Technische Universität München
Institut für Informatik

On Normalization and Type Checking for Tree Transducers

Dipl.-Inform. Sylvia Friese

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Alfons Kemper, Ph.D.
Prüfer der Dissertation:	1. Univ.-Prof. Dr. Helmut Seidl
	2. Univ.-Prof. Dr. Klaus U. Schulz, Ludwig-Maximilians-Universität München

Die Dissertation wurde am 24.03.2011 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 07.07.2011 angenommen.

Abstract

Tree transducers are an expressive formalism for reasoning about tree structured data. Practical applications range from XSLT-like document transformations to translations of natural languages. Important problems for transducers are to decide whether two transducers are equivalent, to construct normal forms, give semantic characterizations, and type checking, i.e., to check whether the produced outputs satisfy given structural constraints. This thesis addresses these problems for important classes of tree transducers. Constructive solutions are provided and classes of transducers for which these algorithms run in polynomial time, are identified.

Equivalence testing, normalization, and semantic characterization are often solved together by the use of a Myhill-Nerode theorem. This identifies necessary and sufficient semantic properties for transformations definable by a specific class of transducers. The theorem also implies that a unique normal form of those transducers exists. Moreover, it implies that, given a transducer, the normal transducer can be constructed. This immediately leads to the question: Are there classes of tree transducers for which a Myhill-Nerode theorem exists? We give an affirmative answer for the class of deterministic bottom-up tree transducers. A semantic characterization of transformations definable by these transducers is presented, and, moreover, it is evidenced that for every deterministic bottom-up tree transducer, a unique equivalent transducer can be constructed, which is minimal. The construction is based on a sequence of normalizing transformations, which, among others, guarantee that non-trivial output is produced as early as possible. For a deterministic bottom-up transducer where every state produces either none or infinitely many outputs, the minimal transducer can be constructed in polynomial time.

One of the useful properties of tree walking transducers is decidability of type checking: Given a transducer and input and output types, it can be checked statically whether each document adhering to the input type is necessarily transformed by the transducer into documents adhering to the output type. Here, a "type" means a regular set of trees specified by a finite-state tree automaton. Usually, type checking of tree transducers is extremely expensive; already for simple top-down tree transducers it is known to be EXPTIME-complete. Are there expressive classes of tree transducers for which type checking can be performed in polynomial time? Most of the previous approaches are based on inverse type inference. In contrast, the approach presented here uses forward

type inference. This means to infer, given a transducer and an input type, the corresponding set of output trees. In general, this set is not a type, i.e., it is not regular. However, it can be decided if its intersection with a given type is empty. Using this approach, we show that type checking can be performed in polynomial time if (1) the output type is specified by a deterministic tree automaton, and (2) the tree walking transducer visits every input node only a bounded number of times. If the transducer is additionally equipped with accumulating call-by-value parameters, then the complexity of type checking also depends (exponentially) on the number of such parameters. For this case, a fast approximative type checking algorithm is presented, based on context-free tree grammars. Finally, the approach is generalized from trees to forest walking transducers, which additionally support concatenation as a built-in output operation.

Zusammenfassung

Baumübersetzer sind ein ausdrucksstarker Formalismus, um baumstrukturierte Daten zu analysieren. Praktische Anwendungen reichen von XSLT-artigen Dokumentumstrukturierungen zu Übersetzungen natürlicher Sprache. Bedeutende Problemstellungen für Übersetzer sind, zu entscheiden, ob zwei Übersetzer äquivalent sind, eine Normalform konstruiert werden kann, es eine semantische Charakterisierung gibt und Typüberprüfung, d.h., zu überprüfen, ob die erzeugten Ausgaben gegebene strukturelle Bedingungen erfüllen. Diese Dissertation befasst sich mit diesen Fragen für wesentliche Baumübersetzerklassen. Wir stellen konstruktive Lösungen bereit und identifizieren Übersetzerklassen, für die diese Algorithmen nur polynomielle Zeit benötigen.

Äquivalenztest, Normalisierung und semantische Charakterisierung sind oft mittels eines Myhill-Nerode Theorems gemeinsam lösbar. Es zeigt notwendige und hinreichende semantische Eigenschaften für Übersetzungen einer bestimmten Klasse von Übersetzern auf und impliziert eine eindeutige Normalform für diese Übersetzer. Zusätzlich beinhaltet das Theorem, dass der entsprechende Übersetzer in Normalform aus einem beliebigen Übersetzer konstruiert werden kann. Es stellt sich also die Frage, ob es Klassen von Baumübersetzern gibt, für die ein Myhill-Nerode Theorem existiert. Für deterministische Aufwärts-Baumübersetzer (deterministic bottom-up tree transducers) geben wir eine positive Antwort. Wir präsentieren eine semantische Charakterisierung für Übersetzungen, die durch solche Übersetzer beschrieben werden können. Desweiteren zeigen wir, dass für jeden deterministischen Aufwärts-Baumübersetzer ein eindeutiger äquivalenter Übersetzer konstruiert werden kann, der minimal ist. Diese Konstruktion basiert auf einer Folge von Normalisierungen, welche unter anderem garantieren, dass nicht-triviale Ausgaben so früh wie möglich erzeugt werden. Wenn jeder Zustand eines deterministischen Aufwärts-Baumübersetzers entweder keine oder unendlich viele Ausgaben produziert, kann der minimale Übersetzer in polynomieller Zeit konstruiert werden.

Eine der nützlichen Eigenschaften von Zwei-Wege-Baumübersetzern (tree walking transducers) ist, dass das Problem der Typüberprüfung entscheidbar ist: Wenn ein Übersetzer und ein Eingabe- sowie ein Ausgabetyp vorgegeben sind, kann statisch geprüft werden, ob der Übersetzer Dokumente, die dem Eingabetyp entsprechen, grundsätzlich in Dokumente des Ausgabetyps übersetzt. Dabei verstehen wir unter “Typ” eine reguläre Baummenge, die durch einen endlichen Automaten spezifiziert ist. Normalerweise ist Typüberprüfung von

Baumübersetzern extrem aufwändig. Es ist bekannt, dass dieses Problem bereits für einfache Abwärts-Baumübersetzer (top-down tree transducers) EXPTIME-vollständig ist. Es stellt sich die Frage, ob es ausdrucksstarke Baumübersetzerklassen gibt, für die Typüberprüfung in polynomieller Zeit durchgeführt werden kann. Die meisten bisherigen Ansätze basierten auf inverser Typinferenz. Wir benutzen hier den entgegengesetzen Ansatz mittels vorwärtsgerichteter Typinferenz. Das heißt, für einen gegebenen Übersetzer und einen Eingabetyp leiten wir die Menge aller zugehörigen Ausgabebäume her. Im Allgemeinen ist diese Menge kein Typ, d.h., es ist keine reguläre Menge. Trotzdem ist es entscheidbar, ob die Schnittmenge mit einem gegebenen Ausgabetyp leer ist. Mit diesem Ansatz zeigen wir, dass man Typüberprüfung in polynomieller Zeit durchführen kann, wenn (1) der Ausgabetyp mittels eines deterministischen Automaten gegeben ist und (2) der Zwei-Wege-Baumübersetzer jeden Knoten eines Eingabebaums höchstens begrenzt oft besucht. Wenn der Baumübersetzer zusätzlich mit Wertparametern (call-by-value) ausgestattet ist, hängt die Komplexität ausserdem (exponentiell) von der Anzahl der Parameter ab. In diesem Fall wird ein schneller approximativer Typüberprüfungsalgorithmus präsentiert, der auf kontextfreien Baumgrammatiken basiert. Zum Schluß wird dieser Ansatz von Bäumen auf Zwei-Wege-*Wald*übersetzer verallgemeinert. Diese Übersetzer erlauben zusätzlich Konkatenation als Operation auf Ausgabebäumen.

Contents

Chapter 1

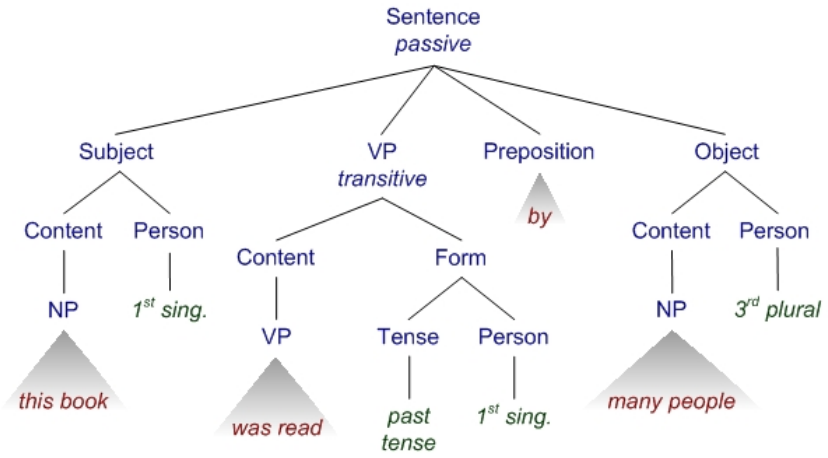
Introduction

Tree transducers are a formal model to describe transformations of tree structured data. Such transformations occur in various fields. Early formal approaches were inspired by compiler development. Internal representations (of the syntax or the semantics) of programs are trees. Separating the syntax of a program language from assigning semantics to the source language led to the analysis of the model of syntax-directed translations [Iro61]. More generally, program analysis, like evaluations and optimizations, are tree transformations. Further works also based on the theory of tree languages, finite tree automata, and the associated algebraic formulation (e.g., [Tha69, Rou70, Tha70, CDG+07]). The formal models of tree transducers are more abstract. Input trees are not only syntax trees of program languages, but trees over a ranked alphabet. Already in the year 1969, Thatcher stated that the abstract models of tree transducers are not only generalizations of transformations of program languages, but also of translations of natural languages:

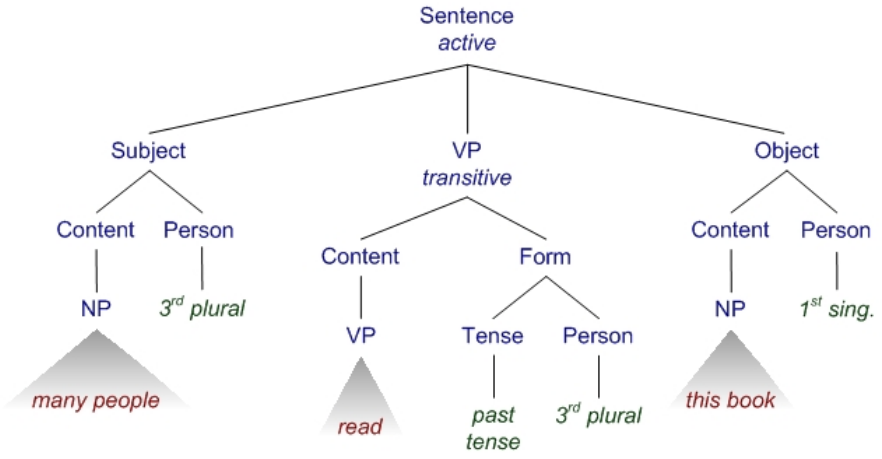
> "It appears to be an area with considerable promise of application to questions of syntax and semantics of programming languages and to the analysis of natural languages." [Tha69]

In the beginning of this century, tree transducers were rediscovered in the context of natural language processing [KG05]. In addition to the string- and phrase-based models, tree-based transformation models are considered in many areas, e.g., in machine translation [Wu97, ADB00, Eis03], i.e., automatic translation from one natural language (like Chinese) into another (like English). Further tasks are, for instance, automatically answering a human-language question [HG01, EM03a], generating natural language from information stored in computer databases [BR00], and automatic summarization [MM99, KM02], i.e., to automatically abstract the relevant information of (multiple) documents.

