence the transducer does not need any memory locations. In exactly the same way, a k-transducer can output any tree in $T_\Omega(X_m)$ subject only to the restriction that each variable occurs at most once and the left-to-right order of variables satisfies a condition analogous to Lemma 67.

Proposition 69. *Suppose that $|\Sigma| \geq |\Omega| + 2$. There exists a 0-transducer M such that, for any $t \in T_\Omega$, there exists an input $w \in \Sigma^*$ such that $M(w) = t$.*

Proof. In the proof we use the input symbols as caterpillar instructions for a tree-walking automaton [57]. Without loss of generality, $\Omega \subseteq \Sigma$ and let σ_{down} and σ_{up} be elements of Σ not occurring in Ω. An input $w \in \Sigma^*$ for M simply lists the node labels of t in left-to-right depth first order. After reading a symbol of Ω, M uses a corresponding "add leaf" instruction. A symbol σ_{down} indicates a *down* move and the needed symbol of Ω appears in w immediately following σ_{down}. Similarly, reading the symbol σ_{up} instructs M to continue adding siblings for the parent of the current node. ■

Note that in Proposition 69 the transducer M does not depend on t.

8.2 Linear homomorphisms preserve recognizability

We show that the k-tree homomorphisms preserve recognizability. Consider a tree automaton A and a k-tree homomorphism computed by a transducer M. We want to construct an NTA(NFA) B for $h(L(A))$. What makes the construction essentially different from the case of linear tree homomorphisms for ranked trees is that, for a subtree of the form $\sigma(s_1, \ldots, s_m)$, the tree $h_m(\sigma, s_1(\varepsilon), \ldots, s_m(\varepsilon))$ can be arbitrarily large for sufficiently large m. Because of this, B has to simulate in parallel both M and A. A horizontal NFA of B guesses the corresponding sequence of states of A and simulating the computation of M verifies that the actual inputs correspond to the outputs produced by M on the hypothetical inputs. At each node, B records the states p_{down} and p_{up} of M that correspond to the point in the computation just before M moves down from the current node and just after M has moved up to the current node.

Our construction borrows ideas from the well-known bottom-up simulation of a tree-walking automaton. The construction of the bottom-up automaton here is more involved because the computation of B has to nondeterministically guess the representation of the input tree in the form $h(t)$ and then simulate the transducer M in computations that output each of the inside subtrees of the form $h_m(\sigma, s_1(\varepsilon), \ldots, s_m(\varepsilon))$ and, by identifying vertical states of A with variable occurrences in the inside subtree, combine the computations as one accepting computation of A on t.

Theorem 70. *Any k-tree homomorphism, $k \geq 0$, preserves recognizability.*

Proof. For simplifying the notations we consider the case where the input and output alphabets coincide. Having distinct input and output alphabets would not change the construction in any essential way.

Let $k \geq 1$ and consider an arbitrary k-tree homomorphism h computed by a k-transducer $M = (\Sigma, P, p_0, R)$. Let $A = (Q, \Sigma, \delta, F)$ be an arbitrary DTA(DFA) where $H^A_{q,\sigma}$ is the horizontal DFA corresponding to $q \in Q$, $\sigma \in \Sigma$.

We outline the construction of an NTA(NFA) B that recognizes the tree language $h(L(A))$. On an input tree t_1, B nondeterministically guesses how t_1 is represented in the form $h(t_2)$, $t_2 \in T_\Sigma$, and then simulates a computation of A on t_2. Consider a subtree $s = \sigma(s_1, \ldots, s_m)$ of t_2 where the computation of A assigns state q_i to the root of s_i, $i = 1, \ldots, m$, and q to the root of s. Now the computation of B should, roughly speaking, assign to the root of

$$h_m((\sigma, s_1(\varepsilon), \ldots, s_m(\varepsilon)))[x_1 \leftarrow q_1, \ldots, x_m \leftarrow q_m] \tag{8.3}$$

the state q. In order to perform the above computation, the vertical states of B need to contain additional components besides the vertical state of A.

A vertical state of B is a 7-tuple

$$(p, r, w, z, p', r', w'), \tag{8.4}$$

where $p, p' \in P$, r and r' are states of a horizontal DFA of A (when simulating a computation on (8.3) this is the DFA $H^A_{q,\sigma}$), $w, w' \in (Q \cup \{\#\})^k$, where $\#$ is a new symbol, and $z \in Q \cup \Sigma$. The use of the components can be described as follows. The first three components record the following information from the simulated computation of M at the point before it makes a *down* move from the current node:

- p is the state of M that the transducer has reached before making a *down* move from the current node.

- r is the state of the horizontal DFA $H^A_{q,\sigma}$ after processing a sequence $q_1 \cdots q_i$ of states of Q where the computation of M has processed the first i elements of the input $s_1(\varepsilon) \cdots s_m(\varepsilon)$. That is, the computation of M has processed a prefix of length i from the input that corresponds to variables $x_1, \ldots, x_i$ in the output of M, and x_j corresponds to q_j in the simulated computation.

- $w \in (Q \cup \{\#\})^k$ is a tuple containing the elements of Q corresponding to the up to k variables stored in the memory of M before the simulated computation

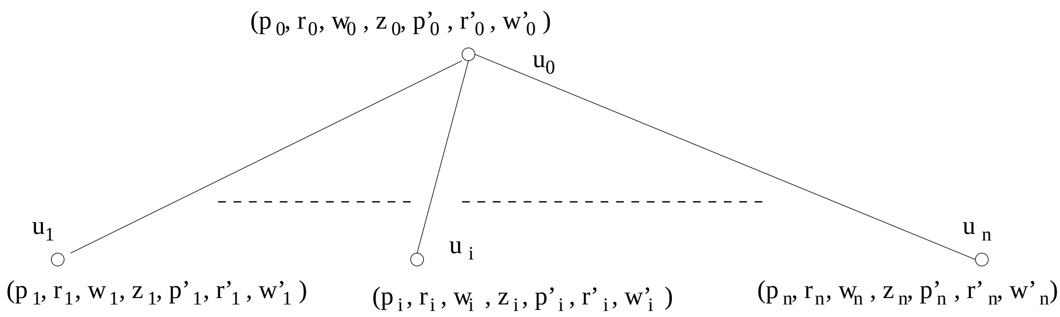

Figure 8.2: A vertical computation step of B.

makes a *down* move from the current node. A component # in the tuple indicates that the corresponding memory location is empty.

