Contributions to the
Theory and Applications of Tree Languages

Johanna Högberg
Preface

The study of tree languages was initiated in the late 1960s and was originally motivated by various problems in computational linguistics. It is presently a well-developed branch of formal language theory and has found many uses, including natural language processing, compiler theory, model checking, and tree-based generation. This thesis is, as its name suggests, concerned with theoretical as well as practical aspects of tree languages. It consists of an opening introduction, written to provide motivation and basic concepts and terminology, and eight papers, organised into three parts. The first part treats algorithmic learning of regular tree languages, the second part is concerned with bisimulation minimisation of nondeterministic tree automata, and the third part is devoted to tree-based generation of music. We now summarise the contributions made in each part, and list the papers included therein. More detailed summaries of the individual papers are given in Chapter 3.

Part I. Algorithmic learning

In Part I, an inference algorithm for regular tree languages is presented. The algorithm is a generalisation of a previous algorithm by Angluin, and the learning task is to derive a deterministic finite tree automaton (henceforth, dfta) that recognises a target language $U$ over some ranked alphabet $\Sigma$. To learn about $U$, the inference algorithm may query a so-called MAT oracle, where MAT is short for Minimal Adequate Teacher. The information returned by the oracle is organised into a table of observations. Upon termination, the algorithm uses this table to synthesise $M_U$; the unique minimal partial dfta that recognises $U$. The inference algorithm executes in time $O(r |Q||\delta| (m + |Q|))$, where $Q$ and $\delta$ are the set of states and the transition table of $M_U$, respectively, $r$ is the maximal rank of any symbol in $\Sigma$, and $m$ is the maximum size of the counterexamples provided by the teacher. This improves on a similar algorithm proposed by Sakakibara as dead states are avoided both in the learning phase and in the resulting automaton. Part I also describes a concrete implementation that includes two extensions of the basic algorithm. The usefulness of these extensions is studied in an experimental setting, by running the variants of the algorithm against target languages with different characteristics.
Part II. Bisimulation minimisation

In Part II, bisimulation minimisation of nondeterministic weighted tree automata (wta) is introduced in general, and for finite tree automata (which can be seen as wta over the Boolean semiring) in particular. The concepts of backward and forward bisimulation are extended to wta, and efficient minimisation algorithms are developed for both types of bisimulation. Minimisation via forward bisimulation coincides with the standard minimisation algorithm when the input automaton is a dfta, whereas minimisation via backward bisimulation is ineffective for deterministic automata, but yields a smaller output automaton in the general case. Both algorithms execute in time $O(rmn)$, where $r$ is the maximal rank of any symbol in the input signature, and $n$ and $m$ are the size of the set of states and the size of the transition table of the input automaton, respectively. In the special case where the underlying semiring of the input automaton is either cancellative or Boolean, this time bound can be improved to $O(r^2 m \log n)$ and $O(rm \log n)$ for backward and forward bisimulation, respectively, by adapting existing partition refinement algorithms by Hopcroft, Paige, and Tarjan. The implemented algorithms are demonstrated on a typical task in natural language processing.


Part III.  Tree-based music generation

In Part III, we consider how tree-based generation can be applied in the area of algorithmic composition. To illuminate the matter, a small system capable of generating simple musical pieces is implemented in the software TREEBAG. This system, which we henceforth refer to as Willow, consists of a sequence of formal devices, familiar from the field of tree grammars and tree transducers. Part III also describes an algebra whose operations act on musical pieces, and shows how this algebra can be used to generate music in a tree-based fashion. Starting from input which is either generated by a regular tree grammar or provided by the user via a digital keyboard, a number of top-down tree transducers are applied to generate a tree over the operations provided by the music algebra. The evaluation of this tree yields the musical piece generated.


Acknowledgements

I have many reasons to be grateful. First of all, Frank Drewes is an excellent supervisor, and his love for formal language theory is most contagious. While he encourages me to work independently and find my own inspiration, his kind and patient support has never faltered. I am also very thankful to Jürgen Börsster who agreed to be my co-supervisor.

The random-context project, which was funded by SIDA, gave me the opportunity to visit South Africa, there to work with Sigrid Ewert at the University of Witwatersrand in Johannesburg, and with Andries van der Walt, Christine du Toit, and Brink van der Merwe at Stellenbosch University. This nice group of people taught me a lot about research. In this way, they have influenced this thesis more than what might be obvious at first sight, even if they are not co-authors of the papers that I decided to include.

I also wish to thank my friend and co-author Lisa Kaati, and her supervisor Parosh Aziz Abdulla, both at Uppsala University. They introduced me to bisimulation and taught me much about cooperation and priorities. I later had the pleasure to continue my work on bisimulation along with Jonathan May at the University of Southern California, and Andreas Maletti at Technische Universität Dresden. I am particularly grateful to Andreas who was a great rôle model, and taught me how to handle technical difficulties.
Albert Gräf at the Gutenberg-University in Mainz encouraged my interest in algorithmic music generation and provided me with useful references. I also appreciated the advice on music theory given to me by Claude Lacoursiére and Carl Rehnberg, and I am very obliged to the Kempe foundation and the European Mathematical Society for their generous financial support.

During the completion of this thesis I received a lot of help. Frank Drewes, Stephen Hegner, Fred Gustavson, Maria Nyström, Lars Larsson, Lars-Daniel Öhman, Andreas Maletti, and Johan Tordsson read (parts of) the draft, and their comments greatly improved the quality of the final document. Pedher Johansson provided me with Latex templates and patiently answered my questions. Helena Lindgren guided me through the intricacies of the printing process, and Lars Sandström at the printer agency Solfärdern Offset anticipated my every need.

Finally, I would like to thank my parents Anders and BrittInger, my sister Emma, my boyfriend Johan, and, of course, all of my friends. You make me feel safe, happy, and like doing a little dance.

Umeå, May 2007

Johanna Högberg
Contents

1 Introduction 1

2 Automata on trees 3
   2.1 Algebras and trees 4
   2.2 Regular tree languages and their automata 6
   2.3 Minimisation of finite tree automata 10
   2.4 Regular tree grammars 12
   2.5 Tree series 13
   2.6 Tree transductions 18

3 Summary of papers 23
   3.1 Part I. Algorithmic learning 23
   3.2 Part II. Bisimulation minimisation 25
   3.3 Part III. Tree-based generation of music 26
Inspired by the importance of context-free languages and the central rôle which derivation trees play for them, a theory of tree languages was established about 40 years ago by the seminal works of Brainerd, Doner, Mezei, Rounds, Thatcher, and Wright [Bra68, Don65, Don70, MW67, Tha67, TW68, Rou68, Rou70]. Since then, interest in tree language theory has been constantly increasing, and countless contributions to the theory and its applications have been made. A few of them constitute the main part of this thesis, which is grouped according to three themes: algorithmic learning of tree languages, bisimulation minimisation of nondeterministic tree automata, and tree-based music generation.

The objective of algorithmic learning is to derive an explicit representation of some target language from the information available. This information may, for instance, come in the form of a labelled sequence of samples, or as an oracle capable of answering certain types of queries. The papers that constitute Part I centre around the question of how a regular tree language may be learnt from a minimal adequate teacher, also known as a MAT oracle. Introduced in [Ang87], the MAT oracle is able to answer membership queries of the form is the tree an element of the target language? but also equivalence queries of the form is the automaton an accurate representation of the target language?. If the answer to the latter question is in the negative, then the MAT oracle responds with a counterexample, i.e. a tree that fails to classify correctly. If, on the other hand, the answer is positive, then the oracle returns a special token signalling that the target language has been successfully acquired. To obtain an efficient algorithm that solves this learning task, we take advantage of the structural properties of regular tree languages that are captured by the Myhill-Nerode theorem [Myh57, Ner58], here restated in Theorem 2.3.4 for ease of reference.

An important consequence of the Myhill-Nerode theorem is that every regular tree language (from now on, rtl) is accepted by a unique minimal deterministic finite tree automaton. There are a number of algorithms that compute this automaton for a given rtl, see for example [GS84, CDG+97]. However, nondeterministic automata offer in general a more succinct form of representation than deterministic automata. In fact, a deterministic automaton may well require an exponential number of states to recognise the same language as a nondeterministic automaton. It is unfortunate, then, that nondetermin-
istic automata come with two major drawbacks. First of all, the existence of a unique minimal nondeterministic automaton recognising \( L \) is not guaranteed, and what is worse, finding any minimal nondeterministic automaton that recognises \( L \) is known to be \( \text{PSPACE} \)-complete, unless \( P = \text{NP} \). To construct an efficient algorithm, we must therefore resort to heuristic methods such as bisimulation minimisation [Mil82]. A bisimulation relation is a congruence on the state space of a (nondeterministic) automaton \( M \) with respect to the transition function of \( M \). Every bisimulation relation is an equivalence relation and as such induces a partition on the state space of \( M \). The states that end up in the same equivalence class perform, in a sense, the same function and can for this reason be replaced by a single state without affecting the recognised language. In Part II of this thesis we develop bisimulation oriented minimisation algorithms for various types of tree automata.

Whereas a classical fta realises an evaluation of its input trees to elements in the Boolean domain – to true if the tree is to be accepted, and to false otherwise – one can equally well consider a scenario where trees are mapped to a more general domain. A tree over a ranked alphabet \( \Sigma \) is after all only a syntactical object, but in the context of an algebra that maps each symbol \( a \) of rank \( k \) in \( \Sigma \) to an operation of arity \( k \) on some domain \( D \), the tree is a term and stands, as such, to represent an element in \( D \). A set in which every element can be expressed as a term over a finite set of operations is said to be structured. This is a property that is interesting for a set to have from a computational point of view: every Turing computable set that is structured can be represented by combining a tree generating device, for example a grammar or regular expression, with an appropriate algebra. Having such a tree-based representation of a set is often convenient, as it is a suitable form for algorithmic processing. In Part III we investigate how a restricted type of musical pieces can be generated and transformed by means of formal devices from the field of tree language theory.