Due to the inherent syntax and semantics of natural language, sentences will be represented by syntax trees, for instance like the two in Figure 1.1. A transformation model may, for instance, translate passive voice into active



(a) Passive voice: "This book was read by many people."



(b) Active voice: "Many people read this book."

Figure 1.1: Syntax trees of the sentence "This book was read by many people." and its active counterpart "Many people read this book.".

voice, transforming the syntax tree in Figure 1.1(a) into the syntax tree in Figure 1.1(b). The worldwide process of globalization increases the demand for natural language processing tools constantly. An international software product has to be adapted to different markets, languages, and cultures. The World Wide Web (WWW) provides a lot of information. Asking for specific information in human language, causes that a computer has to transform the question to understand it. Furthermore, the computer has to extract the correct information from a huge amount of data, which may be given in different languages. Finally, the computer has to summarize these informations and transform the correct answer into human language [FBCC+10].

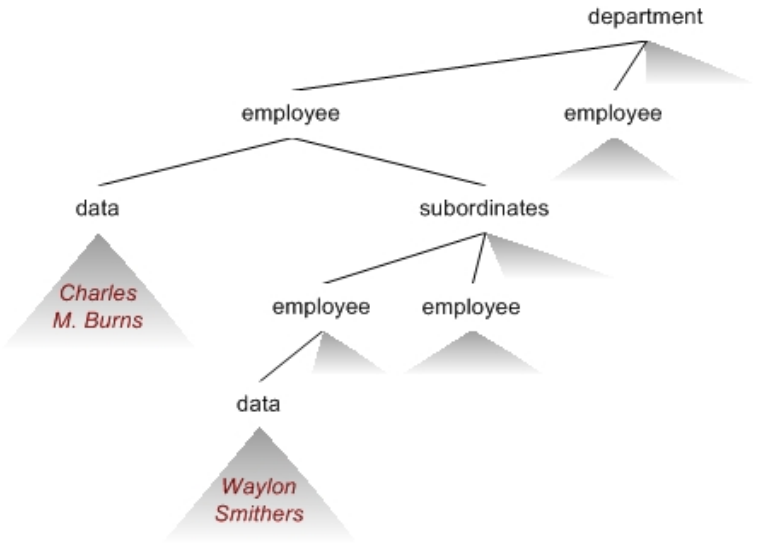
Lots of work on tree transformations was initiated also by the exchange of data via the WWW. In the end of the 1990s, the extensible markup language XML became the current standard data exchange format for the WWW [ABS00, BPS98]. It allows to represent data in a sequential form preserving the underlying structure. Associated parts are wrapped by matching pairs of open and close tags. For instance, a company structure can be presented by an XML-document like the one in Figure 1.2(a). Such an XML document can be seen as a sequential representation of a tree. The tree, which is described by the given example document is shown in Figure 1.2(b). XML is a meta language. It does not dictate, which tags are allowed and how to use them. Services, which require the data in a specific structure, have to declare the type of *valid* documents. Such types can be defined using an XML type definition language like DTD (Document Type Definition, [BPS08]), XML Schema [FW04], or RELAX NG (Regular Language Description for XML New Generation, [CM01]).

Today, data with an inherent structure is used, exchanged, and restructured everywhere. An internet shop has to present its range of products dependent on the requests of different customers. Emails need an exchange format, which is platform-independent. A company uses data in different services, which need the data in different representations. Booking a flight via an online travel agency will cause a request to web services of different airlines. The airlines, in turn, will respond a list of suitable flights with detailed information about time, prices, and more. The travel agency then has to transform the structure of the data to present it on its webpage.

As a consequence, there is a widespread use of transformation and query languages for XML, like the current W3C recommendations XSLT (Extensible Stylesheet Language Transformations, [Kay03]) and XQuery [BCF+03]. XSLT is a complex language. A system, which automatically infers an XSLT program from a given set of examples would lighten the work of web programmers. Moreover, it is necessary to check if an XSLT stylesheet produces the desired output for documents of a given input type. On the one hand, it is possible that the transformation is erroneous, especially if it is generated automatically, on the other hand, one may check if the stylesheet also produces valid output for a new input type or a more restricting output type. Assume that the travel agency gets flight informations from a new airline in a new type. The type differs slightly from types of other airlines. The agency will check their existing transformations before developing a new one.

```
<department>
  <employee>
    <data>
      <name> Charles Montgomery Burns </name>
      <id> ... </id> ...
    </data>
    <subordinates>
      <employee>
        <data><name> Waylon Smithers </name> ... </data> ...
      </employee>
      <employee> ... </employee> ...
    </subordinates>
  </employee>
  <employee> ... </employee> ...
</department>
```

(a) XML-document for a company structure.



(b) The company structure seen as a tree.

Figure 1.2: Company structure as XML-document and as a tree.

These two important examples show that data in many areas can be described using tree structures. Consequently, there is a high demand to understand the various transformations between different structures. To analyze transformations of tree structured data, we consider the formal model of tree transducers.

1.1 Tree Transducers

The formal model of tree transducers enhances finite-state word automata in two dimensions. First, they work on trees rather than on strings. Secondly, tree transducers produce output. Here, we consider tree-to-tree transducers over ranked trees, i.e., input and output are trees with finite signatures. The most simple models of tree transducers are top-down and bottom-up tree transducers, cf. Chapter 7 of the textbook [CDG+07]. Input trees are transformed starting at the root (top-down) or at the leaves (bottom-up). They were introduced in the 1970s by Rounds and Thatcher [Rou70, Tha70], and Thatcher [Tha73], respectively. Their expressive powers are incomparable, both for nondeterministic and deterministic transducers [Eng75]. Deterministic and nondeterministic top-down tree transducers (DTTTs and TTTs, respectively) are able to produce different outputs for the same occurrence of an input subtree, i.e., a (D)TTT copies an input tree and afterwards processes the copies differently. This is not possible by deterministic or nondeterministic bottom-up tree transducers (DBTTs and BTTs, respectively). However, both DBTTs and BTTs can decide whether to generate any output for an input subtree or not after processing the subtree. Moreover, in the nondeterministic case, BTTs can produce copies of the same output tree after nondeterministic processing of the input tree, in contrast to DTTTs, TTTs, and DBTTs. Consider, for instance, the example transformation depicted in Figure 1.1: Transforming sentences in passive voice into sentences in active voice. For an input tree like the one in Figure 1.1(a), a (deterministic) bottom-up tree transducer produces an output tree (Figure 1.1(b)) only if the subtree "`Preposition`(... by ...)" is proper. Thereto, it processes this subtree without producing any output for it. This is neither possible by deterministic nor by nondeterministic top-down tree transducers. On the other hand, there are transformations, which are not definable by bottom-up tree transducers, but by top-down tree transducers (see [Eng75] for examples). Especially, if the transformation produces different outputs for the same subtree.

An enhanced model of tree transducers is the model of tree walking transducers (or two-way tree transducers, 2TTs) [Eng09]. 2TTs are essentially the same as the k-pebble tree transducers of [MSV00] for the case $k = 0$. In contrast to the top-down and bottom-up models, the processing of the input tree by a 2TT is not restricted to one direction. After transforming a node, the transducer can stay at this node, proceed with its father, or one of its children. Thus, in contrast to top-down and bottom-up tree transducers, also in the deterministic case, a tree walking transducer does not have to terminate for

every input tree. A 2TT is similar to an attribute grammar, which operates on derivation trees, and has trees as semantic domain (with tree top-concatenation as only semantic operation). A further generalization is the model of macro tree walking transducers (2MTTs). It can be seen as the k-pebble macro tree transducer of [EM03b], for the case that $k = 0$. The 2MTT generalizes the 2TT by adding formal context-parameters to the attributes, i.e., each attribute is seen as a function, which can take parameters of type output tree. The addition of parameters extends the expressive power of tree walking transducers [EM03b]. Tree transducers work on ranked trees, but for practical applications such as XSLT-transformations we have often to deal with unranked forests. A formal model is the model of macro forest walking transducers (2MFTs), which is an extension of 2MTTs. The 2MFT work on forests instead of ranked trees. For instance, such a transducer can handle department structures like the one in Figure 1.2(b), with arbitrary numbers of employees. Macro forest walking transducers are able to concatenate output forests as an additional operation. Such transducers are very expressive and can simulate most features of transformation languages such as XSLT.

This thesis answers important computational problems for deterministic bottom-up tree transducers and the different models of walking transducers.

1.2 Questions

There are a lot of open questions in the context of tree transducers. The most important computational problems are:

1. Type checking: Does a given transducer produce the desired output trees for proper input trees?
2. Equivalence testing: Do two transducers realize the same transformation?
3. Normalization: Can a unique transducer in normal form for a given transformation be constructed?
4. Learning by example: How could a transducer for a transformation be generated of which we only know examples?

The three latter problems are closely related because a common solution may often be constructed with the use of a Myhill-Nerode theorem. The original Myhill-Nerode theorem was stated in 1957/1958 [Myh57, Ner58] for regular word languages and deterministic finite-state automata. It gives a semantic characterization of word languages definable by finite-state automata and provides a unique (up to isomorphism) minimal deterministic finite string automaton for a regular language. Such a semantic characterization and a unique representation of a language or (in the context of transducers) a transformation often give rise to a learning algorithm [Gol78, OGV93, LMN10]. For sequential string-to-string transducers, a Myhill-Nerode style theorem and a learning algorithm exists [Cho03, OGV93]. It is based on the minimal earliest transducers, which

were introduced for strings by Mohri [Moh00]. Recently, it was shown that also for deterministic top-down tree-to-tree transducers, there is a Myhill-Nerode theorem [LMN10]. The proof relies on a new canonical normal form for such transducers, called the "earliest" normal form (inspired by the earliest string transducers of Mohri [Moh00]). Beside this, similar results are only known for string transducers and simple relabeling tree transducers.

The question arises whether there is a Myhill-Nerode theorem for deterministic bottom-up tree transducers.

1.3 Results

In the first part of this thesis, we show that a Myhill-Nerode theorem for deterministic bottom-up tree transducers exists. Even though the idea is similar, obtaining the normal form and a semantic characterization is quite different for DBTTs than for top-down transducers. First, we present a semantic characterization of transformations definable by DBTTs. We identify one important semantic property of these transformations: They are *path-finite*, which means that every path in input trees evokes only finitely many different paths in output trees. If, in addition, the congruence relation of the transformation has finite index, we show that the transformation is definable by a deterministic bottom-up tree transducer. In particular, the Myhill-Nerode theorem leads to a unique minimal DBTT. Given an arbitrary deterministic bottom-up tree transducer, we show that the unique DBTT can be constructed. Generating the "earliest" normal form for a given deterministic bottom-up tree transducer proceeds in four steps: (1) First, we make the transducer "proper", i.e., we remove all output from states which only produce finitely many different outputs. The output for such states is postponed until the root node of the input tree. This is similar to the "proper normal form" of [AU71, EM03c] (which removes states that produce finitely many outputs, using regular look-ahead). (2) We make the transducer "earliest", i.e., every state produces output as early as possible during translation. (3) We remove pairwise "ground context unifiers" (this is a technical property, achievable in quadratic time on the transducer). (4) Last, we minimize in the usual way (by merging states that are isomorphic). Steps (2)–(4) can be done in polynomial time, i.e., given a proper transducer, its unique earliest transducer is constructed in polynomial time. Hence, equivalence checking for proper transducers can be done in polynomial time. Constructing a proper transducer (Step 1) takes double exponential time in the worst case. The Myhill-Nerode style theorem can be used to build a Gold style learning algorithm [Gol78], as done for deterministic top-down tree transducers in [LMN10]. Furthermore, the normal form can be used to decide certain (semantic) subclasses of DBTTs; e.g., we can decide whether a given DBTT is equivalent to a relabeling, using the normal form. This provides an alternative proof of [Gaz06], for the deterministic case.