The last three components p', r', w' of (8.4) analogously store the same information at the point where the computation of M has made an *up* move to the current node. Finally, z is the state of Q in the simulated computation of A, assuming the current node of t_1 "corresponds" to a node of t_2 (when h maps t_2 to t_1). Otherwise, when the current node is not the root of the corresponding internal subtree of the form (8.3), $z \in \Sigma$ encodes the label of the current node (that is not a variable) in the corresponding internal subtree (8.3).

An individual vertical computation step of B is illustrated in Figure 8.2. We consider a node u_0 with n children, where the bottom-up computation has reached the children in a sequence of states $(p_i, r_i, w_i, z_i, p'_i, r'_i, w'_i)$, $i = 1, \ldots, n$. The horizontal computation of B performs the following steps on this sequence:

(H1) The horizontal NFA of B remembers the components p_1, r_1, w_1, (as these will be needed for verifying the correctness of the computation at the parent node).

(H2) For each $1 \leq i < n$, such that the ith child u_i of u_0 is a leaf of the input tree, the horizontal computation of B checks that

- the states p_i and p'_i correspond to a correct sequence S_1 of state transitions resulting from possible "store instructions" followed by an "add leaf" instruction of M.
  - If the "add leaf" transition of M added a leaf labeled by a variable, then z_i is the state of Q associated to that variable in the memory component w_i.
  - If the "add leaf" transition of M added a leaf labeled by $\sigma \in \Sigma$, then $z_i = \sigma$.
- When performing the sequence of transitions S_1 of M, the sequence of states of Q (corresponding to variables) that were consumed and stored in memory changes the state of the horizontal DFA $H^A_{\sigma,q}$ of A from r_i to r'_i.
- When performing the sequence of transitions S_1, the memory contents of M changes from w_i to w'_i, when a variable stored in memory is interpreted as the corresponding element of Q.

(H3) For $1 \leq i < n$, such that the ith child u_i of u_0 is *not* a leaf, the correspondence between p_i, r_i, w_i and p'_i, r'_i, w'_i, respectively, is arbitrary.[2]

(H4) Verifies that for all $1 \leq i < n$, $p'_i = p_{i+1}$, $r'_i = r_{i+1}$, $w'_i = w_{i+1}$.

Continuing with the notation of Figure 8.2, the vertical computation of B verifies that

(V1) The computation of M in state p_0 makes a *down* move writing the label of the node u_0 and changing the state to p_1. Note that the vertical computation step

[2]Note that in this case the computation of M first makes a *down* move from u_i, and the state of M, as well as, the state of the simulated horizontal computation of A after a corresponding *up* move is represented by the components p'_i, r'_i, w'_i.

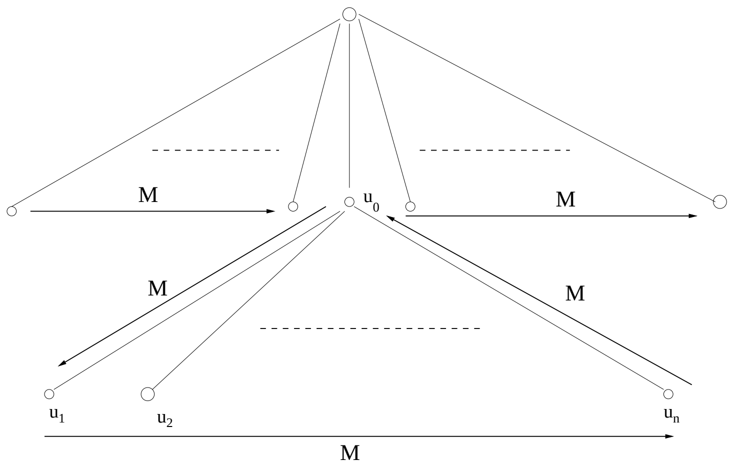

Figure 8.3: Combining the bottom-up computation step at u_0 with the left-to-right computation of M.

of B "knows" the label of u_0 (as the fourth component of the state (8.4)) and, according to (H1), the horizontal computation of B remembers p_1. Also, it is required that $w_0 = w_1$ because the memory contents of M does not change in a *down* move.

(V2) The computation of M in state p'_n makes an *up* move to state p'_0 and $w'_n = w'_0$.

By verifying the properties (V1) and (V2) the computation of B is able to check that the state labeling simulates a correct computation of M also with respect to the siblings of u_0 in Figure 8.2. The overall computation of M is illustrated in Figure 8.3.

The computation of B on an inside subtree of the form (8.3) assigns a state q of A to the root of the inside subtree, where $H^A_{q,\sigma}$ accepts the input $q_1 \cdots q_n$. A state of B as in (8.4) is accepting if $z \in F$ is an accepting state of A. Thus, the nondeterministic computation of B verifies that the input tree t_1 can be represented in the form $h(t_2)$ where A has an accepting computation on t_2. ■

Chapter 9

Conclusion

In the thesis we have investigated various aspects of descriptional complexity of unranked tree automata. In the following we summarize some of the main contributions.