The chapters and sections that make up this preamble are organised as follows. Chapter 2 considers automata that operate on trees, beginning with a preliminary section on terms (viz. trees) and algebras. Section 2.2 is devoted to tree automata that map their input trees to Boolean values, thus defining tree languages, where particular attention is paid to finite tree automata. The results discussed in Section 2.2 form a backdrop for the entire thesis. The minimisation of finite tree automata is the topic of Section 2.3, and a closely related class of tree grammars is described in Section 2.4. Whereas the former section provides a context for the papers in Part I and II, the latter relates to the papers in Part III. In Section 2.5, we proceed to weighted tree automata, which map trees to semiring elements and hence define tree series. Section 2.5 is important for the understanding of the papers on bisimulation minimisation. Section 2.6 is concerned with tree transducers, i.e. automata that map trees to trees, and is written in preparation for the papers on tree-based music generation. The concluding Chapter 3 provides a summary of the papers that comprise Parts I through III.
A tree, in the mathematical sense, is a connected acyclic graph whose edges are either directed or undirected. In the field of Computer Science, trees are usually taken to be labelled, ordered and rooted. Every labelled tree has an associated mapping that takes each of its nodes to a symbol, so a labelled tree cannot be described by structure alone. A rooted tree $t$ has a distinguished node (the root of $t$) which induces a unique hierarchy on the nodes of $t$ that is determined by their distance from the root. Though this hierarchy is usually only given implicitly, it can be made explicit with directed edges. A rooted tree can also be ordered, in which case the subtrees that attach to a node can be enumerated in a fixed order. It is this type of trees that we choose to work with. More precisely, a tree is a formal expression over a finite set of symbols (i.e. labels). The formal definition will be given in Section 2.1.

**Example 2.0.1 (Terms)** Although a tree itself has no predefined semantics, we can associate with each symbol an operation on a domain $\mathbb{D}$, and then evaluate the tree to obtain an object in $\mathbb{D}$. Consider the tree in Figure 1 with nodes labelled by symbols in the set $\{f, g, a, b\}$. If we choose as our domain the set $[0, 3]$, and associate $f$ and $g$ with multiplication and addition modulo four, respectively, $a$ with the constant 1, and $b$ with the constant 2, then the tree evaluates to 1.

But maybe we would rather work with the domain of strings over $\{a, b\}$. We may then associate $f$ with concatenation, and $g$ with the binary operation *longest common prefix* ($lcp$). For example, the longest common prefix of $aaba$ and $aabba$ is $aab$. The symbols $a$ and $b$ are simply mapped to themselves. It is easy to ascertain that under these premises the tree in Figure 1 evaluates to the empty string, as this is the value of the rightmost subtree $g[a, b]$, and above it there are only $g$’s.

In Example 2.0.1, node labels are necessary if more than one function of arity $k$ occurs for any $k \in \mathbb{N}$, and without ordered subtrees, only commutative operations could be used. It should also be clear that only rooted trees can be evaluated in a well-defined manner. Let us now make precise the basic notions that are needed for a more formal discussion of the matter.
2.1 Algebras and trees

The set of all natural numbers 0, 1, 2, ... is denoted by \( \mathbb{N} \), and the subset \( \{1, \ldots, n\} \) of these by \([n]\). We write \(|S|\) for the cardinality of a set \( S \) and write \( \mathcal{P}(S) \) for the power set of \( S \). Furthermore, \( S \) is closed under the \( k \)-ary operation \( f \), if \( s_1, \ldots, s_k \) in \( S \) implies that also \( f(s_1, \ldots, s_k) \) is in \( S \).

Let \( \simeq \) and \( \cong \) be equivalence relations on the set \( S \), and let \( s \) and \( s' \) be elements of \( S \). The equivalence class of \( s \) with respect to the relation \( \cong \) is \([s]_{\cong} = \{s' \mid s \cong s'\}\). Since \([s]_{\cong} \) and \([s']_{\cong} \) are equal if \( s \cong s' \), and disjoint otherwise, the relation \( \cong \) induces a partition \((S/\cong) = \{[s]_{\cong} \mid s \in S\}\) on the set \( S \). The index of \( \cong \) is simply the size of \((S/\cong)\). If \( s \simeq s' \) implies that \( s \cong s' \), then \( \simeq \) is a refinement of \( \cong \), or equivalently, \( \cong \) is coarser than \( \simeq \). Note that if \( \cong \) is coarser than \( \simeq \), then the index of \( \cong \) is less or equal to that of \( \simeq \). If \( L \subseteq S \) is the union of some equivalence classes of \( \cong \), then \( \cong \) is said to saturate \( L \).

Definition 2.1.1 (Ranked alphabet) An alphabet is a finite set of symbols. A ranked alphabet is an alphabet \( \Sigma = \bigcup_{k \in \mathbb{N}} \Sigma_{(k)} \) which is partitioned into pairwise disjoint subsets \( \Sigma_{(k)} \), where the symbols in \( \Sigma_{(k)} \) are said to have rank \( k \). To indicate that a symbol \( f \) is in the set \( \Sigma_{(k)} \), we may add \((k)\) to \( f \) as a superscript, i.e. write \( f^{(k)} \).

Depending on the context, the terms signature and operator set are sometimes used as synonyms for ranked alphabet. From now on, \( \Sigma \) is a fixed but arbitrary ranked alphabet, unless otherwise explicitly stated.

Definition 2.1.2 (\( \Sigma \)-Algebra) A \( \Sigma \)-algebra \( A \) consists of a nonempty set \( A \) (the carrier set of \( A \)) and a family of operations \((f_a)_{a \in \Sigma}\). For each \( a \in \Sigma_{(k)} \), \( f_a \) is a function \( f_a : A^k \to A \).

Note that Definition 2.1.2 does not exclude operators of arity 0, which can be seen as constants. Let \( A = (A, (f_a^A)_{a \in \Sigma}) \) be a \( \Sigma \)-algebra, and let \( A' \) be a subset of \( A \). The sub-algebra generated by \( A' \) has as its carrier the smallest
superset $B$ of $A'$ that is closed under the operations $(f^A_a)_{a \in \Sigma}$, and the same operations, albeit restricted to $B$. The algebra $A$ is finite if $A$ is finite, and locally finite if the sub-algebra generated by any finite subset of $A$ is finite.

Let $A = (A, (f^A_a)_{a \in \Sigma})$ be a $\Sigma$-algebra. An equivalence relation $\simeq$ on $A$ is a congruence with respect to $A$ if $f^A_a(x_1, \ldots, x_k) \equiv f^A_a(x'_1, \ldots, x'_k)$ for all $x_1, \ldots, x_k, x'_1, \ldots, x'_k \in A$ such that $x_1 \equiv x'_1, \ldots, x_k \equiv x'_k$. Equivalently, $\simeq$ is compatible with the operations of $A$.

Intuitively, a homomorphism is a map between the carrier sets of a pair of algebras that preserves all relevant structure. Let $\Sigma$ be a set, and let $A = (A, (f^A_a)_{a \in \Sigma})$ and $B = (B, (f^B_a)_{a \in \Sigma})$ be $\Sigma$-algebras. A homomorphism from $A$ to $B$ is a map $\phi : A \to B$ such that $\phi(f^A_a(x_1, \ldots, x_k)) = f^B_a(\phi(x_1), \ldots, \phi(x_k))$ for every $a \in \Sigma$. An isomorphism is a bijective homomorphism.

**Definition 2.1.3 (Tree)** The set $T_\Sigma$ of trees over $\Sigma$ is the smallest set of strings such that $a[t_1, \ldots, t_k]$ is in $T_\Sigma$ whenever $a$ is in $\Sigma(\ell)$ and $t_1, \ldots, t_k$ are in $T_\Sigma$. A tree language over $\Sigma$ is a subset of $T_\Sigma$.

To improve readability, we omit the brackets if $f$ is of rank less than or equal to one, e.g. the tree $f_1[f_2[. . . [f_n]]]$ is written $f_1f_2 . . . f_n$. The delimiters '(', ')', and '{,' may of course be replaced by any other three symbols, but to avoid confusion, one should take care that they are not in $\Sigma$.

Although we usually choose to work with the string representation of a tree, but some tasks are more easily accomplished when the tree is in the guise of a graph. Given a tree $t = f[t_1, \ldots, t_k] \in T_\Sigma$, the node set $\text{nodes}(t)$ of $t$ is $\{\lambda\} \cup \{iv \mid 1 \leq i \leq k \text{ and } v \in \text{nodes}(t_i)\}$. For a node $v \in \text{nodes}(t)$, the symbol $t(v)$ that labels $t$ and the subtree $t/v$ rooted at $v$ are inductively defined by

- $t(\lambda) = f$ and $t/\lambda = t$, and
- $t(iv) = t_i(v)$ and $t/iv = t_i/v$ for $i \in [k]$ and $v \in \text{nodes}(t_i)$.

The root of $t$ is the node $\lambda$, and the leaves of $t$ are the nodes in $\text{nodes}(t)$ that are labelled with symbols in $\Sigma(0)$. The height of $t$ is $\max_{v \in \text{nodes}(t)} |v|$, and the size of $t$ is $|\text{nodes}(t)|$. The set of paths through the tree $t \in T_\Sigma$ is a subset of $\Sigma^+ \cup \Sigma^*$ defined as follows: if $t = a$ for some $a \in \Sigma(0)$, then $\text{paths}(t) = a$. Otherwise, $t$ is of the form $f[t_1 \cdots t_k]$ for some $f \in \Sigma(\ell)$, where $k \geq 1$, and $t_1 \cdots t_k \in T_\Sigma$, in which case $\text{paths}(t) = \{fw \mid w \in \text{paths}(t_i)\text{ for some } i \in [k]\}$. To be able to address specific positions in a tree, we assume that we have access to a countably infinite set of variables $X = \{x_1, x_2, \ldots\}$. All variables have rank zero, and are by convention never used as ordinary symbols. The subset $\{x_1, \ldots, x_k\}$, where $k \in \mathbb{N}$, of $X$ is written $X_k$. Note that $X_k$ is a ranked alphabet of the form $\Sigma(\ell)$ for every $k \in \mathbb{N}$.

We often want to to assemble or take apart a tree piece-by-piece. For this purpose substitution comes in handy: if $t$ is a tree in $T_{\Sigma \cup X_k}$, then we denote by $t[t_1, \ldots, t_k]$ the tree that results when each occurrence of $x_i$ in $t$ is replaced by $t_i$, and this is done simultaneously for every $i \in \{1, \ldots, k\}$. A tree $c \in T_{\Sigma \cup X}$
in which \( x_1 \) occurs exactly once is called a **context** (over \( \Sigma \)). The set of all contexts over \( \Sigma \) is denoted by \( C_\Sigma \).