For the first problem, type checking a given transducer w.r.t. given input and output types, we are interested in exact solutions (in contrast to the approxi-

mative approach). The class of transformations for which exact type checking is possible, is surprisingly large [EM03b, MSV03, MBPS05]; the price to be paid for exactness is also extremely high. Even for simple top-down transformations, exact type checking is exponential-time complete [MN05], and for more complex transformations such as the k-pebble transducers of [MSV03], the problem is non-elementary, i.e., cannot be expressed by an iterated exponential function. For practical considerations, however, one is interested in useful subclasses of transformations for which exact type checking is provably tractable. In the second part of this dissertation, we investigate type checking of transformations formulated through tree walking transducers, macro tree walking transducers and macro forest walking transducers. Given suitable descriptions (types) of admissible inputs and outputs for a tree walking transducer, type checking the transducer means to test whether all outputs produced by the 2TT on admissible inputs are again admissible.

Our main result in Part II is: If valid output trees are described by deterministic tree automata, then exact type checking can be done in polynomial time for a large class of practically interesting transformations obtained by putting only mild restrictions on these kinds of transducers. Our approach is forward type inference, i.e., we infer the set of output trees produced by the transducer for valid input trees and compare this set with the given output type. More precisely, we determine if the intersection of the inferred outputs with the complement of the given output type is empty. The presented algorithms solve the exact type checking problem for arbitrary 2TTs, 2MTTs, and 2MFTs, with respect to regular tree (or forest) languages as types. In general, they run in exponential time, but we present subclasses for which they run in polynomial time. We show that if a 2TT visits the same node only constantly often, i.e., is *strictly b-bounded*, then it can be type checked in polynomial time. For the more complex classes of macro tree walking and macro forest walking transducers, we also present approximative algorithms.

1.4 Related Work

The Myhill-Nerode theorem for bottom-up tree automata is the straightforward generalization of the corresponding results for finite string automata. The minimal automaton and the congruence relation appear in [Bra68, Bra69, AG68, EW67], see also Chapter 1.5 in [CDG$^+$07]. Special transformations can be defined by tree automata, like relabelings and recognizable tree relations [DT90]. For finite state transducers, a Myhill-Nerode style theorem exists for subsequential string transducers [Cho03, OGV93], which is based on the minimal earliest transducers for strings by Mohri [Moh00]. For tree-to-tree transducers, in [LMN10] a Myhill-Nerode theorem for top-down tree transducers was the basis for a Gold style learning algorithm [Gol78]. In 1980 it was shown that equivalence for deterministic transducers is decidable both in the top-down [Ési80] and bottom-up case [Zac80]. Later, a polynomial-time algorithm for single-valued bottom-up transducers was provided [Sei92].

A valid generalization of both deterministic bottom-up and deterministic top-down tree transducers is the deterministic top-down tree transducer with regular look-ahead [Eng77]. Even though the equivalence problem for DTTTs with regular look-ahead is easily reduced to the one for DTTTs [EMS09], it is an intriguing open problem whether DTTTs with regular look-ahead have a canonical normal form. Another related model of transformation is the attribute grammar [Knu68], seen as a tree transducer [Fül81, FV98]. For attributed tree transducers, decidability of equivalence is still an open problem, but for the special subclass of "nonnested, separated" attribute grammars (those, which can be evaluated in one strict top-down phase followed by one strict bottom-up phase), equivalence is known to be decidable [CFZ82]. This class strictly includes DTTTs (but not DBTTs [FV95]). There are several other interesting incomparable classes of tree translations for which equivalence is known to be decidable, but no normal form (and no complexity) is known, e.g., MSO-definable tree translations [EM06]. This class coincides with single-use restricted attribute grammars or macro tree transducers with look-ahead [EM99]. Is there a canonical normal form for such transducers?

Another interesting generalization are tree-to-*string* transducers. It is a long standing open problem [Eng80] whether or not deterministic top-down tree-to-string transducers (DTTSTs) have decidable equivalence. Recently, for the subcase of *linear sequential* DTTSTs, a unique normal form similar to the earliest normal form was proved [LLN$^+$11]. Another recent result states that functional visibly pushdown transducers have decidable equivalence [FRR$^+$10]. This class is closely related to non-copying DTTSTs. It raises the question whether our Myhill-Nerode theorem for DBTTs can be extended to functional (but nondeterministic) bottom-up tree transducers.

In its most general setting, the type checking problem for XML transformations is undecidable. Hence, general solutions are bound to be approximative, but seem to work well for practical XSLT transformations [MOS07]. Approximative type checking for XML transformations is typically based on (subclasses of) recognizable tree languages. Using XPath [CD99] as pattern language, XQuery [BCF$^+$03] is a functional language for querying XML documents. It is strongly-typed and type checking is performed via type inference rules computing approximative types for each expression. Approximative type inference is also used in XDuce [HP03] and its follow-up version CDuce [Fri04]; navigation and deconstruction are based on an extension of the pattern matching mechanism of functional languages with regular expression constructs. Recently, Hosoya et al. proposed a type checking system based on the approximative type inference of [HP02] for parametric polymorphism for XML [HFC05]. Type variables are interpreted as markings indicating the parameterized subparts. In [MOS07], a sound type checking algorithm is proposed (originally developed for the Java-based language Xact [KMIS04]) based on an XSLT flow analysis that determines the possible outcomes of pattern matching operations; for the benefit of better performance, the algorithm deals with regular approximations of possible outputs.

Milo et al. [MSV03] propose the k-pebble tree transducer (k-PTT) as a for-

mal model for XML transformations, and show that exact type checking can be done for k-PTTs using *inverse* type inference. The latter means to start with an output type O of a transformation τ and then to construct the type of the inputs by backwards translating O through τ. Each k-pebble tree transducer can be simulated by compositions of $k+1$ stay macro tree transducers (stay-MTTs) [EM03b], thus, type checking can be solved in time (iterated) exponential in the number of used pebbles. Intuitively, k-pebble tree transducers for $k = 0$ correspond to our 2TTs. In [Eng09], it was shown that inverse type inference for tree walking transducers can be done in exponential time, and can be done for k-fold compositions of 2TTs in k-fold exponential time. Maneth et al. show in [MBPS05] that inverse type inference can be done for a transformation language providing all standard features of most XML transformation languages using a simulation by at most three stay-MTTs. Inverse type inference is used in [MN04, MN05] to identify subclasses of top-down XML transformations, which have tractable exact type checking. We note that the classes considered there are incomparable to the ones considered in this thesis.

Tree transducers are a promising model to describe translations on natural languages, but the requirements to transformation models used in this context are complex [KG05]. Because of the ambiguity of natural language, most approaches in machine translation are on statistical models [MS99]. The overview [KG05] of (probabilistic) tree-based models in natural language processing figures out important requirements, like closure properties, on a reasonable formal syntax-based translation model (see also [Mal11]). Until now there is no model known which fulfills all of these properties. Most of the transducers, which are considered as translation models, are extensions of the basic models of bottom-up and top-down tree transducers (as required by Knight et al. in [KG05]). Thus, it is important to understand the basic models of tree transducers, even though the results are not directly applicable to machine translation models. Promising models to handle the probabilities are weighted transducers, i.e., rules are equipped with a weight (see [FV09] for a survey). In [Mal11], weighted (linear and nondeleting) extended top-down tree transducers are analyzed in the context of machine translation where extended means that the patterns on left-hand sides of rules are not restricted to one input symbol. They fulfill most of the requirements [Mal11], but are not closed under composition [AD82]. The different subclasses of transformations induced by extended top-down tree transducers are studied in [MGHK09]. The model of linear extended multi bottom-up tree transducers fulfills most of the requirements stated in [KG05], but it is an open problem if these transducers are efficiently trainable [ELM09, ELM08]. Instead of learning a transducer, the problem in this context is to train probabilistic transducers (given by sets of rules) on huge amounts of collected training data in the form of tree pairs. In [GKM08], training of probabilistic tree-to-tree and tree-to-string transducers is discussed. Especially, for the model of weighted extended top-down tree transducers a training algorithm is provided. In [Eis03] a training algorithm for synchronous tree substitution grammars is given where the model is trained on all possible derivations in proportion to their probabilities.

1.5 Own Publications

This thesis is based on the three original publications [FSM10, FSM11, MFS11].

- The paper [FSM10] was presented in 2010 at the 14th International Conference on Developments in Language Theory (DLT) in London, Ontario, Canada. It covers the minimization of deterministic bottom-up tree transducers detailed in Section 6.1 of Part I of this work.

- A longer version of the afore mentioned paper will be published in the International Journal of Foundations of Computer Science (IJFCS) [FSM11]. It shows some extended proofs also contained in this thesis.

- Part II is based on the chapter [MFS11] of the book "Modern Applications of Automata Theory" [DS11]. It is estimated to appear as Volume 2 of the IISc Research Monographs Series in September 2011.

These three publications are co-authored by Prof. Dr. Sebastian Maneth and my supervisor Prof. Dr. Helmut Seidl.

Chapter 2

Preliminaries

In this chapter, we designate some elementary notations of sets, words, and functions and define trees, forests, and other basic terms.

In the following $\mathbb{N}$ denotes the set of all positive natural numbers and by $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ we refer to the set of natural numbers including 0. For a constant $k \in \mathbb{N}_0$ we abbreviate by $[k]$ the set $\{1, \ldots, k\} \subset \mathbb{N}$ where $[0] = \emptyset$.

For a set X, we denote by X^* the set of *words* or *strings* (including the empty word ϵ) over X. The concatenation of two words u and v over X is denoted by $u.v$ or uv. If there are words u, v, and w such that $u.v = w$, we say that u is a *prefix* and v a *suffix* of w. A subset $W \subseteq X^*$ of words is called *prefix-closed* if for all $u, v \in X^*$ holds: $u.v \in W$ implies $u \in W$. Note that ϵ is element of every non-empty, prefix-closed set.

For two sets X and Y and a subset $D \subseteq X$, a function $\varphi : D \to Y$ is a partial function from X to Y, written $\varphi : X \dashrightarrow Y$ where φ is undefined for every $x \in X \setminus D$. Then D is the domain of φ. We also use $\mathrm{dom}(\varphi)$ to refer to the domain. The image of a subset $S \subseteq X$ under φ is the set of all images of elements in S for which φ is defined, i.e., $\varphi(S) = \{\varphi(x) \mid x \in D \cap S\}$.

2.1 Trees and Forests

Trees and forests are build over alphabets (i.e., finite sets) of symbols or node labels. An *unranked tree* over an alphabet Σ consists of a root node labeled by a symbol $\mathtt{a}$ from Σ and a forest f, written $\mathtt{a}\langle f \rangle$. A *forest* (or *hedge*) is a sequence of an arbitrary number of unranked trees, written $u_1 u_2 \ldots u_m$. The number m is called the *length* of the forest. The empty forest, i.e., a forest with length $m = 0$, is denoted by ϵ.