Since a minimal DTA(DFA) need not be unique [35], DTA(DFA)s can have trade-offs between the numbers of vertical and horizontal states, that is, we can reduce the number of horizontal states by introducing additional vertical states. In Chapter 3, we investigated the state trade-offs in unranked tree automaton models. For DTA(DFA)s we gave a lower bound construction that is close to the corresponding worst-case upper bound for state trade-offs. Although a DTA(NFA) is a more general model than a DTA(DFA), we proved that there can be no state trade-offs in DTA(NFA)s. We gave a lower bound example where by adding one vertical state to an NTA(NFA), the number of horizontal states is reduced to the square root. However, the lower bound remains very far away from the upper bound. The state trade-offs for nondeterministic unranked tree automata is a topic for further research.

In Chapter 4, we studied the state complexity of conversions between different models of tree automata operating on unranked trees. The comparison shows that the horizontal state complexity of SDTAs and DTA(DFA)s, respectively, is incomparable. A DTA(DFA) can be more succinct than an SDTA because, while the vertical computation is deterministic, a DTA(DFA) can be viewed to use "unambiguous nondeterminism" to choose the corresponding horizontal DFA. An SDTA, on the other hand, has to process the sequence of states reached below the current node with label σ completely deterministically. However, an SDTA can be more succinct than a DTA(DFA) in cases where, roughly speaking, it is easy to distinguish the different states to be assigned to σ but the sizes of the horizontal DFAs are large. For converting nondeterministic automata to SDTAs and DTA(DFA)s, respectively, we gave non-polynomial lower bounds, however, the upper and lower bounds for the numbers of horizontal states remain far apart, and this is a topic for further research.

In Chapter 6 and Chapter 7 we investigated state complexity on various operations of deterministic tree automata (DTA(DFA)s). For the basic operations of union and intersection we gave bounds also for strongly deterministic automata (SDTAs). We studied the operational state complexity of two variants of deterministic unranked tree automata in Section 7.1 and Section 7.2. For union and intersection, tight upper bounds on the number of vertical states were established for both strongly deterministic automata and DTA(DFA)s. An almost tight upper bound on the number of horizontal states was obtained in the case of strongly deterministic unranked tree automata. For DTA(DFA)s, lower bounds for the numbers of horizontal states are hard to establish because there can be trade-offs between the numbers of vertical and horizontal states. This is indicated also by the fact that minimization of DTA(DFA)s is intractable and the minimal automaton need not be unique [35].

In Section 6.1, we established tight state complexity bounds for both sequential and parallel tree concatenation. The bound for sequential concatenation of tree languages differs by an order of magnitude from the corresponding bound for regular string languages. The results for parallel concatenation are similar to the string case.

In the natural way, based on the concatenation operations we can define the ith powers, $i \geq 0$, of a tree language T, and then we can define the Kleene-star of T as the infinite union of all ith powers of T, $i \geq 0$. With this definition it is easy to see that a Kleene-star operation based on parallel concatenation does not need to preserve regularity. There are two different ways to define the Kleene-star based on sequential concatenation, depending on how we define the powers of a tree language, which we call the bottom-up and the top-down star.

We gave tight state complexity bounds for both bottom-up and top-down Kleene-star operations in Section 6.2. We showed that the bottom-up star of a tree language recognized by a deterministic bottom-up automaton with n vertical states can be recognized by an automaton with $(n + \frac{3}{2}) \cdot 2^{n-1}$ states and, furthermore, there exist worst-case examples where this number of states is needed. This bound is, roughly, n times the corresponding bound for regular string languages. On the other hand, the state complexity of the top-down star operation is shown to coincide with the state complexity of Kleene-star on string languages.

The state complexity for bottom-up star is of a different order of magnitude than the corresponding bound for string languages. This means that the worst-case constructions essentially need to rely on "tree properties" and finding the minimal alphabet size remains an open question.

In Section 7.3, we extended the projection operation to tree languages and gave a tight upper bound on the number of vertical states. It differs from the known tight

Table 9.1: Operational state complexity of tree languages

Union	Intersection
$(m+1)(n+1)-1$	mn
Concatenation	
Sequential concatenation	Parallel concatenation
$(n+1)\cdot(m\cdot 2^n+2^{n-1})-1$	$m\cdot 2^n+2^{n-1}-1$
Kleene-star	
Bottom-up star	Top-down star
$(n+\frac{3}{2})2^{n-1}$	$\frac{3}{4}\cdot 2^n$
Quotient	
Bottom-quotient	Top-quotient
$(n+1)2^n-1$	n
Projection	
2^n-1	

upper bound on the string languages. For quotient operation, we get a tight bound on the number of the vertical states for the state complexity of top-quotient and bottom-quotient, respectively, in Section 7.4. The bound for bottom-quotient differs, roughly, by a multiplicative factor $n+1$ from the corresponding result for ordinary finite automata. To obtain tight upper bounds on the number of horizontal states for different operations will be a topic for future research.

Finally we sum up the state complexity of different operations in Table 9.1. Given two tree languages T_1 and T_2 with state complexity of m and n respectively, the state complexity of different operations on the number of vertical states is listed in the table. The bounds listed in Table 9.1 are all tight.

9.1 Future work

In this section we list some questions that are left open.

- We do not know what is the smallest alphabet size needed for the lower bounds for tree concatenation. In Section 6.1, for ease of presentation our lower bound constructions for sequential and parallel concatenation were based on the lower bound construction of [64] that uses an alphabet of size three. The alphabet size could be reduced by basing the constructions on the lower bound example of Jiráskova [27] over a binary alphabet. However, even in the simpler case of parallel concatenation our construction requires the addition of a new symbol and we do not know whether the lower bound of Theorem 43 holds for incomplete DFAs over a binary alphabet. The question of minimal alphabet size is more involved for sequential concatenation because there the lower bound construction needs to use a non-unary ranked alphabet.

- In Section 3.4, we have given a lower bound example where by adding one vertical state to an NTA(NFA), the number of horizontal states is reduced to the square root. However, the lower bound remains very far away from the upper bound. For state trade-offs in NTA(NFA)s, it remains open whether it is possible to have exponential trade-offs.