Another way of putting trees together is through **top-concatenation**:

**Definition 2.1.4 (Top-concatenation)** For each symbol \( a \in \Sigma_{(k)} \), where \( k \geq 1 \), we define the operation of **top-concatenation** with \( a \), denoted by \( tc_a \), to be the mapping from \( T^k_\Sigma \) to \( T_\Sigma \) such that, for all \( t_1, \ldots, t_k \in T_\Sigma \),

\[
    tc_a(t_1, \ldots, t_k) = a[t_1, \ldots, t_k] .
\]

Moreover, for tree languages \( L_1, \ldots, L_k \), we define

\[
    tc_a(L_1, \ldots, L_k) = \{ a[t_1, \ldots, t_k] \mid t_i \in L_i, i \in [k] \} .
\]

As pointed out in [Eng74], every tree in \( T_\Sigma \) can be compiled in a unique way from the symbols in \( \Sigma_{(0)} \) through the repeated top-concatenation with symbols in \( \Sigma \) of rank greater than zero. In other words, \( (T_\Sigma, (tc_a)_{a \in \Sigma}) \) is a \( \Sigma \)-algebra, and an important one at that. From now on we refer to it as the **free term algebra** over \( \Sigma \). As it turns out, every congruence of this algebra is also a congruence with respect to substitutions into contexts over \( \Sigma \).

**Lemma 2.1.5** Let \( \cong \) be a congruence of the \( \Sigma \)-algebra \( (T_\Sigma, (tc_a)_{a \in \Sigma}) \), and let \( t \) and \( t' \) be trees in \( T_\Sigma \). If \( t \cong t' \), then \( c[t] \cong c[t'] \) for every \( c \in C_\Sigma \).

The straightforward proof (by structural induction on \( c \)) is omitted.

### 2.2 Regular tree languages and their automata

Recall that a **tree language** over the alphabet \( \Sigma \) is a subset of \( T_\Sigma \). As most interesting tree languages are infinite, one must resort to algebraic and/or formal devices to obtain finite representations. This allows for a taxonomy of the tree languages over \( \Sigma \) based on the complexity of the devices needed to express them. A class that thus arises is the class **regular** tree languages (rtl). The claim that this is a very natural and important class is strengthened by the great number of ways in which it can be characterised. The definition that we give here is of an algebraic nature.

**Definition 2.2.1** A tree language \( L \) over \( \Sigma \) is **regular** if there is a congruence \( \cong \) of the \( \Sigma \)-algebra \( (T_\Sigma, (tc_a)_{a \in \Sigma}) \) such that

1. \( \cong \) is of finite index, and
2. \( \cong \) saturates \( L \).

The membership problem associated with every regular tree language is decided by a finite tree automaton, a recognising device introduced independently by [Don65, Don70] and [TW65, TW68]. Finite tree automata, or fta for short,
have a well-developed theory and are the subject of Gecseg & Steinby’s classical book *Tree Automata* [GS84], and also of *Tree Automata Techniques and Applications* [CDG+97]. We shall only concern ourselves with fta that traverse their input bottom-up; it is well known (see for example, [GS84, CDG+97, Eng74]) that this model is equivalent – in terms of the set of languages that can be represented – to nondeterministic fta that operate top-down, whereas deterministic top-down fta cannot even recognise every finite language.

Hence, an fta processes its input tree *t* bottom-up, starting at the leaves and working its way up towards the root. When the entire tree has been read, the automaton either accepts or rejects *t* as a member of its target language. The manner in which information is propagated during the run of a finite automata is quite simple, and this gives the device many nice mathematical properties.

**Definition 2.2.2 (Finite tree automaton)** A finite tree automaton, or *fta* for short, is a quadruple $M = (Q, \Sigma, (\delta_a)_{a \in \Sigma}, F)$, where $Q$ is a finite set of states, $\Sigma$ is a ranked alphabet, and $(\delta_a)_{a \in \Sigma}$ is a family of transition relations: for each $a \in \Sigma^+(k)$, $\delta_a$ is a mapping from $Q^k$ to the power set $\mathcal{P}(Q)$ of $Q$. The subset $F$ of $Q$ contains the accepting states.

The set of operations $(\delta_a)_{a \in \Sigma}$ induces a mapping $\delta$ from $T_\Sigma$ to $\mathcal{P}(Q)$ in the canonical way. To be precise, for $t = a[t_1, \ldots, t_k]$ in $T_\Sigma$,

$$\delta(t) = \{ q \mid q \in \delta_a(q_1, \ldots, q_k) \text{ and } q_i \in \delta(t_i) \text{ for all } i \in [k] \}.$$  

The tree language recognised by $M$ is $L(M) = \{ t \mid \delta(t) \cap F \neq \emptyset \}$. The fta $M$ is deterministic if $|\delta_a(q_1 \cdots q_k)| \leq 1$ for every $a \in \Sigma^+(k)$ and $q_1 \cdots q_k \in Q$. Similarly, $M$ is total if $|\delta_a(q_1 \cdots q_k)| \geq 1$ for every $a \in \Sigma^+(k)$ and $q_1 \cdots q_k \in Q$. Both properties are inherited by the induced mapping $\delta$, so if $M$ is deterministic (respectively, total) mapping, then every tree $t \in T_\Sigma$ is mapped by $\delta$ to at most one (respectively, at least one) state in $Q$.

The semantics of an fta $M = (Q, \Sigma, (\delta_a)_{a \in \Sigma}, F)$ can also be described in the following, equivalent way. A run of $M$ on input $t$ is a mapping $\text{run}_t$ from nodes($t$) to $Q$ that agrees with $(\delta_a)_{a \in \Sigma}$: if $v$ is in nodes($t$) and $t(v)$ is in $\Sigma^+(k)$, then run$_t$($v$) must be in $\delta_{t(v)}(\text{run}_t(v1), \ldots, \text{run}_t(vk))$. A run $\text{run}_t$ on the tree $t$ is said to be accepting if $\text{run}_t(\lambda)$ is in $F$. The language recognised by $M$ is the set of trees on which $M$ has at least one accepting run.

In the declaration of an fta, the exact names of the states are not important, as long as they serve as unique identifiers. Two fta are are language equivalent if they recognise the same language. The size of an fta $M = (\Sigma, Q, (\delta_a)_{a \in \Sigma}, F)$ is given by

$$|Q| + \sum_{a \in \Sigma} |\delta_a|.$$  

It follows that if the fta $M$ is both total and deterministic, then its size is uniquely determined by the ranked alphabet $\Sigma$ and the cardinality of $Q$. 

7
Figure 2: The computation finishes in a rejecting state because of the offending path, which is indicated with a dashed line in the input tree.

Example 2.2.3 In the following, $\Sigma = \{a^{(2)}, b^{(2)}, c^{(2)}, d^{(0)}\}$ is a ranked alphabet, and $L = \{t \mid \text{paths}(t) \subseteq a^+b^+c^+d\}$ a tree language over $\Sigma$. Intuitively, every tree in $L$ contains of four layers; a top layer that consists of $a$’s, a second layer of $b$’s, a third of $c$’s, and a bottom layer of $d$’s. To recognise $L$ with a total deterministic fta $M$, five states are needed. The transition function $\delta$ of $M$ maps the input tree $t \in T_\Sigma$ to:

- $q_d$ if and only if $t$ is equal to $d$.
- $q_c$ if and only if $\text{paths}(t)$ is a subset of $c^+d$.
- $q_b$ if and only if $\text{paths}(t)$ is a subset of $b^+c^+d$.
- $q_a$ if and only if $\text{paths}(t)$ is a subset of $a^+b^+c^+d$.
- $q_\perp$ for all other $t$ in $T_\Sigma$.

Among these states, only $q_a$ is final. The transition function is as follows:

\[
\begin{align*}
\delta_d() &= q_d \\
\delta_c(q_x, q_y) &= q_c & \text{for } x, y \in \{c, d\} \\
\delta_b(q_x, q_y) &= q_b & \text{for } x, y \in \{b, c\} \\
\delta_a(q_x, q_y) &= q_a & \text{for } x, y \in \{a, b\} \\
\delta_z(q_x, q_y) &= q_\perp & \text{for every other combination of } x, y, z \in \{a, b, c, d\}
\end{align*}
\]

A (rejecting) computation is illustrated in Figure 2, where the transitions “consume” the tree stepwise from the leaves towards the root. The notation $t \xrightarrow{n} s$ should be interpreted as “$s$ is obtained from $t$ after $n$ applications of transitions in $\delta$”. As the state $q_\perp$ in which this computation finishes is not among the accepting states, the input tree $t$ is not accepted. $\diamond$

It is common to add the epithet *nondeterministic* to the term finite tree automata, or use the abbreviation nfta, to stress that unrestricted fta are considered, as opposed to deterministic fta (dfta). However, as long as we are only
concerned with the set of languages that can be recognised, this discrimination is unnecessary. As in the string case, every fta can be converted to an equivalent dfta, i.e. one the recognises the same tree language, through a subset construction of the state space [Eil74, TW68, Don70].

**Theorem 2.2.4** Every fta has a language equivalent dfta.

The proof of the theorem is analogous to the well known string case, and is therefore omitted. In fact, the monadic case is basically equal to the string case. As a consequence, the well-known examples from the string case can be used to show that an exponential surge in number of states when going from an nfta to an equivalent dfta cannot be avoided.

We now describe the relation between regular tree languages and the tree languages that are recognisable by finite tree automata.

**Definition 2.2.5 (The automaton \( M_{\cong} \))** Let \( L \) be a tree language, and let \( \cong \) be a congruence of finite index that saturates \( L \). We denote by \( M_{(L, \cong)} \) (or \( M_{\cong} \) if \( L \) is understood) the system \((Q, \Sigma, (\delta_a)_{a \in \Sigma}, F)\), where

\[
\begin{align*}
Q &= \{ [t] \mid t \in T_\Sigma \}, \\
[t] \cong & \text{ is in } F \text{ iff } t \text{ is in } L, \text{ and} \\
\delta_a & \text{ is given by} \\
\delta_a([t_1] \cong, \ldots, [t_k] \cong) &= [a[t_1, \ldots, t_k]] \cong,
\end{align*}
\]

for every \( a \in \Sigma^{(k)} \) and \( t_1, \ldots, t_k \in T_\Sigma \).