Definition 2.1 (Forests)**.** Let Σ be an alphabet. The set $\mathcal{F}_\Sigma$ of forests f over Σ is defined by the grammar rules

$$f \ ::= \ \epsilon \mid uf \qquad \text{and} \qquad u \ ::= \ \mathtt{a}\langle f \rangle$$

where $\mathtt{a} \in \Sigma$.

The tree in Figure 1.2(b) is such a forest (with only one non-empty tree) of the form $\texttt{department}\langle f \rangle\, \epsilon$ where f is a forest of "`employee`-trees". The length of f depends on the company structure and varies from company to company.

Rather than on forests, *tree* transducers work on *ranked* trees. There, we assume that a fixed rank is given for every element of Σ, i.e., $\Sigma = \biguplus_{m \in \mathbb{N}} \Sigma^{(m)}$ where $\Sigma^{(m)}$ is the set of all symbols with rank m and $\biguplus$ denotes the union of distinct sets. Then, Σ is said to be a *ranked alphabet*. We define $\mathit{rank}(\mathtt{a}) = m$ for all symbols $\mathtt{a} \in \Sigma^{(m)}$ for $m \geq 0$. We also write $\mathtt{a}^{(m)}$ to denote a symbol $\mathtt{a}$ of rank m. The maximal rank $mr(\Sigma)$ is the smallest number m such that $\Sigma^{(m)} \neq \emptyset$ and $\Sigma^{(m+i)} = \emptyset$ for all $i \geq 1$.

Definition 2.2 (Ranked Trees)**.** Let Σ be a ranked alphabet. The set $\mathcal{T}_\Sigma$ of ranked trees over Σ is defined by the grammar rules

$$u \;::=\; \mathtt{a}(\overbrace{u, \ldots, u}^{m \text{ times}}) \mid \mathtt{b}$$

where $\mathtt{a} \in \Sigma^{(m)}$ and $\mathtt{b} \in \Sigma^{(0)}$.

In the following, we use the term 'tree' as a synonym for *ranked tree*. In the company structure example in Figure 1.2, the symbol `employee` has rank 2. Every node with label `employee` in the tree has exactly two children which have the labels `data` and `subordinates`, respectively. On the other side, for the symbol `department` there is no fixed rank. In different companies, there are different many employees. Thus, in order to define transducers on arbitrary trees and forests, e.g., on XML documents, we rely on ranked tree representations of forests, e.g., through binary trees. The empty forest then is represented by a leaf with label `e` (where `e` is a new symbol that does not appear in the document). The content of an element node `a` is coded as the left child of `a`, while the forest of right siblings of `a` is represented as the right child. This is the well-known *first-child next-sibling encoding*, cf. the textbooks [CDG+07, Knu97]. Accordingly, the ranks of symbols are either 0 or 2. Figure 2.1 shows the binary encoding of the tree of the company structure presented in Figure 1.2(b) in the introduction. In the following, we refer to the unranked tree of this example structure by u_B and its encoding by $u_B{}'$. Several other encodings of unranked trees or forests by ranked trees have been used in the literature, for instance the *extension encoding* was presented in [CNT04, Tak75]. The encoding *encode* in [MSV03] is similar to the first-child next-sibling encoding.

A tree u', which occurs in a tree u is called *subtree* of u. The set $\texttt{Subtrees}(u)$ of all subtrees of a tree u is recursively defined as follows:

$$\begin{aligned} \texttt{Subtrees}(\mathtt{b}) &= \{\mathtt{b}\} \\ \texttt{Subtrees}(\mathtt{a}(u_1, \ldots, u_m)) &= \{\mathtt{a}(u_1, \ldots, u_m)\} \cup \bigcup_{i \in [m]} \texttt{Subtrees}(u_i) \end{aligned}$$

where `b` and `a` are labels of rank 0 and m, respectively. For instance, the tree

$$\texttt{Form}(\texttt{Tense}(\text{past tense}), \texttt{Person}(1^{st} \text{ sing.}))$$

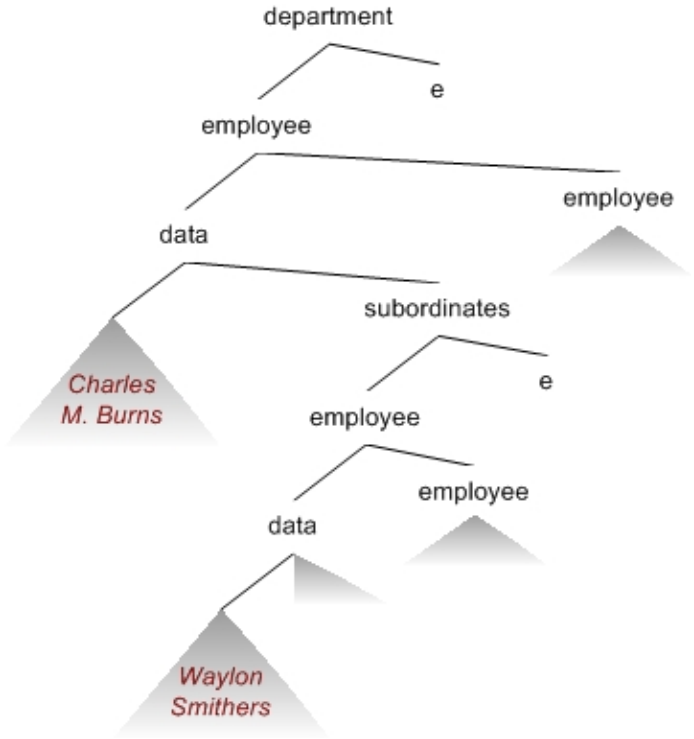


Figure 2.1: The binary encoding $u_B{}'$ of the tree in Figure 1.2(b).

is a subtree of the syntax tree in Figure 1.1(a). The height of this (sub)tree is 3. We define the *height* of a tree recursively as

$$\begin{aligned} \mathit{height}(\mathtt{b}) &= 1 && \text{for } \mathtt{b} \in \Sigma^{(0)}, \\ \mathit{height}(\mathtt{a}(u_1, \ldots, u_m)) &= 1 + \max\{\mathit{height}(u_1), \ldots, \mathit{height}(u_m)\} && \text{for } \mathtt{a} \in \Sigma^{(m)}. \end{aligned}$$

The height of a tree is the maximal length of a path from the root to a leaf plus one (cf. Section 2.2).

We often consider trees with variables. For example, the right-hand sides of rules of deterministic bottom-up tree transducers are such trees (Chapter 4). Let $X = \{x_1, x_2, \ldots\}$ denote a countable set of distinct variables of rank 0. Trees with variables of X are trees in $\mathcal{T}_\Sigma(X) = \mathcal{T}_{\Sigma \cup X}$. Let $t \in \mathcal{T}_\Sigma(X)$ be such a tree. We abbreviate by $t[t_1, \ldots, t_k]$ the substitution $t[t_1/x_1, \ldots, t_k/x_k]$ of trees t_i for the variables x_i ($i \in [k]$) in the tree t. The output of bottom-up tree transducers is built by replacing the variables of trees in $\mathcal{T}_\Sigma(X)$ by trees. For a finite set of m distinct variables, we also write $\mathcal{X}_m = \{x_1, \ldots, x_m\}$. Talking about the semantics of transducers, X is often a singleton and, in this case, we also use variable y instead of x_1.

2.2 Nodes and Paths

The same tree may occur several times as a subtree. For instance, the subtree $\texttt{Person}(1^{st}$ sing.) occurs twice in the syntax tree of the sentence in passive voice in Figure 1.1(a). It is necessary to distinguish between a subtree and an occurrence of a subtree. Assigning unique coordinates to the nodes of a ranked tree enables us to indicate a certain occurrence of a subtree by the coordinates of its root.

Definition 2.3 (Nodes)**.** The set $\texttt{Nodes}(u) \subseteq \mathbb{N}^*$ of all *nodes* ϑ in a ranked tree $u \in \mathcal{T}_\Sigma$ is defined as

$$\begin{aligned} \texttt{Nodes}(\texttt{b}) &= \{\epsilon\} \\ \texttt{Nodes}(\texttt{a}(u_1, \ldots, u_m)) &= \{\epsilon\} \cup \{i\vartheta \mid i \in [m], \vartheta \in \texttt{Nodes}(u_i)\} \end{aligned}$$

where $\texttt{b}$ and $\texttt{a}$ are symbols of rank 0 and m, respectively, and $u_i \in \mathcal{T}_\Sigma$ for $i \in [m]$.

Note that this definition also holds for trees in $\mathcal{T}_\Sigma(X)$ for a set X of variables. The two occurrences of the subtree $\texttt{Person}(1^{st}$ sing.) in the tree in Figure 1.1(a) are now uniquely indicated by their root nodes 1.2 and 2.2.2. Here, we write 1.2 instead of 12 for the second child of the first child of the root, to distinguish between this node and the twelfth son of the root. The *direction* $\eta(\vartheta)$ of a node ϑ indicates whether ϑ is the root of the tree or a particular child, i.e., we define $\eta(\epsilon) = 0$ and $\eta(\vartheta' j) = j$. Both nodes 1.2 and 2.2.2 have the direction 2, they are the second child of their fathers. The *depth* $|\vartheta|$ of a node ϑ is the length of the string describing ϑ, i.e., $|\epsilon| = 0$ and $|i\vartheta'| = 1 + |\vartheta'|$. The *size* $|u|$ of a tree u is defined as the number of nodes, i.e., $|u| = |\texttt{Nodes}(u)|$. For a ranked tree u and a given node $\vartheta \in \texttt{Nodes}(u)$, the tree $u[\vartheta]$ is called the *subtree of* u *located at* ϑ and is defined as

$$\begin{aligned} u[\epsilon] &= u \\ \texttt{a}(u_1, \ldots, u_m)[i\vartheta] &= u_i[\vartheta] \qquad\qquad \text{for } i \in [m]. \end{aligned}$$

With $lab_u(\vartheta)$ we refer to the label of the node ϑ in a tree u, or $lab(\vartheta)$ if u is given by the context. Let u_p be the tree in Figure 1.1(a). Then, we get

$$u_p[1.2] = u_p[2.2.2] = \texttt{Person}(1^{st} \text{ sing.}) \; .$$

The label of these nodes is $lab_{u_p}(1.2) = lab_{u_p}(2.2.2) = \texttt{Person}$.