- For converting an NTA(NFA) A to a DTA(DFA) B, we have a tight upper bound on the number of vertical states. For horizontal states, Lemma 30 in Section 4.2.2 has given an upper bound for B that is exponential in the sum of vertical and horizontal states of A. However, the number of horizontal states given in the corresponding lower bound of Theorem 33 is exponential only in the number of horizontal states of A. We can try to get close estimates on the number of horizontal states by state complexity approximation as the authors did in [11] for finite automata operating on strings. A tight upper bound for sizes of horizontal automata in the NTA(NFA) to DTA(DFA) transformation is still open.

- We have obtained tight upper bounds for the number of vertical states for various operations in Chapter 6 and Chapter 7. For operations of union and intersection we also get almost tight upper bounds on the number of horizontal states for DTA(DFA)s and SDTAs. We also gave upper bounds on the number of horizontal states for various operations in Chapter 7 and Appendix A. As we can see it is already non-trivial to obtain tight upper bounds for the number of vertical states. It would be challenging to get any reasonable lower bound for the size of horizontal automata in the operational state complexity constructions.

- We have defined regularity-preserving tree homomorphisms for unranked trees. Establishing bounds for the state complexity of tree homomorphisms remains a topic for future work. If a tree homomorphism h is defined using a string-to-tree transducer M as we discussed in Chapter 8, then a state complexity bound for $h(L)$ would depend on the size of a tree automaton for the tree language L and on the number of states of the transducer M.

Bibliography

[1] J. Alur and P. Madhusudan. Adding nesting structure to words. In O.H. Ibarra and Z. Dang, editors, *Developments in Language Theory*, Lecture Notes in Computer Science 4036, pages 1–13. Springer, 2006.

[2] J. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM (Association for Computing Machinery)*, 56(3), 2009.

[3] A. Barrero. Unranked tree languages. *Pattern Recognition*, 24:9–18, 1991.

[4] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets. Research report HKUST-TCSC-2001-05, Hong Kong University of Science & Technology Computer Science Centre, 2001.

[5] J. Carme, J. Niehren, and M. Tommasi. Querying unranked trees with stepwise tree automata. In *Rewriting Techniques and Applications*, Lecture Notes in Computer Science 3091, pages 105–118. Springer, 2004.

[6] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 2nd edition, 2008.

[7] J. Champavère, R. Gilleron, A. Lemay, and J. Niehren. Efficient inclusion checking for deterministic tree automata and XML schemas. *Information and Computation*, 207:1181–1208, 2009.

[8] C. Chitic and D. Rosu. On validation of XML streams using finite state machines. In *Proceedings of the 7th International Workshop on the Web and Databases*, pages 85–90, 2004.

[9] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications.* electronic book available at tata.gforge.inria.fr, 2007.

[10] J. Cristau, C. Löding, and W. Thomas. Deterministic automata on unranked trees. In M. Liskiewicz and R. Reischuk, editors, *Fundamentals of Computation Theory*, Lecture Notes in Computer Science 3623, pages 68–79. Springer, 2005.

[11] Z. Ésik, Y. Gao, G. Liu, and S. Yu. Estimation of state complexity of combined operations. *Theoretical Computer Science*, 410:3272–3280, 2009.

[12] M. Frick, M. Grohe, and C. Koch. Query evaluation on compressed trees. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, pages 188–197, 2003.

[13] Y. Gao, K. Salomaa, and S. Yu. Transition complexity of incomplete DFAs. *Fundamenta Informaticae*, 110(1–4):143–158, 2011.

[14] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages Vol. III*, pages 1–68. Springer, 1997.

[15] G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation and XML typing. *Journal of the ACM (Association for Computing Machinery)*, 52:284–335, 2005.

[16] T. J. Green, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata. In *Proceedings of the 9th International Conference on Database Theory*, Lecture Notes in Computer Science 2572, pages 173–189. Springer, 2003.

[17] H. Gruber and M. Holzer. Tight bounds on the descriptional complexity of regular expressions. In *Developments in Language Theory*, Lecture Notes in Computer Science 5583, pages 276–287. Springer, 2009.

[18] Y.-S. Han and K. Salomaa. Nondeterministic state complexity of nested word automata. *Theoretical Computer Science*, 410:2961–2971, 2009.

[19] M. Holzer and M. Kutrib. Nondeterministic descriptional complexity of regular languages. *International Journal of Foundations of Computer Science*, 14:1087–1102, 2003.

[20] M. Holzer and M. Kutrib. Descriptional and computational complexity of finite automata. In *Language and Automata Theory and Applications*, Lecture Notes in Computer Science 5457, pages 23–42. Springer, 2009.

[21] M. Holzer and M. Kutrib. Nondeterministic finite automata – recent results on the descriptional and computational complexity. *International Journal of Foundations of Computer Science*, 20(4):563–580, 2009.

[22] M. Holzer and M. Kutrib. Descriptional and computational complexity of finite automata – A survey. *Information and Computation*, 209(3):456–470, 2011.

[23] M. Holzer, K. Salomaa, and S. Yu. On the state complexity of k-entry deterministic finite automata. *Journal of Automata, Languages and Combinatorics*, 6:453–466, 2001.

[24] H. Hosoya and B. C. Pierce. Regular expression pattern matching for XML. *Journal of Functional Programming*, 13:961–1004, 2003.

[25] J. Hromkovič and G. Schnitger. On the hardness of determining small NFAs and of proving lower bounds on their size. In M. Ito and M. Toyama, editors, *Developments in Language Theory*, Lecture Notes in Computer Science 5257, pages 34–55. Springer, 2008.

[26] H.Seidl. Equivalence of finite-valued tree transducers is decidable. *Mathematical System Theory*, 27:285–346, 1994.

[27] G. Jiráskova. State complexity of some operations on binary regular languages. *Theoretical Computer Science*, 330:287–298, 2005.