**Lemma 2.2.6** If \( L \) is a tree language, and \( \cong \) is a congruence of finite index that saturates \( L \), then \( M_{(L, \cong)} \) is a dfta recognising \( L \).

Although the proof of the lemma is well known, let us state it explicitly, because of its importance for several papers in this thesis.

**Proof.** We first show that \( M \) is well-defined, and then argue that it recognises \( L \). The relation \( \cong \) is, by definition, of finite index, so we know that \( Q \) is a finite set, and since \( t \cong t' \) implies that \( t \) is in \( L \) if and only if \( t' \) is in \( L' \), the definition of \( F \) is sound. The condition that \( \cong \) is compatible with top concatenation guarantees that \((\delta_a)_{a \in \Sigma} \) is a family of functions. Furthermore, it can be shown by structural induction that \( \delta \) maps every tree \( t = a[t_1, \ldots, t_k] \) in \( T_\Sigma \) to \([t] \cong\),

\[
\delta(t) = \delta(a[t_1, \ldots, t_k]) = \delta_a(\delta(t_1), \ldots, \delta(t_k)) \quad \text{(by definition of } \delta) \\
= \delta_a([t_1] \cong, \ldots, [t_k] \cong) \quad \text{(by the induction hypothesis)} \\
= [a[t_1, \ldots, t_k]] \cong \quad \text{(by definition of } \delta_a) \\
= [t] \cong.
\]

Since \([t] \cong \) is accepting if and only \( t \in L \), it follows that \( L(M_{(L, \cong)}) = L \). \( \square \)
Definition 2.2.7 (\(\cong_M\)) Let \(M\) be a deterministic fta with transition function \(\delta\). The relation \(\cong_M\) on \(T_\Sigma\) is given by \(t \cong_M t'\) if and only if \(\delta(t) = \delta(t')\). 

Lemma 2.2.8 If \(M\) is a deterministic fta recognising \(L\), then \(\cong_M\) is a congruence relation with respect to top-concatenation. Moreover, \(\cong_M\) is of finite index and saturates \(L\).

Again, let us make the argument explicit.

Proof. Let \(M = (\Sigma, Q, \delta, F)\) be a tree automaton. Since \(\delta\) is a function, each tree \(t\) is mapped to exactly one state, so \(\cong_M\) is clearly an equivalence relation, and there is a bijection between its equivalence classes and the states of \(M\). Since \(Q\) is finite, so is the index of \(\cong_M\). It remains to show that \(\cong_M\) is compatible with top-concatenation. Let \(a\) be a arbitrary symbol in \(\Sigma\), and let \(t_1, \ldots, t_k, t'_1, \ldots, t'_k\) be such that \(t_i \cong_M t'_i\) for all \(i \in [k]\). Then,

\[
\delta(a[t_1, \ldots, t_k]) = \delta_a(\delta(t_1), \ldots, \delta(t_k)) \quad \text{(by definition of } \delta) \\
= \delta_a(\delta(t'_1), \ldots, \delta(t'_k)) \quad \text{(since } t_i \cong_M t'_i \text{ for ever } i \in [k]) \\
= \delta(a[t'_1, \ldots, t'_k]) \quad \text{(by definition of } \delta) .
\]

It follows that \(a[t_1, \ldots, t_k] \cong_M a[t'_1, \ldots, t'_k]\), so the relation \(\cong\) is a congruence with respect to top-concatenation. 

If a tree language \(L\) is accepted by an fta, then according to Theorem 2.2.4 it is also accepted by a deterministic fta \(M\). According to Lemma 2.2.8, \(\cong_M\) is a congruence of finite index that saturates \(L\), so \(L\) is a regular tree language. On the other hand, if \(L\) is a regular tree language, then there is a congruence \(\cong\) of finite index that saturates it, and by Lemma 2.2.6 \(M(L, \cong)\) is a dfta that recognises \(L\). This argument yields Theorem 2.2.9.

Theorem 2.2.9 A tree language is regular iff it is recognised by an fta.

Because of this result it is permissible to use the term recognisable synonymously with regular when speaking about tree languages.

2.3 Minimisation of finite tree automata

From what we learnt in the previous section, it is easy to see that every regular tree language can be represented by an infinite number of pairwise distinct finite tree automata. In this section we search for a lower bound on the size of an fta that recognises a regular tree language \(L\). For this purpose, we formalise what it means for a pair of fta to be identical by extending the notion of isomorphism: two fta that share the same input alphabet are isomorphic if there is an isomorphism \(\phi\) between their state spaces that respects their transition functions and also the accepting states.

A question that is interesting both from a theoretical and a practical point of view is the following: is there, for every regular tree language \(L\), a unique (up
to isomorphism) minimal fta that recognises $L$, and if so, is there a polynomial-
time algorithm that finds it? It is obvious that for every rtl $L$, there is a least
natural number $n$ such that $L$ is recognised by an fta with $n$ states, but from
here on things get worse. According to a well-known result from string theory
that also generalises to trees, this fta is not unique, and finding any minimal
fta is PSPACE complete [JR93].

What happens then if we try to simplify the problem by only considering
deterministic tree automata? According to Theorem 2.2.9, any dfta $M$ can be
identified with a congruence $\cong_A$ that satisfies the conditions of Lemma 2.2.1,
and the index of this congruence is equal to the size of the state space of $M$. In
other words, the coarser the congruence, the smaller the associated dfta. The
search for a minimal dfta recognising a regular tree language $L$ can thus be
reduced to establishing the coarsest congruence that $L$ allows.

Definition 2.3.1 The language congruence $\cong_L$ of a (possibly non-regular) tree
language $L$ over $\Sigma$ is the following congruence on the trees $T_\Sigma$: The trees $t$ and
$t'$ in $T_\Sigma$ are in the relation $\cong_L$ if, for every $c \in C_\Sigma$, it holds that $c[t]$ is in $L$ if
and only if $c[t']$ is in $L$.

Lemma 2.3.2 Every congruence $\cong$ that saturates $L$ is a refinement of $\cong_L$.

Proof. Let $\cong$ be a congruence of that saturates $L$. Suppose that $t \cong t'$ but
that $t \not\cong_L t'$. Under these conditions, there is a $c \in C_\Sigma$ such exactly one of
$c[t]$ and $c[t']$ is in $L$. However, as $\cong$ is compatible with top-concatenation,
$c[t] \cong c[t']$ and this contradicts the assumption that $\cong$ saturates $L$. ◊

As a consequence of the results mentioned, we get the following corollary.

Corollary 2.3.3 $L$ is a regular language if and only if $\cong_L$ is of finite index.
Moreover, the dfta $M_L = M(L, \cong_L)$ is the unique minimal dfta (up to isomor-
phism) that recognises $L$.

The first statement follows from Definition 2.2.1 and Lemma 2.3.2 for the
only if direction, and from Lemma 2.2.6 for the if direction. For the second
statement, if $M$ is a minimal dfta that recognises $L$, by Lemma 2.3.2, the
associated congruence $\cong_M$ is a refinement of $\cong_L$, and since $M$ is minimal, the
two congruences coincide. It follows that $M = M_{\cong_M} = M_{\cong_L} = M_L$.

Combining Theorem 2.2.9 and Corollary 2.3.3 we obtain the famous Myhill-
Nerode theorem which was originally proved for strings.

Theorem 2.3.4 (Cf. [Myh57], [Ner58]) Given a tree language $L$, the fol-
lowing statements are equivalent.

1. The congruence $\cong_L$ is of finite index.
2. $L$ is recognised by a finite tree automaton.
3. There is a congruence $\cong$ of the free term algebra, such that $\cong$ is of finite
index and saturates $L$. 

11
2.4 Regular tree grammars

For some applications it is more convenient to have a generative representation than an accepting. This is for example true when we are working with language synthesis or model checking. The generative device that one usually associates with regular tree languages is the regular tree grammar (rtg) which compiles trees top-down [Bra68]. The rtg can be seen as a generalisation of the regular (string) grammar [HMU00]; for monadic alphabets the two devices coincide.

Definition 2.4.1 (Regular tree grammar) A regular tree grammar G is a quadruple \((N, \Sigma, P, S)\), where

- \(N\) is a ranked alphabet of nonterminals, all of rank zero,
- \(\Sigma\) is an alphabet of terminals such that \(N \cap \Sigma = \emptyset\),
- \(P\) is a finite set of productions of the form \(A \rightarrow t\), where \(A \in N\) and \(t \in T_{\Sigma \cup N}\), and
- \(S \subseteq N\) is the set of initial nonterminals.

Let \(G = (N, \Sigma, P, S)\) be an rtg, and let \(s\) and \(s'\) be trees in \(T_{\Sigma \cup N}\). There is a derivation step in \(G\) from \(s\) to \(s'\), written \(s \Rightarrow_G s'\), if \(s'\) can be obtained by replacing one occurrence of a nonterminal in \(s\) according to a production in \(P\). The transitive closure of \(\Rightarrow\) is denoted by \(\Rightarrow^*\), and the language \(L(G)\) generated by \(G\) is

\[ L(G) = \{ t \in T_\Sigma \mid S' \Rightarrow^*_G t \text{ for some } S' \in S \} \] .

A regular tree grammar is in normal form if each of its productions is of the form \(A \rightarrow a[A_1, \ldots, A_k]\), where \(a\) is a terminal of rank \(k\), and \(A, A_1, \ldots, A_k\) are nonterminals. It is not difficult to show that every rtg can be rewritten in normal form without affecting the generated language.

A tree language is accepted by a finite tree automaton if and only if it is generated by a regular tree grammar. In fact, every fta can be seen as an rtg in normal form, and vice versa: the input alphabet of the automaton is the output alphabet of the grammar, the states correspond to the nonterminals, and accepting states to the initial nonterminals. The only translation that requires any effort is the interpretation of the family of transition relations as a set of productions, but even that is straightforward: Given a family of transition relations \((\delta_a)_{a \in \Sigma}\),

\[ P_\delta = \{ q \rightarrow a[q_1, \ldots, q_k] \mid a \in \Sigma_{(k)} \text{ and } q \in \delta_a(q_1, \ldots, q_k) \} \] ,

is an equivalent set of productions, and given a set \(P\) of productions, \((\delta_a)_{a \in \Sigma}\),

\[ \delta_a(q_1, \ldots, q_k) = \{ q \mid q \rightarrow a[q_1, \ldots, q_k] \in P \} \]

is an equivalent family of transition relations.