Similar notions also apply to forests. In particular, the set $\texttt{Nodes}(f)$ of nodes of a forest f is defined as

$$\begin{aligned} \texttt{Nodes}(\epsilon) &= \emptyset \\ \texttt{Nodes}(\texttt{a}\langle f_1 \rangle f_2) &= \{0\} \cup \{0\vartheta \mid \vartheta \in \texttt{Nodes}(f_1)\} \cup \{(i+1)\vartheta \mid i\vartheta \in \texttt{Nodes}(f_2)\} \; . \end{aligned}$$

Note that the definition of nodes of a ranked tree differs from the definition of nodes in a forest consisting of one tree only. Accordingly, also the definition of *direction* differs. For a node ϑ in a forest we define the *direction* $\eta(\vartheta)$, which

now indicates whether ϑ is at the top-level, has a left sibling or both. Thus, $\eta(0) = 0$, $\eta(i) = 1$ for $i > 0$, $\eta(\vartheta'0) = 2$ for $\vartheta' \neq \epsilon$ and $\eta(\vartheta) = 3$ otherwise. The subforest $f[\vartheta]$ at a node ϑ in a forest is defined by:

$$f[i] \quad = \quad \begin{cases} f & \text{if } i = 0 \\ f'[i-1] & \text{if } i > 0 \wedge f = uf' \end{cases}$$

Moreover, for $\vartheta = i\vartheta'$ with $\vartheta' \neq \epsilon$,

$$f[\vartheta] \quad = \quad \begin{cases} f_1[\vartheta'] & \text{if } i = 0 \wedge f = \mathtt{a}\langle f_1\rangle f_2 \\ f'[(i-1)\vartheta'] & \text{if } i > 0 \wedge f = uf' \,. \end{cases}$$

The *label* $lab_f(\vartheta)$ of ϑ in the forest f is defined by $lab_f(0) = \epsilon$ if $f = \epsilon$, and

$$lab_f(i\vartheta') \quad = \quad \begin{cases} \mathtt{a} & \text{if } i = 0 \wedge \vartheta' = \epsilon \\ lab_{f_1}(\vartheta') & \text{if } i = 0 \wedge \vartheta' \neq \epsilon \\ lab_{f_2}((i-1)\vartheta') & \text{if } i > 0 \end{cases}$$

if $f = \mathtt{a}\langle f_1\rangle f_2$. Note that the label at a node in a forest thus either is from Σ or equals the empty forest ϵ. Consider for instance the unranked tree u_B (in Figure 1.2(b)) seen as a forest. We observe that the `subordinate`-node has the coordinates 0.0.1. The direction of this node is $\eta(0.0.1) = 3$. The subforest describing the personal data of Waylon Smithers (with `name` as root label of the first tree) is $u_B[0.0.1.0.0.0]$. Its father has the label $lab_{u_B}(0.0.1.0.0) = \mathtt{data}$. The direction of this node in u_B is $\eta(0.0.1.0.0) = 2$. Note that the subforest indicated by this node, i.e., $u_B[0.0.1.0.0]$ consists of the tree rooted at this `data`-node (the personal data of Waylon Smithers) and all its siblings.

In Part I we are not only interested in the node itself but also in the path from the root to the node including the sequence of labels at this path. A Σ-*path* is a string over the set $\{(\mathtt{a}, i) \mid \mathtt{a} \in \Sigma, 1 \leq i \leq \mathit{rank}(\mathtt{a})\}$.

Definition 2.4 (Paths)**.** The set $\mathtt{Paths}(u)$ of *paths* of the tree u is inductively defined as:

$$\mathtt{Paths}(\mathtt{b}) = \{\epsilon\}$$
$$\mathtt{Paths}(\mathtt{a}(u_1, \ldots, u_m)) = \{\epsilon\} \cup \{(\mathtt{a}, i).p \mid i \in [m], p \in \mathtt{Paths}(u_i)\}$$

The set of all paths over Σ is $\mathtt{Paths}_\Sigma = \bigcup_{u \in \mathcal{T}_\Sigma} \mathtt{Paths}(u)$.

The *node* of a path p is its stepwise projection to the second component, i.e.,

$$\mathit{node}(\epsilon) = \epsilon \quad \text{and} \quad \mathit{node}((\mathtt{a}, i).p') = i.\mathit{node}(p') \,.$$

The *subtree* $u[p]$ of a tree u at a path p and the *label* $lab_u(p)$ of p in u are given by the node of p in u: $u[p] = u[\mathit{node}(p)]$ and $lab_u(p) = lab_u(\mathit{node}(p))$ if $p \in \mathtt{Paths}(u)$, otherwise both are undefined. The *length* $|p|$ of a path p is given by the depth of its node, it is $|p| = |\mathit{node}(p)|$. We say that a path p *belongs to a tree* u if $p \in \mathtt{Paths}(u)$.

Let $u_{\mathtt{F}} = u_p[2.2]$ the subtree of the syntax tree u_p with root label $\mathtt{Form}$ (cf. Figure 1.1(a)). The path $p = (\mathtt{Form}, 2)(\mathtt{Person}, 1)$ refers to the leaf with label '1^{st} sing.'. The node of this path is $node(p) = 2.1$ and the length of this path is $|p| = 2$. Remember that the height of this tree $|u_{\mathtt{F}}|$ defined by the length of the longest path from the root to a leaf plus one is 3 (cf. Page 15).

Let u be a tree in $\mathcal{T}_\Sigma$. For a symbol $\mathtt{a} \in \Sigma$, the set $\mathtt{Paths}_{\mathtt{a}}(u)$ denotes the set of all paths $p \in \mathtt{Paths}(u)$ with $lab_u(p) = \mathtt{a}$. For a set $\Sigma' \subseteq \Sigma$, the set $\mathtt{Paths}_{\Sigma'}(u) = \bigcup_{\mathtt{a}\in\Sigma'} \mathtt{Paths}_{\mathtt{a}}(u)$ is the set of Σ'-*labeled paths* of tree $u \in \mathcal{T}_\Sigma$. Note that these definitions hold also for trees with variables if $\mathcal{T}_\Sigma(X)$ is interpreted as $\mathcal{T}_{\Sigma\cup X}$. Mostly, we consider for a variable y the y-*paths* of a tree in $\mathcal{T}_\Sigma(y)$, i.e., the set $\mathtt{Paths}_y(t)$. We also say *variable path* if the variable is clear by the context.

2.3 Tree Monoid

In the first part of this thesis, we need to distinguish different kinds of trees: Trees without any variable, trees with at least one occurrence of a dedicated variable y, and trees with exactly one occurrence of y. The latter are called *context*. Together, trees with or without occurrences of the variable y build a monoid with substitution into the variable as binary operation.

First, we consider trees possibly containing a dedicated variable $y \notin \Sigma$ of rank 0. Let $\mathcal{T}_\Sigma(y)$ denote this set: $\mathcal{T}_\Sigma(y) = \mathcal{T}_{\Sigma\cup\{y\}}$. On $\mathcal{T}_\Sigma(y)$, we define a binary operation "$\cdot$" by substitution into the variable y:

$$t_1 \cdot t_2 = t_1[t_2/y]$$

Note that every occurrence of y in t_1 will be replaced by the same tree t_2. For example, $\mathtt{Form}(y, y) \cdot \mathtt{Person}(y) = \mathtt{Form}(\mathtt{Person}(y), \mathtt{Person}(y))$. The result of $t_1 \cdot t_2$ is a ground tree, i.e., does not contain y, iff $t_1 \in \mathcal{T}_\Sigma$ or $t_2 \in \mathcal{T}_\Sigma$. For instance, 3^{rd} plural $\cdot\, \mathtt{Person}(y) = 3^{rd}$ plural. Moreover, the operation "$\cdot$" is associative with neutral element y. Therefore, the set $\mathcal{T}_\Sigma(y)$ together with the operation "$\cdot$" and y forms a monoid. Let $\hat{\mathcal{T}}_\Sigma(y)$ denote the sub-monoid consisting of all trees, which contain at least one occurrence of y, i.e., $\hat{\mathcal{T}}_\Sigma(y) = \mathcal{T}_\Sigma(y) \setminus \mathcal{T}_\Sigma$. Then $\mathcal{T}_\Sigma(y) = \hat{\mathcal{T}}_\Sigma(y) \cup \mathcal{T}_\Sigma$, and we have:

Proposition 2.1. *[Eng80]*

1. *Let $z, z', t_1, t_2, t_1', t_2' \in \mathcal{T}_\Sigma(y)$ with $t_1 \neq t_2$ and $t_1' \neq t_2'$. Assume that the two equalities $z \cdot t_1 = z' \cdot t_1'$ and $z \cdot t_2 = z' \cdot t_2'$ hold. Then one of the following two assertions is true:*
   (a) *$z, z' \in \mathcal{T}_\Sigma$ and $z = z'$; or*
   (b) *both z and z' contain an occurrence of y, i.e., are from $\hat{\mathcal{T}}_\Sigma(y)$ and $z \cdot s = z'$ or $z = z' \cdot s$ for some $s \in \hat{\mathcal{T}}_\Sigma(y)$.*
2. *The sub-monoid $\hat{\mathcal{T}}_\Sigma(y)$ is free.*

If there are $z, z' \in \mathcal{T}_\Sigma(y)$ and $t, t' \in \hat{\mathcal{T}}_\Sigma(y)$ with $z \cdot t = z' \cdot t'$, then already one of the two assertions (a) and (b) is true.

Unfortunately, this proposition does not hold for forests. Let $\mathcal{F}_\Sigma(y)$ be the set of forests possibly containing the variable y (similar to $\mathcal{T}_\Sigma(y)$). For the forests $f = \mathtt{b}yy$ and $f' = yy\mathtt{b}$ (i.e., forests with three trees each of height 1), we get that

$$f[y\mathtt{b}/y] = f'[\mathtt{b}y/y] \ ,$$

but neither $f = f'$ nor is there a forest $g \in \hat{\mathcal{F}}_\Sigma(y)$ such that $f[g/y] = f'$ or $f = f'[g/y]$ where $\hat{\mathcal{F}}_\Sigma(y)$ is the set of all forests with at least one occurrence of y. Since most of the results of Part I rely on Proposition 2.1, they cannot be adapted to forest transformations in a straightforward manner.

Furthermore, we also need the following relations. Consider the set

$$\hat{\mathcal{T}}_\Sigma(y)_\bot = \hat{\mathcal{T}}_\Sigma(y) \cup \{\bot\}$$

of all trees containing at least one occurrence of the variable y enhanced with an extra bottom element $\bot$ (not in $\Sigma \cup \{y\}$). On this set, we define a partial ordering by $\bot \sqsubseteq s$ for all s, and $s_1 \sqsubseteq s_2$ for $s_1, s_2 \in \hat{\mathcal{T}}_\Sigma(y)$ iff $s_1 = s' \cdot s_2$ for a suitable $s' \in \hat{\mathcal{T}}_\Sigma(y)$. Note that the trees s_1, s_2 and s' are not ground, i.e., they contain at least one occurrence of y. For example, for the trees on Page 18, we have $\bot \sqsubseteq \mathtt{Form}(\mathtt{Person}(y), \mathtt{Person}(y)) \sqsubseteq \mathtt{Person}(y)$. The greatest element w.r.t. the ordering $\sqsubseteq$ is y while the least element is given by $\bot$. With respect to this ordering, we observe:

1. Every $s \in \hat{\mathcal{T}}_\Sigma(y)$ has finitely many upper bounds only.
2. For every $s_1, s_2 \in \hat{\mathcal{T}}_\Sigma(y)_\bot$, there exists a least upper bound $s_1 \sqcup s_2$ in $\hat{\mathcal{T}}_\Sigma(y)_\bot$.

Since $\hat{\mathcal{T}}_\Sigma(y)_\bot$ also has a least element, namely $\bot$, we conclude that $\hat{\mathcal{T}}_\Sigma(y)_\bot$ is a *complete* lattice satisfying the ascending chain condition, i.e., every set $S \subseteq \hat{\mathcal{T}}_\Sigma(y)_\bot$ has a least upper bound $s = \bigsqcup S$, and there are no infinite strictly ascending sequences $\bot \sqsubset s_1 \sqsubset s_2 \sqsubset \ldots$ where $s \sqsubset s'$ iff $s \sqsubseteq s'$ and $s \neq s'$. We call the least upper bound of a set $S \subseteq \hat{\mathcal{T}}_\Sigma(y)_\bot$ the *greatest common suffix* of all trees in S and denote it by $\mathsf{gcs}(S)$. Thus, we have

$$\mathsf{gcs}(S) = \bigsqcup S$$

w.r.t. the order $\sqsubseteq$ on $\hat{\mathcal{T}}_\Sigma(y)_\bot$. We call a tree $s \in \hat{\mathcal{T}}_\Sigma(y)$ *irreducible* if $s \neq y$ and $s \sqsubseteq s'$ only holds for $s' \in \{y, s\}$. The tree $\mathtt{Person}(y)$ is irreducible.