[28] G. Jirásková and T. Masopust. On the state complexity of projected languages. In *Descriptional Complexity of Formal Systems*, Lecture Notes in Computer Science 6808, pages 198–211. Springer, 2011.

[29] F. E. Krause. Maximizing the product of summands; minimizing the sum of factors. *Mathematics Magazine*, 69(4):270–278, 1996.

[30] D. Krieger, A. Miller, N. Rampersad, B. Ravikumar, and J. Shallit. Decimations

of languages and state complexity. *Theoretical Computer Science*, 410:2401–2409, 2009.

[31] D. Lee, M. Mani, and M. Murata. Reasoning about XML schema languages using formal language theory. Technical report, IBM Almaden Research Center, 2000.

[32] H. Leung. Descriptional complexity of NFA of different ambiguity. *International Journal of Foundations of Computer Science*, 16:975–984, 2005.

[33] G. Liu, C. Martin-Vide, A. Salomaa, and S. Yu. State complexity of basic language operations combined with reversal. *Information and Computation*, 206:1178–1186, 2008.

[34] W. Martens. *Static Analysis of* XML *transformation and schema languages*. PhD thesis, Hasselt University (Belgium), 2006.

[35] W. Martens and J. Niehren. On the minimization of XML schemas and tree automata for unranked trees. *The Journal of Computer and System Sciences*, 73:550–583, 2007.

[36] A.N. Maslov. Estimates of the number of states of finite automata. *Soviet Mathematics – Doklady*, 11:1373–1375, 1970.

[37] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *Journal of the ACM (Association for Computing Machinery)*, 51:2–45, 2004.

[38] F. R. Moore. On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Transactions on Computers*, 20:1211–1214, 1971.

[39] M. Murata, D. Lee, M. Man, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM (Association for Computing Machinery) Transactions on Internet Technology*, 5:660–704, 2005.

[40] F. Neven. Automata theory for XML researchers. *SIGMOD (Special Interest Group on Management Of Data) Record*, 31:39–46, 2002.

[41] A. Okhotin and K. Salomaa. Descriptional complexity of unambiguous nested word automata. In *Language and Automata Theory and Applications*, Lecture Notes in Computer Science 6638, pages 414–426. Springer, 2011.

[42] Y. Pan, Y. Zhang, and K. Chiu. Simultaneous transducers for data-parallel xml parsing. In *Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, pages 1–12, 2008.

[43] T. Pankowski. Transformation of xml data using an unranked tree transducer. In *Proceedings of E-Commerce and Web Technologies*, pages 259–26. Springer-verlag Berlin, 2003.

[44] A. V. Payne. Matrix-based algorithms and an analysis of system structure for partially-observable discrete-event systems. Master's thesis, Queen's University, 1997.

[45] X. Piao and K. Salomaa. Operational state complexity of nested word automata. *Theoretical Computer Science*, 410:3290–3302, 2009.

[46] X. Piao and K. Salomaa. Operational state complexity of deterministic unranked

tree automata. In G. Pighizzini, I. McQuillan, and B. Trost, editors, *Proceedings of The 12th International Workshop on Descriptional Complexity of Formal Systems*, pages 181–192. University of Saskatchewan, 2010.

[47] X. Piao and K. Salomaa. State complexity of concatenation of regular tree languages. *Theoretical Computer Science*, 2011. Accepted for publication.

[48] X. Piao and K. Salomaa. State complexity of star and quotient operation for unranked tree automata. Technical Report 2011-577, School of Computing, Queen's University, 2011.

[49] X. Piao and K. Salomaa. State trade-offs in unranked tree automata. In *Descriptional Complexity of Formal Systems*, Lecture Notes in Computer Science 6808, pages 261–274. Springer, 2011.

[50] X. Piao and K. Salomaa. Transformations between different models of unranked bottom-up tree automata. *Fundamenta Informaticae*, 109:405–424, 2011.

[51] X. Piao and K. Salomaa. State complexity of Kleene-star operations on trees. In *International Workshop on Theoretical Computer Science*, Lecture Notes in Computer Science 7160, pages 388–402. Springer, 2012. Auckland, New Zealand.

[52] X. Piao and K. Salomaa. Lower bounds for the size of deterministic unranked tree automata. *submitted to Theoretical Computer Science*, December 2011.

[53] X. Piao, K. Salomaa, and S. Vágvölgyi. Tree homomorphisms for unranked trees. In *Proceedings of 13th International Conference on Automata and Formal Languages*, pages 343–357, Debrecen, Hungary, Aug. 2011.

[54] S. Raeymaekers and M. Bruynooghe. Minimization of finite unranked tree automata. manuscript, 2004.

[55] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume I–III. Springer-Verlag, 1997.

[56] A. Salomaa, K. Salomaa, and S. Yu. Undecidability of the state complexity of composed regular operations. In *Language and Automata Theory and Applications*, Lecture Notes in Computer Science 6638, pages 489–498. Springer, 2011.

[57] K. Salomaa, S. Yu, and J. Zan. Deciding determinism of caterpillar expressions. *Theoretical Computer Science*, 410:3438–3446, 2009.

[58] E. M. Schmidt. *Succinctness of descriptions of context-free, regular and finite languages*. PhD thesis, Cornell University, 1978.

[59] T. Schwentick. Automata for XML. *The Journal of Computer and System Sciences*, 73:289–315, 2007.

[60] J. Shallit. *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press, 2009.

[61] G. Slutzki and S. Vagvolgyi. Deterministic top-down tree transducers with iterated look-ahead. *Theorical Computer Science*, 143:285–308, 1995.

[62] W3C Recommendation, http://www.w3.org/TR/xml/. *Extensible Markup Language (XML) 1.0*, 5th edition, Nov. 2008.

[63] K. Wong. On the complexity of projections of discrete-event systems. In *Proceedings of Workshop Series on Discrete Event Systems*, pages 201–206, Cagliari, Italy, 1998.

[64] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages Vol. I*, pages 41–110. Springer-Verlag, 1997.