Regular tree grammars are widely used in natural language theory, and this popularity can be explained by their close relationship to the context-free
languages [Cho56]. The connection between regular tree languages and \( \lambda \)-free context-free languages is that the latter are the yields of the former. We illustrate this rather obvious fact by means of an example.

**Example 2.4.2** Let \( \Sigma = \{a^{(0)}, b^{(0)}, c^{(2)}, d^{(3)}\} \) be a ranked alphabet, and let \( G = (N, \Sigma, P, N) \) be an rtg, where \( N = \{S\} \) and \( P \) is given by

\[
S \rightarrow d[a, S, b], \quad S \rightarrow d[b, S, a], \quad \text{and} \quad S \rightarrow c[S, S].
\]

A non-empty string \( w \) is in the language \( \text{yield}(L(G)) \subset \{a, b\}^* \) if and only if it contains equally many \( a \)'s and \( b \)'s. To show that \( \text{yield}(L(G)) \) is a context-free string language it suffices to find a context-free grammar \( G' \) that generates \( \text{yield}(L(G)) \), but this is easy: nonterminals and terminals are \( N \) and \( \Sigma^{(0)} \), respectively, where \( S \) is the initial nonterminal, and the set of productions is

\[
P' = \{ A \rightarrow \text{yield}(t) \mid A \rightarrow t \in P \}.
\]

In this particular case, \( P' \) would thus contain the productions

\[
S \rightarrow aSb, \quad S \rightarrow bSa, \quad \text{and} \quad S \rightarrow SS.
\]

As indicated by Example 2.4.2, the yield of a regular tree language is a context-free language, and it is not difficult to see why also the opposite direction holds. The general theorem reads as follows:

**Theorem 2.4.3 (Cf. [Tha67])** A \( \lambda \)-free string language \( L \) is context-free if and only if it is the yield of a regular tree language.

**2.5 Tree series**

Tree languages have many uses, but they can only describe qualitative properties of trees, and for some applications this is just not sufficient. In [MPR96], Mohri, Pereira, and Riley argue that speech processing requires a more general model, one in which also quantitative properties can be expressed. It would for example be useful to be able to accept a tree as a member of some target language with certainty \( p \), or at a cost \( c \). The objective is thus to associate every tree over some ranked alphabet \( \Sigma \) with a weight in some domain \( \mathcal{K} \), i.e. to define a mapping or tree series from \( T_\Sigma \) to \( \mathcal{K} \). In particular, every tree language \( L \subseteq T_\Sigma \) is a tree series with weights in the Boolean semiring: the weight of \( t \in T_\Sigma \) is \( \text{true} \) if \( t \in L \) and \( \text{false} \) otherwise. In preparation for the formal definition of tree series, we recall a bit of algebra.

A **monoid** is a set \( A \) together with an associative operation \( \odot : A \times A \rightarrow A \) and a **neutral** element \( 1 \) in \( A \); i.e., we have \( a \odot 1 = 1 \odot a = a \), for all \( a \in A \). If, in addition, \( a \odot b = b \odot a \), for all \( a, b \) in \( A \), then \((A, \odot, 1)\) is a said to be **commutative**. For instance, the set of strings over some alphabet together with concatenation forms a monoid in which the empty string is the neutral element.

A **semiring** is a system \((A, \oplus, \odot, 0, 1)\) where \((A, \oplus, 0)\) is a commutative monoid and \((A, \odot, 1)\) is a monoid. It addition, the following must hold:
- For every \( a, b, \) and \( c \) in \( A \), it holds that \((a \oplus b) \odot c = (a \odot c) \oplus (b \odot c)\) and that \( a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)\), i.e. the multiplicative operation distributes over the additive.

- For every \( a \) in \( A \), we have that \( a \odot 0 = 0 \odot a = 0 \). In other words, the neutral element with respect to \( \oplus \) is absorptive.

A semiring is commutative if also its multiplicative monoid is commutative, and zero-divisor free if \( a \odot b = 0 \) implies \( 0 \in \{a, b\} \). There are many natural examples of semirings, e.g. the natural numbers (including zero) with ordinary addition and multiplication, and the polynomials over the variable \( x \) with natural number coefficients. Also the Boolean set \( \{true, false\} \) with logical-or and logical-and is a semiring, and so is the set of square \( n \)-by-\( n \) matrices over a semiring under matrix addition and multiplication.

**Definition 2.5.1** A tree series over the ranked alphabet \( \Sigma \) and the semiring \( K = (A, \oplus, \odot, 0, 1) \) is a mapping from \( T_\Sigma \) to \( A \).

In the previous chapter on tree languages, most of our attention was devoted to the regular tree languages. As the reader may recall, these are exactly those which are recognisable by finite tree automata. The more general device that is obtained by associating each transition in an fta with a weight is called weighted tree automaton, or wta, for short. There are many different models of wta described in the literature, see for example [BR82, Boz99, BV03, Mal05], and a comparison between several of these is found in [FV06]. In this section we adhere to the definition of wta by Berstel and Reutenauer [BR82] that was originally given for semifields, but later generalised to semirings in [Boz97, Kui97]. During the run of a wta \( M \) on an input tree, weight accumulates linearly when transitions are made. The wta maps its input to a value in \( A \) by taking sums over the weights of its runs on \( t \). Hence, \( M \) defines a tree series. Any tree series that can be represented by a wta is said to be recognisable [Bor05].

For the remainder of this section, we adopt the following notational convention. Let \( K \) and \( Q \) be sets, then \( K^Q \) denotes the set of all \( Q \)-indexed families with elements in \( K \), i.e. the set

\[
\{(a_q)_{q \in Q} \mid a_q \in K \text{ for every } q \in Q\}
\]

Given the index family \( v = (a_q)_{q \in Q} \), we write \( v_q \) to specify the element \( a_q \).

**Definition 2.5.2** A weighted tree automaton \( M \) is a system \((Q, \Sigma, K, F, \mu)\), where \( Q \) and \( \Sigma \) are as in the unweighted case, \( K = (K, +, \odot, 0, 1) \) is a semiring, \( F \in K^Q \) is a final weight distribution, and \( \mu = (\mu_a)_{a \in \Sigma} \), with \( \mu_a : Q^k \to K^Q \) for each \( a \in \Sigma_{(k)} \), is a tree representation.

The initial algebra semantics of the wta \( M \) is determined by the mapping \( h_\mu \) that takes \( T_\Sigma \) to \( K^Q \) and is given by

\[
h_\mu(a[t_1, \ldots, t_k])_q = \bigoplus_{q_1, \ldots, q_k \in Q} \mu_a(q_1, \ldots, q_k)_q \odot h_\mu(t_1)_{q_1} \odot \ldots \odot h_\mu(t_k)_{q_k}
\]
for every \( a \in \Sigma(k) \), \( q \in Q \), and trees \( t_1, \ldots, t_k \in T_{\Sigma} \). The tree series recognised by \( M \), denoted by \( S_M \), is defined by

\[
S_M(t) = \bigoplus_{q \in Q} F_q \Diamond h_\mu(t)_q
\]

for every tree \( t \in T_{\Sigma} \). The wta \( M \) is deterministic if, for every \( a \in \Sigma(k) \), and \( q_1, \ldots, q_k \in Q \), there is at most one \( q \in Q \) such that \( \mu_a(q_1, \ldots, q_k)_q \neq 0 \). \( \diamondsuit \)

Since \( \Diamond \) distributes over \( \oplus \), we can give the semantics of a weighted tree automaton \( M = (Q, \Sigma, K, F, (\mu_a)_{a \in \Sigma}) \) in an alternative way: A run of \( M \) on an input tree \( t \) is a mapping \( \text{run}_t \) from \( \text{nodes}(t) \) to \( Q \). The weight of a run \( \text{run}_t \) with respect to a node \( v \in \text{nodes}(t) \) is

\[
\text{wt}(\text{run}_t, v) = \mu_t(v)(\text{run}_t(v1), \ldots, \text{run}_t(vk)) \otimes \bigoplus_{i \in [k]} \text{wt}(\text{run}_t, vi)
\]

where \( t(v) \in \Sigma(k) \).\(^1\) Moreover, the total weight of a run \( \text{run}_t \) is computed to \( F_{\text{run}_t(\lambda)} \otimes \text{wt}(\text{run}_t, \lambda) \). Given a wta \( M \) and a tree \( t \in T_{\Sigma} \), the value of \( S_M(t) \) is obtained by taking the semiring sum over the weights of all runs of \( M \) on \( t \). As there is nothing that corresponds to epsilon transitions in a wta, the number of distinct runs of \( M \) on \( t \) is exponential in the size of \( t \), so the mentioned sum is finite and thus well defined.

**Example 2.5.3** Let \( \Sigma \) and \( L \) be as in Example 2.2.3. Given a tree \( t \in T_{\Sigma} \), we denote by \( |t|_b \) the number of occurrences of the symbol \( b \) in \( t \). We now consider the tree series \( S \) over the natural numbers (with ordinary addition and multiplication) that maps \( t \) to \( |t|_b \) if \( t \) is in \( L \), and to zero otherwise. To recognise \( S \), the wta \( M \) uses six states: \( q_a, q^*_a, q_b, q^*_b, q_c, \) and \( q_d \). The tagged versions \( q^*_a \) and \( q^*_b \) of \( q_a \) and \( q_b \) are used to remember that the automaton has seen a \( b \) that it wants to count. The final weight of the state \( q^*_a \) is one, but every other state has final weight zero. Let \( (\delta_a)_{a \in \Sigma} \) be as in Example 2.2.3. Below, we list the entries in the tree representation \( \mu \) that have weight one. Every other entry in \( \mu \) has weight zero.

\[
\begin{align*}
\mu_c(q_x, q_y)_{q_c} & \text{ if } \delta_c(q_x, q_y) = q_c \\
\mu_b(q_x, q_y)_{q_b} & \text{ if } \delta_b(q_x, q_y) = q_b \\
\mu_b(q_x, q_y)_{q^*_b} & \text{ if } \delta_b(q_x, q_y) = q_b \\
\mu_b(q^*_x, q_y)_{q^*_b} & \text{ if } \delta_b(q_x, q_y) = q_b \\
\mu_b(q_x, q^*_y)_{q^*_b} & \text{ if } \delta_b(q_x, q_y) = q_b \\
\end{align*}
\]

When a \( b \) is read, the automaton can go into a state marked \(*\), and this mark is then propagated towards the root.\(^2\) If two different \( b \)'s have caused a

\(^1\)Here, \( \bigotimes_{i \in [k]} a_i = a_1 \Diamond \cdots \Diamond a_k \).