Example 2.1. According to the example of syntax trees for sentences (Figure 1.1), consider the trees

$$\begin{aligned} t_{s3} &= \mathtt{Subject}(\mathtt{Content}(y), \mathtt{Person}(3^{rd} \text{ plural})) \\ t_{s1} &= \mathtt{Subject}(\mathtt{Content}(y), \mathtt{Person}(1^{st} \text{ sing.})) \\ t_{o1} &= \mathtt{Object}(\mathtt{Content}(y), \mathtt{Person}(1^{st} \text{ sing.})) \ . \end{aligned}$$

For the first tree, we observe that its subtree $\texttt{Content}(y)$ is greater than t_{s3}, i.e., $t_{s3} \sqsubseteq \texttt{Content}(y)$. This follows because we can decompose the tree as follows:

$$t_{s3} = \texttt{Subject}(y, \texttt{Person}(3^{rd} \text{ plural})) \cdot \texttt{Content}(y)$$

Also y is greater than t_{s3} and for both other trees, we get the same upper bounds. Additionally, each of these trees is a upper bound of itself. There are no more upper bounds of these trees. Furthermore, $\texttt{Content}(y) \sqsubseteq y$. Thus, we get that the greatest common suffix of the set $S = \{t_{s3}, t_{s1}, t_{o1}\}$ is

$$\texttt{gcs}(S) = \texttt{Content}(y) \ .$$

It is the least tree of the common upper bounds of t_{s3}, t_{s1}, and t_{o1}.

Now consider the trees

$$t_{sy} = \texttt{Subject}(\texttt{Content}(y), \texttt{Person}(y))$$
$$t_{oy} = \texttt{Object}(\texttt{Content}(y), \texttt{Person}(y)) \ .$$

Neither $\texttt{Content}(y)$ nor $\texttt{Person}(y)$ is an upper bound of one of these trees. Only y and t_{sy} are upper bounds of t_{sy}. Thus, this tree is irreducible. Analog, it follows that t_{oy} is irreducible. We get that the greatest common suffix of t_{sy} and t_{oy} is y. ◁

We enhance the definition of the partial order $\sqsubseteq$ to trees in $\mathcal{T}_\Sigma(y) \cup \{\bot\}$ by $\bot \sqsubseteq t$ for all $t \in \mathcal{T}_\Sigma(y) \cup \{\bot\}$, and $t_1 \sqsubseteq t_2$ for $t_1, t_2 \in \mathcal{T}_\Sigma(y)$ iff $t_1 = s \cdot t_2$ for a suitable $s \in \hat{\mathcal{T}}_\Sigma(y)$. Note that the prefix s of t_1 must contain y. With this condition, the tree t_2 really occurs in t_1.

Two trees (with or without the variable y) are unifiable if they are prefixes of the same tree. Let $\top \notin \Sigma \cup \{y, \bot\}$ be a new symbol. Assume that $t_1, t_2 \in \mathcal{T}_\Sigma(y)$ are trees, and that there are trees $s_1, s_2 \in \mathcal{T}_\Sigma(y) \cup \{\top\}$ such that $t_1 \cdot s_1 = t_2 \cdot s_2$. Note that $t_i \cdot s_i = t_i$ if $t_i \in \mathcal{T}_\Sigma$. Then we call t_1, t_2 *unifiable* and $\langle s_1, s_2 \rangle$ a *unifier* of t_1, t_2. Note that we substitute the variable y in t_1 and t_2 possibly differently. In terms of the common unification problem [BS01, KB70], we can interpret the trees t_1 and t_2 as trees with different variables, i.e., unifying t_1 and $t_2[y']$ for some new variable $y' \neq y$.

We consider the set

$$\mathbb{D}_\Sigma = (\{y\} \times \hat{\mathcal{T}}_\Sigma(y)) \cup (\hat{\mathcal{T}}_\Sigma(y) \times \{y\}) \cup (\mathcal{T}_\Sigma \cup \{\top\})^2 \cup \{\bot\}$$

of *candidate unifiers*. The set $\mathbb{D}_\Sigma$ forms a complete lattice w.r.t. the ordering $\leq$ defined by

- $\bot \leq d \leq \langle \top, \top \rangle$ for all $d \in \mathbb{D}_\Sigma$,
- $\langle t_1, t_2 \rangle \leq \langle t_1', t_2' \rangle$ if $t_i = t_i' \cdot u$ for all $i \in \{1, 2\}$ for some tree $u \in \mathcal{T}_\Sigma \cup \{y\}$, and
- $\langle t_1, t_2 \rangle \leq \langle t_1, \top \rangle$ and $\langle t_1, t_2 \rangle \leq \langle \top, t_2 \rangle$ if $t_1, t_2 \in \mathcal{T}_\Sigma$.

Then the *most-general unifier* $\mathsf{mgu}(t_1, t_2) \in \mathbb{D}_\Sigma$ for trees $t_1, t_2 \in \mathcal{T}_\Sigma(y)$ is the greatest unifier of t_1, t_2 w.r.t. the ordering $\leq$. It is $\bot$ if t_1, t_2 are not unifiable. Furthermore, for a set of pairs $P \subseteq \mathcal{T}_\Sigma(y)^2$ the most-general unifier $\mathsf{mgu}(P)$ is the least upper bound of the unifiers of pairs in P, i.e.,

$$\mathsf{mgu}(P) = \bigvee \{\mathsf{mgu}(t_1, t_2) \mid (t_1, t_2) \in P\} \; .$$

If t_1 is ground and t_1, t_2 are unifiable by some pair $\langle s_1, s_2 \rangle$ then $t_1 \cdot s_1' = t_1 = t_2 \cdot s_2$ holds for all $s_1' \in \mathcal{T}_\Sigma(y)$. It follows that t_1, t_2 are unifiable by each pair $\langle s_1', s_2 \rangle$ with $s_1' \in \mathcal{T}_\Sigma(y)$. Thus, the most-general unifier is either $\langle \top, s_2 \rangle$ if t_2 is not ground, or $\langle \top, \top \rangle$ if also $t_2 \in \mathcal{T}_\Sigma$.

Example 2.2. Again, we consider parts of syntax trees of sentences. Thereto, let Σ be a ranked alphabet containing the symbols $\mathtt{Form}^{(2)}$, $\mathtt{Person}^{(1)}$, $\mathtt{Tense}^{(1)}$, 3^{rd} plural$^{(0)}$, and past tense$^{(0)}$ (the exponents give the ranks of the symbols). Consider the trees

$$\begin{aligned} t_1 &= \mathtt{Form}(y, \mathtt{Person}(3^{rd} \text{ plural})) \\ t_2 &= \mathtt{Form}(\mathtt{Tense}(\text{past tense}), \mathtt{Person}(y)) \\ t_3 &= \mathtt{Form}(\mathtt{Tense}(y), y) \\ t_4 &= \mathtt{Form}(\mathtt{Tense}(y), \mathtt{Person}(3^{rd} \text{ plural})) \end{aligned}$$

The trees t_1 and t_2 are unifiable by the unifier $\langle \mathtt{Tense}(\text{past tense}), 3^{rd} \text{ plural} \rangle$. This pair is already the most-general unifier $\mathsf{mgu}(t_1, t_2)$ of the two trees. The trees t_1 and t_3 have the most-general unifier

$$\mathsf{mgu}(t_1, t_3) = \langle \mathtt{Tense}(\mathtt{Person}(3^{rd} \text{ plural})), \mathtt{Person}(3^{rd} \text{ plural}) \rangle \; ,$$

whereas the trees t_2 and t_3 are not unifiable, i.e., $\mathsf{mgu}(t_2, t_3) = \bot$. Finally, consider the trees t_4 and t_1, for which $\langle \mathtt{Tense}(\text{past tense}), \text{past tense} \rangle$ is a unifier. Also $\langle \mathtt{Tense}(3^{rd} \text{ plural}), 3^{rd} \text{ plural} \rangle$ or $\langle \mathtt{Tense}(y), y \rangle$ are possible unifiers of this pair. Thus, we observe that the most-general unifier is in $\hat{\mathcal{T}}_\Sigma(y) \times \{y\}$, it is

$$\mathsf{mgu}(t_1, t_4) = \langle \mathtt{Tense}(y), y \rangle \; .$$

Note that the unifier $\langle \mathtt{Tense}(\mathtt{Tense}(y)), \mathtt{Tense}(y) \rangle$ of t_1 and t_4 is no candidate unifier, i.e., is not in $\mathbb{D}_\Sigma$, because both parts are in $\hat{\mathcal{T}}_\Sigma(y)$ and different to y. $\triangleleft$

For the semantics of transducers, we also need the definition of contexts over an alphabet Σ. A *context* c is a tree $c \in \hat{\mathcal{T}}_\Sigma(y)$, which contains exactly one occurrence of y. The set of all contexts is denoted by $\mathcal{C}_\Sigma(y)$ (or $\mathcal{C}_\Sigma$). The *length* of a context c is the length of the path from the root to y. If context c has length n, then there are irreducible trees $c_1, \ldots, c_n$ such that $c = c_1 \cdot c_2 \cdots c_n$. In the previous example, the trees t_1, t_2, and t_4 are contexts, in contrast to t_3. The length of t_4 is 2 because the y-path of t_4 is $(\mathtt{Form}, 1)(\mathtt{Tense}, 1)$.

Let $\sim$ be a equivalence relation over a set $U \subseteq \mathcal{T}_\Sigma$. It is a *congruence* (w.r.t. to contexts in $\mathcal{C}_\Sigma(y)$) if for all $c \in \mathcal{C}_\Sigma(y)$ and all $u, u' \in U$ holds:

$$u \sim u' \;\text{ and }\; c \cdot u \in U \quad \Longrightarrow \quad c \cdot u' \in U \;\text{ and }\; c \cdot u \sim c \cdot u'$$

We write $U/_\sim$ to denote the set of equivalence classes of $\sim$ over U.

2.4 Trees with State Calls

In Part II, we consider (macro) tree walking transducers. The right-hand sides of rules of such transducers are trees containing *state calls*. The definitions related to trees like the height of a tree have to be enhanced to trees with state calls. Tree walking transducers (2TTs, cf. Section 9), which do not support accumulating parameters, have rules like, e.g.,

$$q_I(\texttt{department}) \;\to\; \texttt{staff}(q(\mathit{down}_1), \texttt{e})$$

where q_I and q are states, `department` is an input symbol and `staff` and `e` are output symbols of rank 2 and 0, respectively. For 2TTs, state calls are of the form $q(op)$ where op stands for up, $stay$, or $down_i$, like $q(down_1)$ in the example rule. To enhance the definitions related to trees to right-hand sides of rules of 2TTs, we deal with $q(op)$ as a symbol of rank 0 for all states q and all operations op. The right-hand sides of tree walking transducers are trees over the alphabet $\Sigma \cup (Q \times Op)$ where Q is the set of states and Op the set of possible operations: $Op = \{up, stay\} \cup \{down_i \mid i \in \mathbb{N}\}$. We get $height(q(op)) = 1$ for all states q and all operations op. The right-hand side $\texttt{staff}(q(down_1), \texttt{e})$ of the example rule has height 2.