[65] S. Yu. State complexity: Recent results and open problems. *Fundamenta Informaticae*, 64:471–481, 2005.

[66] S. Yu, Q. Zhuang, and K. Salomaa. The state complexity of some basic operations on regular languages. *Theoretical Computer Science*, 125:315–328, 1994.

Appendix A

Upper bound constructions for Concatenation and Kleene-star on unranked trees

In Chapter 6 we presented upper bound constructions for the different concatenation and iterated concatenation operations only for ranked trees. This was done for readability, because also for the corresponding tight lower bound constructions we needed only binary trees. Here, for the sake of completeness, we present upper bounds for the Kleene-star and concatenation operations in the more general case of unranked tree automata.

A.1 Kleene-star

The following lemma gives an upper bound for the state complexity of bottom-up star.

Lemma 71. *Let $A = (Q, \Sigma, \delta, F)$ be an arbitrary DTA(DFA), and $H^A_{\sigma,q} = (C_{\sigma,q}, Q, \mu_{\sigma,q}, c_{\sigma,q,0}, U_{\sigma,q})$ be the DFA representing the horizontal language associated with σ and q.*

We can construct a DTA(DFA) B recognizing the language $L(A)^{b,*}_{\alpha}$, $\alpha \in \Sigma$ (bottom-up star operation) with at most

$$(|Q| + \frac{3}{2})2^{|Q|-1}$$

vertical states.

Proof. Let $Q = \{s_1, s_2, \ldots, s_n\}$ and $0 \in Q$ be the state assigned to the leaf nodes labeled by α. Choose $B = (P, \Sigma, \lambda, E)$, where $P = P_1 \cup P_2 \cup P_3$,

- $P_1 = \{(\{q\} \cup p, q) \mid p \in \mathcal{P}(Q - \{q\}), q \in Q - F\}$,
- $P_2 = \{(\{q, 0\} \cup p, q) \mid p \in \mathcal{P}(Q - \{q, 0\}), q \in F\}$,
- $P_3 = \{(p, dead) \mid p \in \mathcal{P}(Q)\}$.

State (p, q) in P is final if q is in F or there exists state q' in p such that q' is in F. In the following, we assume that $\{0\} \cap F \neq F$, which means there is a final state different from 0.

For states $(p \cup \{q\}, q) \in P_1$ and states $(p \cup \{q, 0\}, q) \in P_2$, let $p = \{q_1, \ldots, q_m\}$, $0 \leq m < n$. The transition function $\lambda(\sigma, (p, q))$ is defined by DFA $G_{\sigma,(p,q)} = (D_{\sigma,(p,q)}, P, \gamma_{\sigma,(p,q)}, d_{\sigma,(p,q),0}, V_{\sigma,(p,q)})$. The set of states is $D_{\sigma,(p,q)} = ((\mathcal{P}(C_{\sigma,q_1}) \times \ldots \times \mathcal{P}(C_{\sigma,q_m})) \times (C_{\sigma,q_1} \times \ldots \times C_{\sigma,q_m}) \times C_{\sigma,q}$. The initial state is $d_{\sigma,(p,q),0} = ((\{c_{\sigma,q_1,0}\}, \ldots, \{c_{\sigma,q_m,0}\}), (c_{\sigma,q_1,0}, \ldots, c_{\sigma,q_m,0}), c_{\sigma,q,0})$. The set of final states $V_{\sigma,(p,q)}$ consists of all the states $v = ((C_1, \ldots, C_m), (u_1, u_2, \ldots, u_m), u)$ where for every $1 \leq i \leq m$, $C_i \cap U_{\sigma,q_i} \neq \emptyset$, $u \in U_{\sigma,q}$.

The transition function $\gamma_{\sigma,(p,q)}$ is defined as follows: for any input $(p', q') = (\{j_1, j_2, \ldots, j_i\}, q')$ and any state $d = ((C_1, \ldots, C_m), (c_1, c_2, \ldots, c_m), c)$ in $D_{\sigma,(p,q)}$,

$$\gamma_{\sigma,(p,q)}((p', q'), d) =$$
$$((\bigcup_{c \in C_1} \mu_{\sigma,q_1}(q', c) \cup \bigcup_{l=1}^{i} \mu_{\sigma,q_1}(j_l, c_1), \ldots, \bigcup_{c \in C_m} \mu_{\sigma,q_m}(q', c) \cup \bigcup_{l=1}^{i} \mu_{\sigma,q_m}(j_l, c_m)),$$
$$(\mu_{\sigma,q_1}(q', c_1), \ldots, \mu_{\sigma,q_m}(q', c_m)), \mu_{\sigma,q}(q', c))$$

The γ-transition in the horizontal DFA $G_{\sigma,(p,q)}$ of B is defined using the μ-transition in the horizontal DFAs in A. However, since the input of $G_{\sigma,(p,q)}$ is a sequence of states of B, e.g. $(p_1, q_1), (p_2, dead), \ldots, (p_m, q_m)$, *dead* is also a possible input for the μ-transition. Since *dead* is not in the alphabet of the horizontal DFAs of A, which means that μ is undefined for input *dead*, we define the μ-transition in the horizontal DFAs in A such that any transition labeled by *dead* goes to the sink state of the DFAs.