\(^2\)If we say that \( M \) can (only) go into certain states, this is because no other runs will contribute to the result, because they all use a zero entry of the tree representation and thus have total weight zero.
star to appear, then the overall weight of the run will be zero, because there are no non-zero entries in the tree representation to handle this case. Moreover, if the run finishes in a state that is not labelled by a star, then the weight of the run is multiplied with the final weight zero. Given an input tree \( t \), there are \( |t|_b \) distinct runs of \( M \) on \( t \) that each has the total weight one; every other run of \( M \) on \( t \) has the total weight zero. Hence, \( t \) is mapped to \( |t|_b \).

Definition 2.5.2 can also be explained in terms of homomorphisms between algebraic structures. Let \( M = (Q, \Sigma, \mathcal{K}, F, \mu) \) be a wta over the ranked alphabet \( \Sigma \) and the semiring \( \mathcal{K} \). The tree representation \( \mu \) of \( M \) defines an homomorphism \( h_\mu \) from the algebra \((T_\Sigma, (tc_a)_{a \in \Sigma})\) to the algebra \((K^Q, (\overline{\mu}_a)_{a \in \Sigma})\), where, for each \( a \in \Sigma(k) \), the mapping \( \overline{\mu}_a : (K^Q)^k \rightarrow K^Q \) is given by

\[
\overline{\mu}_a(x_1, \ldots, x_k)_q = \bigoplus_{q_1, \ldots, q_k \in Q} \mu_a(q_1, \ldots, q_k)_q \otimes (x_1)_{q_1} \otimes \ldots \otimes (x_k)_{q_k}.
\]

In this more general setting, every fta \( M = (Q, \Sigma, (\delta_a)_{a \in \Sigma}, F) \) can be interpreted as a wta \((Q, \Sigma, \mathbb{B}, F', (\mu_a)_{a \in \Sigma})\) with weights in the Boolean semiring. Its tree representation \( \mu_a : Q^k \rightarrow \mathbb{B}^Q \) is defined as

\[
\mu_a(q_1, \ldots, q_k)_q = \begin{cases} 
\text{true} & \text{if } q \text{ is in } \delta_a(q_1, \ldots, q_k), \text{ and} \\
\text{false} & \text{otherwise.}
\end{cases}
\]

The function \( F' : Q \rightarrow \mathbb{B} \) is given by \( F'(q) = \text{true} \) iff \( q \) is an accepting state.

Let us briefly reconsider the subset construction used in the determinisation of classical fta. In wta notation, this determinisation procedure can be described as follows. Let \( M = (Q, \Sigma, \mathbb{B}, F, (\mu_a)_{a \in \Sigma}) \) be the wta that we wish to determinise. For every \( a \in \Sigma(k) \) and \( P, P_1, \ldots, P_k \subseteq Q \), the function \( \mu_a^{\det} : (\mathbb{B}(Q))^k \rightarrow \mathbb{B}^P(Q) \) is such that \( \mu_a^{\det}(P_1, \ldots, P_k)_P \) is true when \( P \) equals

\[
\{ q \mid \mu_a(q_1, \ldots, q_k)_q \text{ is true for some } q_1 \in P_1, \ldots, q_k \in P_k \},
\]

and false otherwise. Based on \( \mu_a^{\det} \), we now assemble the deterministic wta \( M^{\det} = (\mathbb{B}(Q), \Sigma, \mathbb{B}, F^{\det}, (\mu_a^{\det})_{a \in \Sigma}) \), where

\[
F^{\det}(P) = \bigvee_{q \in P} F(q)
\]

for every \( P \subseteq Q \). It should be clear that \( M^{\det} \) recognises precisely \( S_M \).

In contrast to the case of regular tree languages, there are recognisable tree series that cannot be recognised with deterministic wta. As shown in [Bor04], the mapping \( \text{height} \) is recognisable by a nondeterministic wta over the arctic semiring \((\mathbb{N} \cup \{-\infty\}, \text{max, +, } -\infty, 0)\), but not deterministically so. In fact, \( \text{height} \) is not recognisable by a wta over any number field, even if the wta is allowed to be nondeterministic [BR82]. Another interesting example is the tree
series \textit{zigzag} : T_\Sigma \rightarrow \mathbb{N}, \text{ where } \Sigma = \{a(0), f(2)\}, \text{ which is given by}

\[
\text{zigzag}(a) = 1, \\
\text{zigzag}(f[a, t_2]) = 2, \text{ and} \\
\text{zigzag}(f[f[t_1, t_2], t_3]) = 2 + \text{zigzag}(t_2).
\]

for every \(t_1, t_2, \text{ and } t_3 \in T_\Sigma\). If the underlying algebraic structure is the tropical semiring \((\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)\), then \textit{zigzag} can be computed by a nondeterministic wta, but not by a (bottom-up) deterministic wta [Mal07]. On the other hand, it is not difficult to construct a deterministic top-down wta with only three states that recognises \textit{zigzag}.

In [Bor05], Borchardt presents a generalisation of the subset construction for weighted tree automata. Before examining this construction closer, let us first recall that the \textit{kernel} of a mapping \(f : A \rightarrow B\) is an equivalence relation on \(A\), defined as \textit{kernel}(\(f\)) = \{(\(a_1, a_2\) \in A \times A \mid f(\(a_1\)) = f(\(a_2\))\}.

\textbf{Definition 2.5.4 (Cf. Def. 5.2.1 of [Bor05])} Let \(M = (Q, \Sigma, K, F, \mu)\) be a wta. The \textit{determinisation} of \(M\) yields \(M^{\det} = (Q^{\det}, \Sigma, K^{\det}, F^{\det}, \mu^{\det})\), where

- \(Q^{\det} = (Q/\text{kernel}(h_\mu))\),
- \(F^{\det}(q) = S_M(t)\), for every \(q \in Q\) such that \(q = [t]_{\text{kernel}(h_\mu)}\), and
- it holds that

\[
\mu^{\det}_a(q_1, \ldots, q_k)_q = \begin{cases} 1 & \text{if, for all } i \in [k], \text{ there is a } t_i \in T_\Sigma, \text{ such that} \\
q_i = [t_i]_{\text{kernel}(h_\mu)} \text{ and} \\
q = [a(t_1, \ldots, t_k)]_{\text{kernel}(h_\mu)}, \text{ and} \\
0 & \text{otherwise}, \end{cases}
\]

for every input symbol \(a \in \Sigma_{(k)}\) and sequence \(q_1, \ldots, q_k, q \in Q^{\det}\). 

The construction in Definition 2.5.4 does not always yield an output automaton with a finite state space. However, if the determinisation \(M^{\det}\) of a wta \(M\) has a finite state space, then \(M^{\det}\) is a well-defined deterministic wta that recognises the same tree series as \(M\). This is for instance always the case when the underlying semiring is locally finite (cf. Lemma 6.3.2 of [Bor05]).

In [Bor05], there is also included a Myhill-Nerode like theorem for tree series over commutative semifields, a semifield being a semiring in which every non-zero element has a multiplicative inverse. The congruence \(\cong_S\) that Borchardt associates with the tree series \(S\) is thus defined: a pair of trees \(t\) and \(t'\) in \(T_\Sigma\) are in relation \(\cong_S\) if there are non-zero constants \(a, b\) in \(K\), such that \(a \circ S(c[t]) = b \circ S(c[t'])\) for every context \(c \in C_\Sigma\). Among other things, Borchardt’s Myhill-Nerode theorem states that a tree series \(S\) over a commutative semifield is recognisable by a deterministic wta if and only if \(\cong_S\) is of finite index. The work in [Bor05] is continued in [Mal07] by Maletti, who establishes that if a tree series is deterministically recognisable over a zero-divisor free and commutative semiring, then its Myhill-Nerode congruence is of finite index. It remains an open problem whether the opposite implication is also true.
Example 2.5.5 The following is an adaption of Example 5.1.28 in [Bor05].

Figure 3 (a) shows the nondeterministic wta $M = (Q, \Sigma, F, \text{Arctic}_{sf}, (\mu_a)_{a \in \Sigma})$, where $Q = \{q_1, q_2\}$ and $\Sigma = \{a^{(0)}, f^{(2)}\}$. Here, Arctic$_{sf}$ signifies the semifield $(\mathbb{Z} \cup \{\infty\}, \max, +, -, -\infty, 0)$, an algebraic structure obtained from the arctic semiring by extending the carrier set to include the integers and defining the inverse of every $a \in \mathbb{Z}$ to be $-a$. Moreover, the final weight of every state is one, and the transitions in the diagram should be read in a clockwise direction, starting at the destination arrow. For example, the tree representation of $M$ is such that the entries $\mu_f(q_1, q_2)_{q_1}$ and $\mu_f(q_2, q_1)_{q_2}$ both have weight one. All entries with weight $-\infty$ have been omitted from the figure. The maximum weight of any run of the wta $M$ on an input tree is always $\text{size}(t)$, so this is the weight with which $t$ is accepted. Now, Arctic$_{sf}$ is a commutative semifield, and for every pair of trees $t$ and $t'$ there are constants $a$ and $b$ (namely $\text{size}(t)$ and $\text{size}(t')$) such that $a + S_M(c[t]) = b + S_M(c[t'])$ for every context $c \in C_{\Sigma}$, so it would appear that the Myhill-Nerode congruence of $S_M$ has a single equivalence class, and that this tree series is recognisable by a deterministic wta with a single state. This is indeed so, as the deterministic wta in Figure 3 (b), also over the arctic semifield, recognises precisely $S_M$.

\[\diamond\]

2.6 Tree transductions

A tree is in itself a purely syntactical object – a labelled structure and nothing more. However, in the context of a $\Sigma$-algebra $A = (A, (f_a)_{a \in \Sigma})$, every tree $t \in T_{\Sigma}$ is a term that can be evaluated to an element $val_A(t)$ in the carrier of $A$. If $t$ is of the form $a[t_1, \ldots, t_k]$, then the $A$-evaluation of $t$ is given by $val_A(t) = f_a[val_A(t_1), \ldots, val_A(t_k)]$. The value of a tree is thus uniquely determined by the values of its direct subtrees and the $A$-interpretation of the symbol labelling its root. To illustrate this idea, we consider Example 2.6.1 in which trees are evaluated to strings.
Example 2.6.1 Let $G$ and $\Sigma$ be as in Example 2.4.2. Furthermore, let $A$ be the $\Sigma$-algebra $(\Sigma^*_0, (f_a)_{a \in \Sigma})$, where $f_a$ is a constant with value $a$ for every $a \in \Sigma_{(0)}$, and corresponds to $k$-ary concatenation for every $a$ in $\Sigma_{(k)}$ with $k > 0$. It is easy to see that every tree $t$ in $L(G)$ is mapped by $\text{val}_A$ to the string $\text{yield}(t)$ in $\Sigma^*_0$.