In Chapter 12, we then will add parameters to state calls to obtain *macro tree walking transducers* (2MTTs). In this case, we fix a set $Y = \{y_1, y_2, \ldots\}$ of formal *parameters*. These parameters are of rank 0. We use the same definition as for sets of variables X (Page 15). Then, states have a fixed rank, which determines the number of parameters. A rule of a 2MTT is for instance:

$$q_I(\texttt{department}, 0) \;\to\; \texttt{staff}(q(\mathit{down}_1, \texttt{e}, \texttt{e}), \texttt{e})$$

On the right-hand side, we have the state q of rank 3. The exact definition of rules of 2MTTs with ranked states will be given in Chapter 12. In a 2MTT, a state call has the form $q(op, t_1, \ldots, t_n)$ where t_i $(i \in [n])$ are ranked trees over $\Sigma \cup Y$ and further state calls. These right-hand sides are seen as trees over the alphabet $\Sigma \cup Y \cup (Q \times Op)$ where for all states q and all operations op, a label of the form (q, op) has the rank $rank(q) - 1$. Figure 2.2 illustrates the tree representation of the right-hand side of the example rule. There, the state call $(q, down_1)$ has rank 2. In general, there may occur further state calls in the subtrees. Additionally, if the state on the left-hand side has a rank greater than 1, there may occur parameters y_i on the right-hand side. For these trees on right-hand sides of 2MTTs, we define the height recursively by:

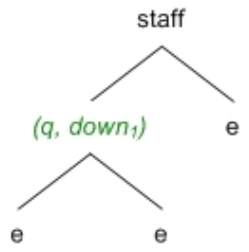


Figure 2.2: Tree representation of a right-hand side in a 2MTT.

$$height(q(op, t_1, \ldots, t_n)) = 1 + max(height(t_1), \ldots, height(t_n))$$

Part I

Myhill-Nerode Theorem for Bottom-Up Tree Transducers

Chapter 3

Introduction

In Chapter 1, we pointed out the relevance of a Myhill-Nerode theorem for tree transducers. In this part, we now present a Myhill-Nerode theorem for deterministic bottom-up tree transducers (DBTTs). We provide a semantic characterization of tree transformations, which are definable by DBTTs. Thereto, we split a tree-to-tree transformation into a subtree function, which provides an output tree for every subtree of an input tree, and context functions, which map contexts to images. Such a division is called *partition*. In Figure 3.1, a partition for the example transformation, which translates sentences in passive voice into sentences in active voice (Figure 1.1), is depicted. The `Object`-subtree of the input tree is transformed into the `Subject`-subtree of the output tree. Furthermore, we get an image for the context of the `Object`-subtree, the remainder of the output tree. Note that the grammatical person of the verb phrase in the image of the context is determined by the subtree. Thus, different subtrees may induce different transformations of the same contexts. We will specify a unique partition for every transformation. Furthermore, for this unique partition, we will present necessary conditions that the transformation is definable by a DBTT. These conditions are that the partition has *finite index* and is *path-finite*. Roughly spoken, having finite index means that a partition has only finitely many different context functions and path-finiteness means that each path in input trees from the root to a node only affects finitely many different paths in output trees. Both properties are caused by, amongst others, the finiteness of the number of states of deterministic bottom-up tree transducers. We will prove that these two conditions are not only necessary but also sufficient, i.e., a transformation is definable by a DBTT if and only if its partition is path-finite and has finite index.

Moreover, the path-finite partition with finite index of a transformation leads to a unique minimal DBTT. Given an arbitrary deterministic bottom-up tree transducer, we show that this unique DBTT can be constructed. The construction is based on a sequence of normalizing transformations, which, among others, guarantee that non-trivial output is produced as early as possible. For a deterministic bottom-up tree transducer where every state produces either none

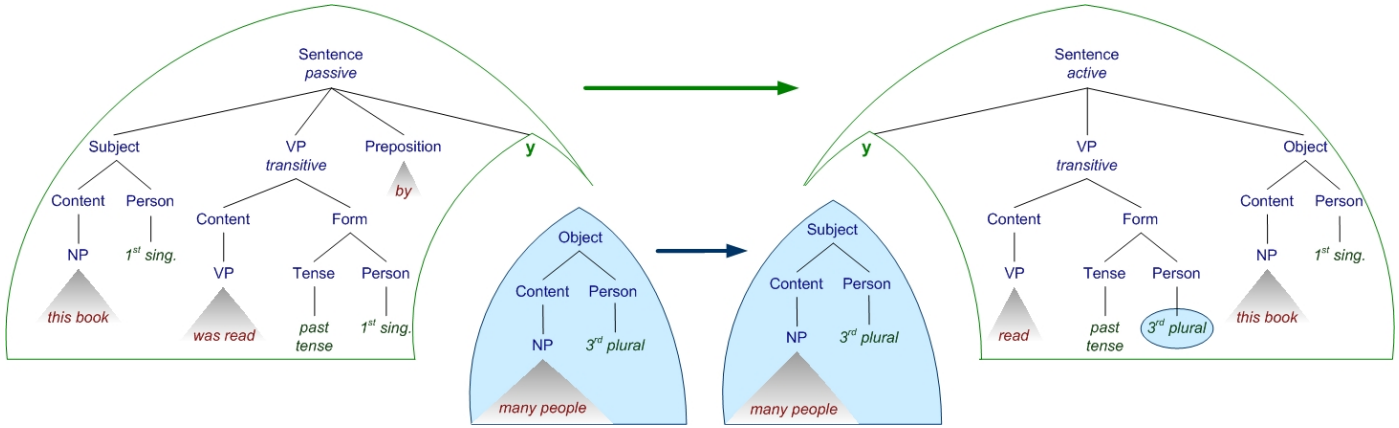


Figure 3.1: Transformation of a subtree and its context.

or infinitely many outputs, the minimal transducer can be constructed in polynomial time. This leads to a polynomial time algorithm to decide equivalence of deterministic bottom-up tree transducers.

Summarizing, we get the Myhill-Nerode theorem for deterministic bottom-up tree transducers. It says that for a tree transformation the following properties are equivalent:

1. The transformation is definable by a DBTT.
2. It exists a unique minimal DBTT describing this transformation.
3. The partition of the transformation is path-finite and has finite index, i.e., is a *bottom-up partition*.

Part I is organized as follows: Some basic notations were already given in the previous Chapter 2. In **Chapter 4**, we add the formal definition of deterministic bottom-up tree transducers and related notions and properties.

The semantic representation via partitions is given in **Chapter 5**. First, we define the basic terms that are used in this context such as subtree functions, context functions, and partitions. Every deterministic bottom-up tree transducer induces a dedicated partition. We deduce this partition and the inherent properties like path-finiteness. The necessity of these properties is proven. In the second section of this chapter (Section 5.2), we consider arbitrary transformations. We present a normal form for the partition of a transformation, which is the unique bottom-up partition if there is a path-finite partition of the transformation with finite index. This partition is disambiguated by some properties like producing non-trivial output as early as possible, i.e., by the subtree function, whereas trivial output is postponed to the context functions. In the end, the bottom-up partition is the partition of the same transformation with the minimal index fulfilling these properties.

In **Chapter 6** we prove the Myhill-Nerode theorem. First, we present the minimization of arbitrary deterministic bottom-up tree transducers (Section 6.1). It is shown that the minimal DBTT is unique and we prove that the construction can be done in polynomial time if states only produce infinitely many different output trees or none. The second part of the proof of the Myhill-Nerode theorem (Section 6.2) is that the partition of a minimal DBTT is already the bottom-up partition of its transformation. In the last section of this chapter (Section 6.3), it is shown that the bottom-up partition of a transformation induces a minimal DBTT, defining the same transformation. It is a technical procedure where we deduce the transition rules and states and prove their correctness.

We present conclusions and ideas of future work in **Chapter 7**. In particular, we give an outlook to a learning algorithm of deterministic bottom-up tree transducers.

Chapter 4

Bottom-Up Tree Transducers

In this part, we analyze deterministic bottom-up tree transducers. Such a finite state tree transducer starts at the leaves of a tree and processes the tree to the root. During the execution, the transducer builds a new tree starting with several subtrees. For instance, a bottom-up transducer for the example transformation in Figure 1.1 produces output trees for the four subtrees of the root node as illustrated in Figure 4.1 and memorizes crucial information (such as the person or the tense) in a state. Depending on these states and the label `Sentence passive` of the root node, it builds a tree using the four output trees (Figure 4.2).

For the following definition, we recall the notation of $\mathcal{X} = \{x_1, x_2, \ldots\}$ for a countable set of variables and $\mathcal{X}_m = \{x_1, \ldots, x_m\}$ for a finite subset of $\mathcal{X}$ with m variables (Page 15). For a ranked alphabet Σ, $mr(\Sigma)$ denotes the maximal rank of symbols in Σ.

Definition 4.1. A *deterministic bottom-up tree transducer* (DBTT for short) is a tuple $T = (Q, \Sigma, \Delta, R, F)$ where

- Q is a finite set of states,
- Σ and Δ are ranked input and output alphabets, respectively, disjoint with Q,
- $R : \bigcup_{m=0}^{mr(\Sigma)} (\Sigma^{(m)} \times Q^m) \dashrightarrow Q \times \mathcal{T}_\Delta(\mathcal{X})$ is a partial *transition function*, and
- $F : Q \dashrightarrow \mathcal{T}_\Delta(y)$ is a partial function mapping states to final outputs, called the *final function* of T.

For every input symbol $\mathtt{a} \in \Sigma$ of rank m and sequence of states $q_1, \ldots, q_m$, the transition function R provides at most one transition (or rule), which is denoted by $\mathtt{a}(q_1, \ldots, q_m) \to q(z)$ where $q \in Q$ and $z \in \mathcal{T}_\Delta(\mathcal{X}_m)$.