For states in $(p, dead) \in P_3$, let $p = \{q_1, \ldots, q_m\}$, $m \leq n$. The transition function $\lambda(\sigma, (p, dead))$ is defined by the DFA
$G_{\sigma,(p,dead)} = (D_{\sigma,(p,dead)}, P, \gamma_{\sigma,(p,dead)}, d_{\sigma,(p,dead),0}, V_{\sigma,(p,dead)})$. The set of states is $D_{\sigma,(p,dead)} = ((\mathcal{P}(C_{\sigma,q_1}) \times \ldots \times \mathcal{P}(C_{\sigma,q_m})) \times ((C_{\sigma,q_1}) \times \ldots \times C_{\sigma,q_m}) \times (C_{\sigma,s_1}, \ldots, C_{\sigma,s_n})$. The initial state is $d_{\sigma,(p,dead),0} = ((\{c_{\sigma,q_1,0}\}, \ldots, \{c_{\sigma,q_m,0}\}),$
$(c_{\sigma,q_1,0}, \ldots, c_{\sigma,q_m,0}), (c_{\sigma,s_1,0}, \ldots, c_{\sigma,s_n,0}))$. The set of final states $V_{\sigma,(p,dead)}$ consists of all the states $v = ((C_1, \ldots, C_m), (u_1, u_2, \ldots, u_m), (v_1, \ldots, v_n))$ where for every $1 \leq i \leq m$, $C_i \cap U_{\sigma,q_i} \neq \emptyset$, and there does not exist any v_i that $v_i \in \bigcup_{s \in Q} U_{\sigma,s}$.

The transition function $\gamma_{\sigma,(p,dead)}$ is defined as follows: for any input $(p', q') = (\{j_1, j_2, \ldots, j_i\}, q')$ and any state

$d = ((C_1, \ldots, C_m), (c_1, c_2, \ldots, c_m), (c^1, \ldots, c^n))$ in $D_{\sigma,(p,q)}$,

$$\gamma_{\sigma,(p,dead)}((p', q'), d) =$$
$$((\bigcup_{c \in C_1} \mu_{\sigma,q_1}(q', c) \cup \bigcup_{l=1}^{i} \mu_{\sigma,q_1}(j_l, c_1), \ldots, \bigcup_{c \in C_m} \mu_{\sigma,q_m}(q', c) \cup \bigcup_{l=1}^{i} \mu_{\sigma,q_m}(j_l, c_m)),$$
$$(\mu_{\sigma,q_1}(q', c_1), \ldots, \mu_{\sigma,q_m}(q', c_m)), (\mu_{\sigma,q_1}(q', c^1), \ldots, \mu_{\sigma,q_n}(q', c^n)))$$

The vertical states in B consist of two components. The first component simulates the situation where the concatenation occurs, and the second component records the computation where there is no concatenation. At a subtree $\sigma(t_1, t_2, \ldots, t_m)$, where states $(p_1, q_1), (p_2, q_2), \ldots, (p_m, q_m)$, $p_i \subseteq Q, 1 \leq i \leq m$ are assigned to the roots of $t_1, t_2, \ldots, t_m$, respectively, state (p, q) is assigned to the root labeled by σ provided that the sequence of states $q_1 q_2 \ldots q_m$ is accepted by $H^A_{\sigma,q}$, and p consists of all states that are obtained by A by in the child nodes "taking" one state in p_i and using q_j, for all $j \neq i$ for all choices of i. The state $(p, dead)$, $p \neq Q$ is assigned to the node labeled by σ if none of the DFAs $H_{\sigma,q}$ in A accepts the sequence $q_1 q_2 \ldots q_m$. When the vertical computation produces a state with the second component in F, the computation adds 0 to the first component of the state.

For states (p, q) in P_1, we always have $q \in p$. This is done because for any state (p, q) where $q \notin p$ there is always an equivalent state $(p \cup \{q\}, q)$. Recall that a state (p, q) is final if $q \in F$ or $p \cap F \neq \emptyset$. Thus, after B reading some tree, B reaches a final state from the state (p, q) if and only if the state $(p \cup \{q\}, q)$ reaches a final state.

According to the construction, $|P_1| = (|Q| - |F|)2^{|Q|-1}$, $|P_2| = |F|2^{|Q|-2}$, $|P_3| = 2^{|Q|}$. We have $|P| = (|Q| - \frac{|F|}{2} + 2)2^{|Q|-1}$. The worst case is when $|F| = 1$. Then the number of vertical states of B is at most $(|Q| + \frac{3}{2})2^{|Q|-1}$. ∎

As regards the top-down star operation, the construction of a DTA(DFA) B recognizing $L(A)^{t,*}_{\sigma}$ ($\sigma \in \Sigma$), is more straightforward as there is no restriction on the location where a leaf can be replaced and B needs to keep track only of a set of states of A. We omit here a detailed construction which is actually quite similar to the construction used for ranked trees. The construction for ranked trees was given in Theorem 48.

A.2 Concatenation

We give a construction of an SDTA recognizing the sequential concatenation of two tree languages recognized by given SDTAs.

Lemma 72. *Let A_1 and A_2 be two arbitrary SDTAs. $A_i = (Q_i, \Sigma, \delta_i, F_i)$, $i = 1, 2$, transition function for each $\sigma \in \Sigma$ is represented by a DFA $H^{A_i}_{\sigma} = (C^i_{\sigma}, Q_i \cup \Sigma, \gamma^i_{\sigma}, c^i_{\sigma,0}, E^i_{\sigma})$ with an output function λ^i_{σ}.*

The language $L(A_2) \cdot^s_{\alpha} L(A_1)$[1]*, $\alpha \in \Sigma$ can be recognized by an SDTA B with*

$$\text{size}(B) \leq [\ (|Q_1|+1)\times(2^{|Q_1|}\times(|Q_2|+1)-2^{|Q_1|-1})-1;\ |\Sigma|(|C^2_{\sigma}|+1)(|C^1_{\sigma}|+1)\times 2^{|C^1_{\sigma}|+1}\ ].$$

Proof. Choose $B = (Q'_1 \times \mathcal{P}(Q_1) \times Q'_2, \Sigma, \delta, F)$, where $Q'_1 = Q_1 \cup \{dead\}$, $Q'_2 = Q_2 \cup \{dead\}$. A state $(p_1, P_2, q) \in Q'_1 \times \mathcal{P}(Q_1) \times Q'_2$ is final if $P_2 \cap F_1 \neq \emptyset$.