We now examine the case when trees are evaluated to trees. This setup is quite interesting for several reasons. First of all it allows for incremental evaluation. Suppose that we want to represent a language $L$ of objects in some $\Sigma_0$-algebra $A_0$. The direct approach would be to describe $L$ as the set \{val$_{A_0}(t) \mid t \in L(G)\}$, where $G$ is some appropriate device generating trees over $T_{\Sigma_0}$. However, if every tree over $\Sigma_i$ is in the carrier set of some $\Sigma_{i+1}$-algebra $A_i$, for all $i : 0 \leq i \leq k$, then every object in our original target set can be written as val$_{A_0}(\text{val}_{A_1}(\cdots (\text{val}_{A_k}(t)) \cdots ))$, where $t$ is a tree in $T_{\Sigma_{k+1}}$.

One benefit with this more modular approach is that the individual evaluation steps can be made less complex, and that it allows us to model one attribute of the target object at a time.

Let $\Sigma$ and $\Delta$ be ranked alphabets. The interpretation of $T_{\Sigma}$ into $\mathfrak{P}(T_{\Delta})$ can be realized by a tree transducer. Intuitively, a tree transducer is an fta with output. Just as tree automata, some types of tree transducers process their input top-down, and some bottom-up; a comparison between the top-down and bottom-up tree transducers is made in [Eng75]. The definition that we give here is for top-down tree transducers [Rou70, Tha70], as it is this variant that is used in the papers [DH07, Hög05] that constitute Part III of this thesis.

In preparation for the formal definition, let us generalise the notion $T_{\Sigma}$: given a set $T$ of trees, $\Sigma(T)$ denotes the set of all trees of the form $f[t_1, \ldots, t_k]$ such that $f \in \Sigma_{(k)}$ for some $k \in \mathbb{N}$ and $t_1, \ldots, t_k \in T$. Furthermore, the set of trees over $\Sigma$ with subtrees in $T$, denoted by $T_{\Sigma}(T)$, is defined inductively:

i) $T \subseteq T_{\Sigma}(T)$ and
ii) if $f \in \Sigma_{(k)}$ and $t_1, \ldots, t_k \in T_{\Sigma}(T)$, then $f[t_1, \ldots, t_k] \in T_{\Sigma}(T)$.

Note that $T_{\Sigma} \subseteq T_{\Sigma}(T)$ and, in particular, $T_{\Sigma}(\emptyset) = T_{\Sigma}$.

Definition 2.6.2 (Tree transducer, Cf. [Eng74]) A top-down (finite) tree transducer (also td transducer) is a system $td = (Q, \Sigma, \Delta, R, q_0)$, where

- $Q$ is a ranked alphabet of states, all of rank one,
- $\Sigma$ and $\Delta$ are the ranked input and output alphabets, respectively,
- $R$ is a finite set of rewrite rules of the form $q[a[x_1, \ldots, x_k]] \rightarrow t$, where $a \in \Sigma_{(k)}$, $q \in Q$, and $t \in T_{\Delta}(Q(X_k))$,
- $q_0 \in Q$, is the initial state.

Let $td = (Q, \Sigma, \Delta, R, q_0)$ be a td transducer. Given an input tree $t \in T_{\Sigma}$, a computation of $td$ starts with the tree $q_0\, t$ and applies the term rewrite rules in $R$ until a tree in $T_{\Delta}$ is reached. Formally, for trees $t, t' \in T_{\Delta}(Q(T_{\Sigma}))$,
there is a computation step $t \rightarrow_{td} t'$ if $t$ is of the form $s[[ q[t_1, \ldots, t_k]]]$, $t'$ is of the form $s[[ t_1, \ldots, t_k]]$, and there is a rewrite rule $q[a[x_1, \ldots, x_k]] \rightarrow t$ in $R$. The transduction computed by $td$ is the mapping $td: T_\Sigma \rightarrow \mathfrak{P}(T_\Delta)$ such that $td(s) = \{ t \in T_\Delta \mid s \xrightarrow{\ast}_{td} t \}$ for all $s \in T_\Sigma$.

**Example 2.6.3** Let $td = (\{ q \}, \Sigma, \Delta, R, q)$ be a top-down tree transducer with input alphabet $\Sigma = \{ f^{(1)}, a^{(0)} \}$ and output alphabet $\Delta = \{ g^{(2)}, b^{(0)} \}$. The set $R$ contains the two rewrite rules $q[f x_1] \rightarrow g[q x_1, q x_1]$ and $q[a] \rightarrow b$. It is easy to see that the transduction computed by $td$ maps every tree $t$ in $T_\Sigma$ to the fully balanced binary tree in $T_\Delta$ that is of the same height as $t$.

A consequence of the Myhill-Nerode theorem is that there is a constant $c_L$ for every regular language $L$ such that the following holds: if the tree $t$ is in $L$ and there is also a tree of size greater than $t$ in $L$, then there is a tree $t'$ in $L$ of size $\text{size}(t) < \text{size}(t') < \text{size}(t) + c_L$. In other words, every regular tree language exhibits linear growth. Let us now return to the td transducer $td$ in Example 2.6.3. Note that there are no distinct trees $t, t' \in T_\Sigma$ such that $\text{size}(t) = \text{size}(t')$. Furthermore, the size of the single tree in $td(t)$ is exponential in the size of $t$, for every tree $t \in T_\Sigma$. It follows that the set of output trees produced by $td$ exhibit exponential growth, so Example 2.6.3 implies that regular tree languages are not closed under transductions computed by td transducers.

To simplify the description of another composition result, let us denote by $D_n$ the set of languages that can be obtained by applying the composition of $n$ top-down tree transducers to an rtl, and by $U_n$ the corresponding set of languages for bottom-up tree transducers. In [Bak79], Baker proved that $X_n$ is a subset of $Y_{n+1}$ for every $n \in \mathbb{N}$ and $X, Y \in \{ U, D \}$, and conjectured that this hierarchy is strict. This conjecture was later proved true by Engelfriet in [Eng82]. The resulting hierarchy is depicted as a Hasse diagram in Figure 4. In this diagram, each set is represented as a vertex, and there is an edge from $X$ to $Y$ if $X \subset Y$ and there is no $Z$ in the vertex set such that $X \subset Z \subset Y$.

Tree transducers are also of practical interest. Consider an algorithm $\text{ALG}$ that takes as input objects in the carrier set of a $\Sigma$-algebra $A$ and produces objects in the carrier set of a $\Delta$-algebra $B$. At a more abstract level, it may be possible to turn $\text{ALG}$ into a symbolic algorithm in the form of a td transducer $td_{\text{ALG}}$ that maps the elements of $T_\Sigma$ to elements of $T_\Delta$. The applica-
Figure 5: A tree transduction is a symbolic algorithm.

tion of the concrete algorithm \texttt{Alg} to an object \(x\) can then be expressed as 
\(\text{val}_B(\text{td}_\text{Alg}(\text{val}_A^{-1}(x)))\), where \(\text{val}_A^{-1}\) is the inverse of the evaluation mapping \(\text{val}_A\) and corresponds to the (possibly nondeterministic) parsing of the element \(x\). The construction is illustrated in Figure 5. Since each transduction is in itself a concrete algorithm, this abstraction can be done repeatedly [Eng80].

The ideas discussed in the previous paragraph are put to practice in syntax-directed translation: here, translation is done on a structural level by relating parse trees of the source language to parse trees of the target language [Man04]. In other words, to transform a string \(s\) in \(\text{yield}(L_{\text{src}})\) to a string in \(\text{yield}(L_{\text{tar}})\), one first computes a parse tree \(t\) in \(L_{\text{src}}\) for \(s\). This parse tree is then interpreted by a tree transducer as a tree \(t'\) in \(L_{\text{tar}}\), from which the output string is obtained by taking the yield of \(t'\). This translation method is explicit in a paper by Irons titled \textit{A syntax directed compiler for Algol 60} [Iro61] and is formalised in [AU68, AU69] and [LS68].
Chapter 3

Summary of papers

3.1 Part I. Algorithmic learning

The first part of this thesis is devoted to algorithmic learning of regular tree languages. We decided on these languages because their structural properties (captured by the Myhill-Nerode theorem) make the problem feasible. There are many different learning models described in the literature, see the survey [Lee96]. The learning scenario described in Papers I through III follows the so-called MAT model: the objective of the learning algorithm (henceforth, the learner), is to derive the minimal dfta recognising the target language \( U \) by actively querying a minimal adequate teacher. This is an oracle capable of answering membership queries of the form is the tree \( t \) in \( U \)?, but also equivalence queries of the form does the dfta \( A \) recognise \( U \) correctly? In response to the latter type of query, the oracle returns a counterexample, i.e., a tree in the symmetric difference between \( L(A) \) and \( U \), and if such a tree does not exist, a token signalling that the learner has succeeded in its task. The MAT model is reasonable from a theoretical point of view, as both the membership problem and the equivalence problem are decidable for finite tree automata.

Paper I. MAT learning

In Paper I we extend a learning algorithm by Angluin for regular string languages to regular tree languages. As mentioned above, the learning scenario is the MAT model; the goal is to find the minimal dfta that recognises the target language \( U \). As the sought automaton is isomorphic to the dfta \( M_U \), the problem can be reduced to deriving the Myhill-Nerode congruence \( \equiv_U \) associated with \( U \). For this purpose, the learner collects a set of representative trees \( S \), one tree for each equivalence class of \( \equiv_U \), together with a set of contexts \( C \). Intuitively, the latter set provides evidence that the trees in \( S \) do in fact represent distinct equivalence classes. The sets \( S \) and \( C \) are accumulated in an iterative fashion. Initially, \( S \) contains an arbitrary tree (that for technical reasons must be of height zero), and \( C \) contains only the empty context, corresponding to the hypothesis that all trees belong to the same equivalence class. In the \( i \)th iteration, the learner builds an fta based on the contents of \( S \) and \( C \), which it then passes to the MAT oracle (from now on, the teacher) in an equivalence query.