The transition function δ associated with each σ is represented by a DFA $H^B_{\sigma} = (S \times \mathcal{P}(C^1_{\sigma}) \times S', (Q'_1 \times \mathcal{P}(Q_1) \times Q'_2) \cup \Sigma, \mu, (c^1_{\sigma,0}, \emptyset, c^2_{\sigma,0}), V)$ with an output function λ^B_{σ}, where $S = C^1_{\sigma} \cup \{dead\}$, $S' = C^2_{\sigma} \cup \{dead\}$. A state $(c_1, C_2, s) \in S \times \mathcal{P}(C^1_{\sigma}) \times S'$ is final if $s \in E^2_{\sigma}$ or there exists $c \in c_1 \cup C_2$ such that $c \in E^1_{\sigma}$. The transition function μ is defined as below:

[1] Recall that $T_A \cdot T_B$ consists of trees where in some tree of T_B a leaf is replaced by a tree of T_A.

For an input $a \in \Sigma$,

$$\mu((c_1, C_2, s), a) = (\gamma^1_\sigma(c_1, a), \bigcup_{c_2 \in C_2} \gamma^1_\sigma(c_2, a),\ \gamma^2_\sigma(s, a))$$

For an input $(p_1, P_2, q) \in Q'_1 \times \mathcal{P}(Q_1) \times Q'_2$,

$$\mu((c_1, C_2, s), (p_1, P_2, q)) = (\gamma^1_\sigma(c_1, p_1),\ [\bigcup_{p_2 \in P_2} \gamma^1_\sigma(c_1, p_2) \cup \bigcup_{c_2 \in C_2} \gamma^1_\sigma(c_2, p_1)],\ \gamma^2_\sigma(s, q))$$

Write the computation above in an abbreviated form as $\mu((c_1, C_2, s), r) = (p'_1, P'_2, q')$, $r \in \Sigma \cup (Q'_1 \times \mathcal{P}(Q_1) \times Q'_2)$. When compute p'_1 and q', if any $\gamma^i_\sigma(c, \xi)$, $i = 1, 2$, $c = c_1, s$, $\xi \in \Sigma \cup Q_i$, is not defined in A_i, assign *dead* to p'_1 or q'. When computing P'_2, add nothing to P'_2 if any $\gamma^i_\sigma(c, \xi)$ is not defined.

Let $p_\alpha \in Q_1$ denote the state assigned to the leaf labeled by α in A_1 that is substituted by a tree in $L(A_2)$. The function λ^B_σ is defined as: for any final state $e = (c_1, C_2, s)$, $x_1 = c_1 \cap E^1_\sigma$, $X_2 = C_2 \cap E^1_\sigma$,

1 If $s \in E^2_\sigma$

$$\lambda^B_\sigma(e) = \begin{cases} (\lambda^1_\sigma(x_1),\ p_\alpha \cup \bigcup_{x_2 \in X_2} \lambda^1_\sigma(x_2),\ \lambda^2_\sigma(s)), \text{ if } \lambda^2_\sigma(s) \in F_2 \\ (\lambda^1_\sigma(x_1),\ \bigcup_{x_2 \in X_2} \lambda^1_\sigma(x_2),\ \lambda^2_\sigma(s)), \text{ if } \lambda^2_\sigma(s) \notin F_2 \end{cases}$$

2 If $s \notin E^2_\sigma$,

$$\lambda^B_\sigma(e) = (\lambda^1_\sigma(x_1), \bigcup_{x_2 \in X_2} \lambda^1_\sigma(x_2),\ dead).$$

If $x_1 = \emptyset$, above we interpret $\lambda^1_\sigma(x_1) = dead$. If $X_2 = \emptyset$, above we interpret $\bigcup_{x_2 \in X_2} \lambda^1_\sigma(x_2) = \emptyset$.

A state of B has three components (p_1, P_2, q). The first component p_1 is used to keep track of A_1's computation where no concatenation is done. p_1 is computed by the first component c_1 in the state of H^B_σ. The set P_2 keeps track of all the vertical states A_1 may be in assuming a concatenation has taken place at some node below the current node. In a state (c_1, C_2, s) of H_σ^B, $C_2 \neq \emptyset$ (or $C_2 = \emptyset$) records whether there is (or is not) a concatenation in the computation. The third component q keeps track of the computation of A_2. When a final state is reached in A_2, this means a concatenation might take place. In this case, the state p_α is added to P_2, which is achieved by the λ_σ^B function in B.

According to the definition of λ_σ^B, when $\lambda_\sigma^2(s) \in F_2$, p_α is always in the second component of the state. We can exclude the cases when $\lambda_\sigma^2(s) \in F_2$, and p_α is not in the second component of the state, and we do not require B be complete. These observations indicate that B needs at most $(|Q_1|+1) \times (2^{|Q_1|} \times (|Q_2|+1) - 2^{|Q_1|-1}) - 1$ vertical states. ■

Note that, when we are not concerned about the number of horizontal states, from an SDTA A we get immediately an equivalent DTA(DFA) B with the the same set of vertical states. For each $\sigma \in \Sigma$ and each state q of A, the horizontal DFA of B corresponding to q and σ, is obtained from the horizontal DFA H_σ^A of A by omitting the output function and selecting as final states all the states of H_σ^A for which the output function assigns the state q.

The above observation means that as a consequence of Lemma 72 we get an upper bound for the number of vertical states needed to recognize the concatenation of two DTA(DFA)s:

Corollary 73. *Let A_i be a DTA(DFA) with m_i states, $i = 1, 2$, and let $\sigma \in \Sigma$. The sequential concatenation $L(A_2) \cdot_\sigma^s L(A_1)$ can be recognized by a DTA(DFA) with*

$$(|m_1| + 1) \times (2^{|m_1|} \times (|m_2| + 1) - 2^{|m_1|-1}) - 1$$

states.

As we have mentioned before, the bound of Corollary 73 coincides with the upper bound given for the state complexity of sequential concatenation of ranked trees in Theorem 47.