23
Unless the proposed fta was correct, the teacher returns a counterexample \( t \). The learner uses contradiction backtracking [Sha83] by means of membership queries to decompose \( t \) into a context \( c \) and a tree \( s \) in such a way that \( s \) is a representative that the learner was missing, and is consequently added to \( S \). Since the target language is regular, \( \approx_U \) is of finite index, so the algorithm is guaranteed to terminate. Correctness of the resulting fta is automatic from the nature of equivalence queries.

**Paper II. Query learning of regular tree languages: how to avoid dead states**

Paper II presents a more efficient and elegant version of the algorithm in Paper I, one that avoids working with **dead states**, i.e. states from which no accepting states can ever be reached. A dead state has no effect on the recognised language, and can therefore be discarded along with all transitions in which it occurs, the only consequence being that the resulting automaton may no longer be total. As the dfta that our learner derives is minimal, it can contain at most one dead state, but in the worst case all but a linear number of its transitions involve this state. This means that the partial minimal dfta recognising the target language \( U \) is potentially much smaller than the total one. Since the algorithm described in Paper II avoids dead states altogether, its running time can be measured in the size of the minimal partial dfta \( M = (Q, \Sigma, \delta, F) \) recognising \( U \), rather than the the total one. Paper II also includes a more careful complexity analysis which shows that the algorithm executes in time \( O(r|Q||\delta|(m+|Q|)) \), where \( r \) is the maximum rank of \( \Sigma \), and \( m \) is the size of the largest counterexample provided by the teacher.

**Paper III. Extensions of a MAT learner for regular tree languages**

Time is not the only valuable resource that the learner from Papers I and II consumes. Any realisation must somehow account for the MAT oracle, and regardless of whether the rôle of this teacher is assumed by a computer or a person, a learner which only asks a few questions must be preferable to one that asks many. In this vein, it also seems natural to consider equivalence queries to be more expensive than membership queries. In Paper III we discuss a implementation of the learning algorithm, written in Java, together with two extensions aimed at reducing the number of equivalence queries posed. The learner obtained from the first of these extensions reuses each counterexample repeatedly, until it cannot derive any new information from it and is forced to ask the teacher for another. The second extension yields a learner which not only reuses the counterexamples, but also saves them for further use. As in real life, the learner may sometimes get the right answer to a question, but for the wrong reason. This means that even though the learner cannot derive any new information from a counterexample **at the moment**, it may be rewarding
to return to it later, when more is known about the target language. Hence, the second extension revisits the saved counterexamples regularly, in search of information previously overlooked.

3.2 Part II. Bisimulation minimisation

In Part II of this thesis, we approach the problem of minimising nondeterministic fta via bisimulation. The concepts of bisimulation and bisimilarity were originally introduced by Milner in [Mil82] as tools for investigating labelled transition systems, i.e. directed (possibly infinite) graphs with labelled edges. A bisimulation between a pair of transition systems with state spaces \( S \) and \( S' \) is a relation \( \cong \subseteq S \times S' \) with the following distinguishing property. If \( p \cong p' \), \( q \cong q' \), and there is an edge labelled \( a \) from \( p \) to \( q \) in the first system, then there is an edge labelled \( a \) from \( p' \) to \( q' \) in the second system, and vice versa. In other words, the two systems can simulate each other and are thus indistinguishable to an outside observer. How bisimulation can be used in minimisation becomes apparent when we consider the consequences of a pair of states in the same system being bisimilar with each other. Now bisimilarity implies redundancy, since it is not necessary to keep two states that perform exactly the same function. To reduce the size of an automaton we can thus initiate a search for the coarsest bisimulation relation on its state space, and then collapse each equivalence class in the resulting partition into a single state.

Paper IV. Bisimulation minimisation of nondeterministic tree automata

Paper IV describes a first attempt at a bisimulation-based minimisation algorithm for nondeterministic fta. This work was originally motivated by the need for a fast, but not necessarily optimal, minimisation algorithm to be applied in the field of regular model checking. To efficiently decide the coarsest bisimulation relation on the input automaton \( M \), we employ a partition refinement algorithm by Paige and Tarjan [PT87]. The resulting minimisation algorithm has a time complexity of \( O(r^2m \log n) \), where \( r \) is the maximal rank of any symbol in the input alphabet \( \Sigma \), and \( n \) and \( m \) are the number of states and the number of entries in the (total) transition table of \( M \), respectively. From now on we refer to the definition of bisimulation in Paper IV as AKH bisimulation.

Paper V. Backward and forward bisimulation minimisation of tree automata

The definition of bisimilarity that is used in Paper IV has an important disadvantage: for a pair of states \( p \) and \( q \) to be bisimilar, the set of trees that is mapped by the transition function to \( p \), and the set of trees that is mapped to \( q \), must coincide. This makes the minimisation algorithm inadequate for dfta,
as any deterministic transition function will map each input tree to at most one state\(^1\). As we shall see, this problem is addressed and solved in Paper V. Paper V generalises two alternative definitions of bisimulation to the domain of fta [Buc07]. Though both types compare favourably with that of Paper IV, they do so for different reasons. The first of these, backward bisimulation, can be obtained by relaxing the requirements of Paper IV, so if a pair of states are AKH bisimilar, then they are automatically backwards bisimilar. It follows that although backward bisimilarity is just as efficient to compute as AKH bisimilarity, minimisation based on backward bisimulation yields smaller output automata. Unfortunately, backward bisimulation shares with AKH bisimulation the weakness that it has no effect on deterministic devices. The second variation, called forward bisimulation, coincides with the classical minimisation algorithm when the input fta is deterministic, and also has the added bonus of being somewhat easier to derive than the backward bisimulation. This does not mean that forward bisimulation replaces backwards bisimulation: for some fta, backward bisimulation yields a smaller output automaton, and for others, forward bisimulation. Backward and forward bisimulation allow for minimisation algorithms that execute in time \(O(r^2m \log n)\) and \(O(rmn \log n)\), respectively, where \(r\) and \(n\) are as before, and \(m\) is the number of entries in the partial transition table of the input fta. As both algorithms are quite efficient, one could in practice apply both algorithms to an fta in an iterative manner.

Paper VI. Bisimulation minimisation for weighted tree automata

In Paper VI, forward and backward bisimulation are defined for weighted tree automata, together with the bisimulation minimisation algorithms that they induce. For arbitrary semirings, these algorithms both execute in time \(O(rmn)\), where \(r\) and \(n\) are as in Papers IV and V, and \(m\) is the number of entries in the tree representation of the input wta. The increased complexity is due to the fact that the partition refinement algorithm used in Papers IV and V was originally designed for the Boolean semiring, and does not generalise. However, when the underlying semiring is cancellative (that is, when \(a + b = a + c\) implies \(b = c\)) we can use a partition refinement strategy by Hopcroft that lets us retain the complexity results of Paper V.

3.3 Part III. Tree-based generation of music

As indicated in Section 2.6, every tree in \(T_\Sigma\) is a term in every \(\Sigma\)-algebra \( \mathcal{A} \), and can, for this reason, be evaluated to an element in the carrier of \( \mathcal{A} \). On the other hand, if the set \(A\) is generated by a family of operations \((f_a)_{a \in \Sigma}\), then \((A, (f_a)_{a \in \Sigma})\) is obviously a \(\Sigma\)-algebra \(\mathcal{A}\) such that \(A\) is given by

\(^1\)This problem was first pointed out by Andreas Maletti at Technische Universität Dresden.
\{\text{val}_A(t) \mid t \in T_\Sigma\}. Moreover, subsets of \(A\) can be represented as a combination of a tree generating device and a suitable \(\Sigma\)-algebra. This tree-based approach was already present in Chapter 2 although it was not made explicit: in Section 2, trees are evaluated to elements in a finite state space, in Section 2.5 to tuples over the carrier set of some semiring, and in Section 2.6 to the domain of trees itself. Tree-based generation and transformation has in addition also been successfully applied to graphs [Eng97, Dre00], pictures [Dre06], and architecture [MvT04]. To extend this list further, we devote Part III to the algorithmic composition of (classical Western) music by means of formal devices from tree language theory. Already at an intuitive level, this domain seems particularly suited for the tree based approach, as more or less anything that we would recognise as music possesses a great degree of both repetition and structure: a song may for example consist of a set of voices, each voice spanning a number of phrases, each of which can in turn be broken down into smaller objects; measures, themes, motifs, and, at the lowest level, individual notes. In fact, one of the challenges faced in Paper VII and Paper VIII is how to describe the structure of a piece in the most natural way, when there are so many choices available.

Paper VII. Wind in the willows

Paper VII presents a tree-based algorithm for automatic composition implemented in the software system TREEBAG [Dre98]. The algorithm, which is called Willow in the paper, produces short musical pieces of the form \(AA'BA''\), where \(A\) and \(B\) are phrases, and \(A'\) and \(A''\) are variations on \(A\). A piece is generated as follows. First a (tree representing a) basic rhythmical structure is derived in a regular tree grammar. This is then passed through a sequence of top-down tree transducers. Each transducer modifies the tree so as to model some specific musical attribute, for example scale, tempo, or ornamentation. Eventually an algebra interprets the output of the last transducer in the sequence, thus realising the final piece.

Paper VIII. An algebra for tree-based music generation

Willow was constructed with a particular type of music in mind, and this is reflected in the algebra: its operations are tailor-made for a particular generation task and this makes it inapt at describing other types of music. The compilation of a more general algebra is addressed in Paper VIII. In contrast to the preceding paper, we now begin by defining the algebra and then explore how well it lends itself to the description of various musical constructs. The carrier set that we settle on spans three domains; integers, reals, and musical pieces. A musical piece is now formalised to be a finite set of notes arranged in time, each note being defined by a tone and a duration. The operations that can be performed on musical pieces include for example union, overlay, transposition, and scaling.
References