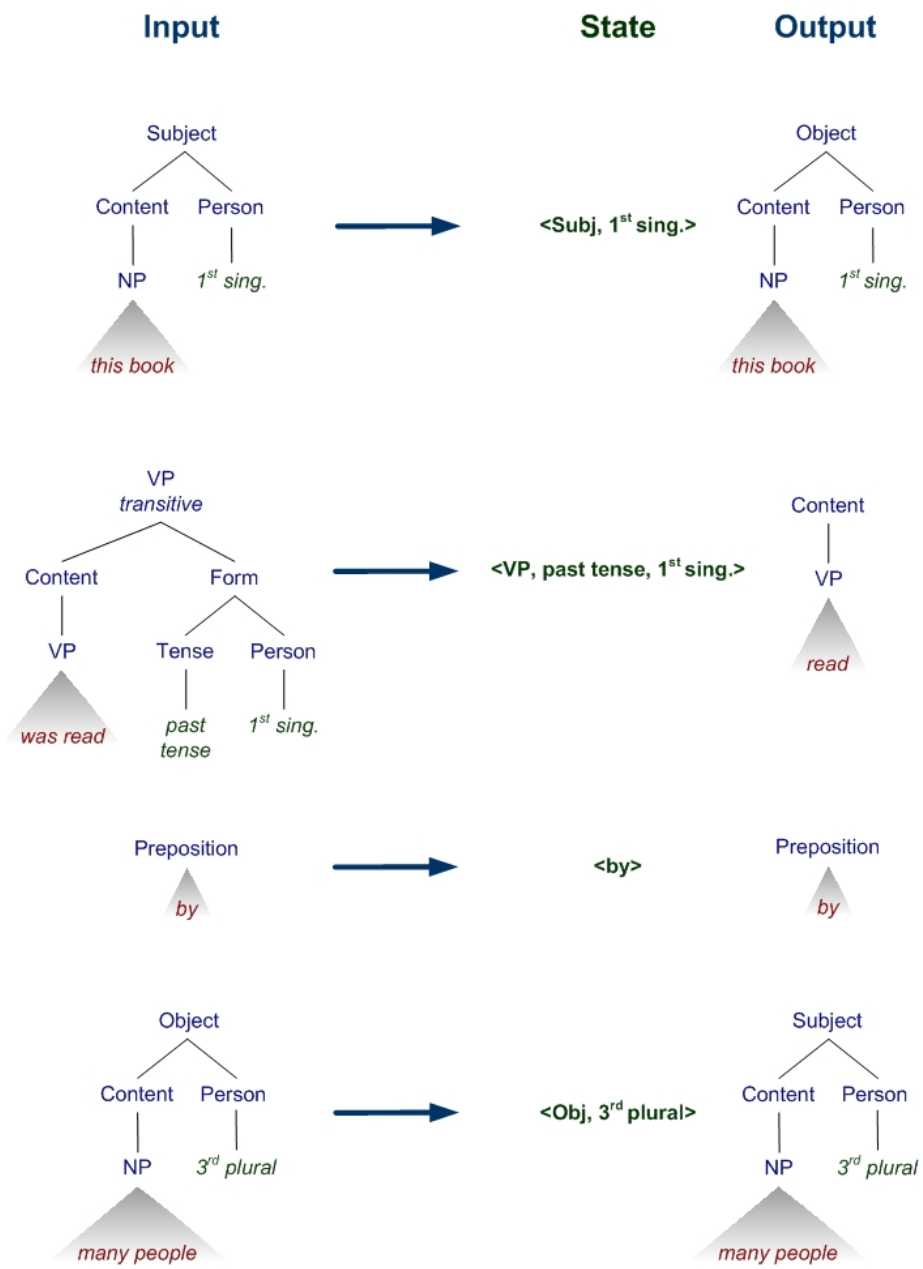


Figure 4.1: The outputs of selected subtrees and the reached states with further informations.

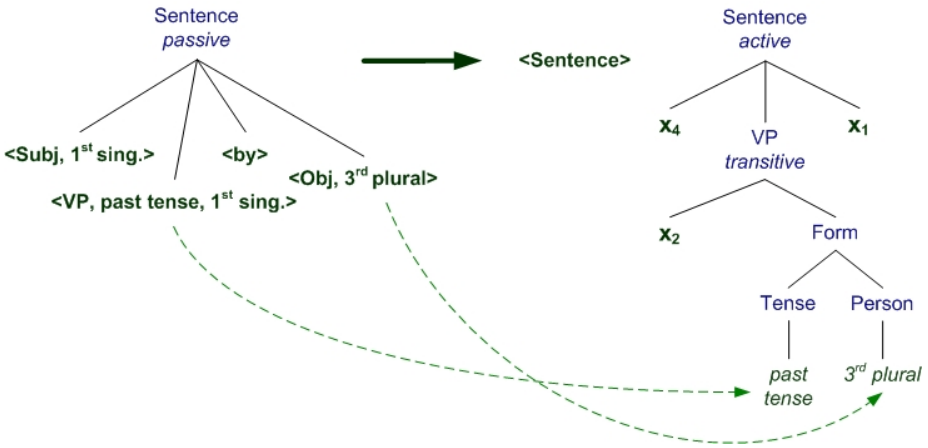


Figure 4.2: A transition to transform a root node of a syntax tree of a sentence in passive voice, depending on the states reached by the subtrees (dashed arrows); x_1, x_2, and x_4 mark where to insert the output of the 1st, 2nd and 4th subtree.

We only consider *deterministic* bottom-up tree transducers in this thesis. Thus, we use the term *bottom-up tree transducer* synonymous. Note that the right-hand side $q(z)$ is a tree over the ranked alphabet $\Delta \cup Q \cup \mathcal{X}$ where each state in Q has rank 1, and each variable in $\mathcal{X}$ has rank 0. Similarly, the left-hand side $\mathtt{a}(q_1, \ldots, q_m)$ can be considered as an abbreviation for the tree $\mathtt{a}(q_1(x_1), \ldots, q_m(x_m))$ over $\Sigma \cup Q \cup \mathcal{X}$.

For every input symbol $\mathtt{a}$ of rank m and sequence of states $q_1 \ldots q_m$ of Q, let $R(\mathtt{a}, q_1 \ldots q_m)$ be the right-hand side of the transition for $\mathtt{a}$ and $q_1 \ldots q_m$ if it is defined, and let $R(\mathtt{a}, q_1 \ldots q_m)$ be undefined otherwise.

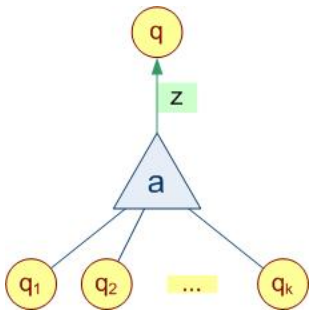
Example 4.1. In Figure 4.2, a transition of a DBTT for the transformation from sentences in passive voice into sentences in active voice is pictured. It is the transformation with the left-hand side

$$\texttt{Sentence passive}(\langle \text{Subj}, 1^{st} \text{ sing.}\rangle, \langle \text{VP}, \text{past tense}, 1^{st} \text{ sing.}\rangle, \langle \text{by}\rangle, \langle \text{Obj}, 3^{rd} \text{ plural}\rangle) \,.$$

The dashed arrows illustrate how the right-hand side depends on the states on the left-hand side of the transition. It can be applied to the root of the tree in Figure 1.1(a), when the four subtrees are transformed as in Figure 4.1. It would reach state $\langle \text{Sentence}\rangle$ and produces an output tree with root label `Sentence active`. The transducer also has a transition with the left-hand side

$$\texttt{Sentence passive}(\langle \text{Subj}, 1^{st} \text{ sing.}\rangle, \langle \text{VP}, \text{past tense}, 1^{st} \text{ sing.}\rangle, \langle \text{by}\rangle, \langle \text{Obj}, \mathbf{2^{nd}\ sing.}\rangle)$$

where the person of the object is 2^{nd} singular. In this case, the node 2.2.2.1 of the right-hand side is labeled with $\mathtt{2^{nd}}$ `sing.`, instead of $\mathtt{3^{rd}}$ `plural`. It does not



(a) The transition of the form $\mathtt{a}(q_1, q_2, \ldots, q_k) \to q(z)$.

(b) The final function of state q, i.e., $F(q) = z$.

Figure 4.3: Sketch of a transition and a final function of a DBTT.

contain a transition with left-hand side

$$\texttt{Sentence passive}(\langle \text{Subj}, \mathbf{1^{st}\ sing.}\rangle, \langle \text{VP}, \text{past tense}, \mathbf{3^{rd}\ plural}\rangle, \langle \text{by}\rangle, \langle \text{Obj}, 2^{nd}\ \text{sing.}\rangle)$$

where the persons of the subject and of the predicate differ. Because this does not occur in proper sentences. ◁

The *size* of a DBTT T, denoted by $|T|$, is the sum of the sizes (= number of symbols) of its final outputs and the sizes of the left-hand sides and right-hand sides of its transitions.

To illustrate DBTTs, we design states by circles, present output trees by squares, and input labels by triangles. In Figure 4.3(a), a transition of the form $\mathtt{a}(q_1, q_2, \ldots, q_k) \to q(z)$ is drawn. The order of the states on the left-hand side is given by the sorting of the incoming edges at the bottom of the triangle representing the label (not the position of the states). The output is located at the arrow connecting the tip of the triangle with the state of the right-hand side of the transition. The final function is presented in Figure 4.3(b). In general, we draw transducers or parts of them only with labels of rank less then 2 (Figure 4.4).

Example 4.2. Consider a smaller example. The following transformation checks if the input tree contains an even or an odd number of `b`-labeled nodes. Depending on that, the root of the output tree is labeled `Even` or `Odd`, respectively. In the second case, the input tree is copied where lower case letters are transformed into capital letters.

$$\tau : \mathtt{b}^n(\mathtt{e}) \mapsto \begin{cases} \mathtt{Even} & n \in \mathbb{N}_0 \text{ even} \\ \mathtt{Odd}(\mathtt{B}^n(\mathtt{E})) & n \in \mathbb{N}_0 \text{ odd} \end{cases}$$

It maps trees of $\mathcal{T}_\Sigma$ to trees of $\mathcal{T}_\Delta$ where the input alphabet is $\Sigma = \{\mathtt{b}^{(1)}, \mathtt{e}^{(0)}\}$ and the output alphabet is given by $\Delta = \{\mathtt{Odd}^{(1)}, \mathtt{B}^{(1)}, \mathtt{E}^{(0)}, \mathtt{Even}^{(0)}\}$, respectively. Both input and output trees are monadic. A DBTT defining this transformation is $T = (Q, \Sigma, \Delta, R, F)$ (Figure 4.4) with set of states $Q = \{q_{\text{even}}, q_{\text{odd}}\}$, the transition function R is given by

$$\begin{aligned} \mathtt{e} &\to q_{\text{even}}(\mathtt{E}) \\ \mathtt{b}(q_{\text{even}}) &\to q_{\text{odd}}(\mathtt{B}(x_1)) \\ \mathtt{b}(q_{\text{odd}}) &\to q_{\text{even}}(\mathtt{B}(x_1)) \ . \end{aligned}$$

The final function F is defined as follows:

$$F(q_{\text{even}}) = \mathtt{Even} \quad \text{and} \quad F(q_{\text{odd}}) = \mathtt{Odd}(y)$$

The transducer memorizes in the state if it has processed an even or an odd number of $\mathtt{b}$-labeled nodes. In the end, it possibly needs the copy of the input tree. Therefore, it produces for every node the corresponding output node. ◁

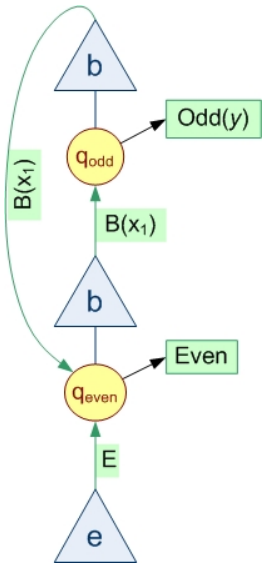


Figure 4.4: The DBTT T of Example 4.2.

Let $T = (Q, \Sigma, \Delta, R, F)$ be a DBTT. Assume that $t \in \mathcal{T}_\Sigma(y)$ and $q \in Q$. The *result* $[\![t]\!]_q^T$ of a computation of T on input t when starting in state q at variable leaves y is defined by induction on the structure of t:

$$\begin{aligned} [\![y]\!]_q^T &= q(y) \\ [\![\mathtt{a}(t_1, \ldots, t_m)]\!]_q^T &= q'(z[z_1, \ldots, z_k]) \\ &\quad \text{if } \forall i \ [\![t_i]\!]_q^T = q_i(z_i) \text{ and } R(\mathtt{a}, q_1 \ldots q_m) = q'(z) \end{aligned}$$

If $[\![t]\!]_q^T = q'(z')$, then z' is also called the *output* produced for t at q. Note that the function $[\![\,.\,]\!]_q^T$ may not be defined for all trees t. The superscript ^{T} can be omitted if T is clear from the context. If $t \in \mathcal{T}_\Sigma$ we also omit the subscript q, i.e., we write $[\![t]\!]^T$ for $[\![t]\!]_q^T$. In this case if $[\![t]\!]^T = q'(z)$, we say that q' is *reached by* t. For a ground tree $t \in \mathcal{T}_\Sigma$, which reaches some state q', we refer to the output by $\mathtt{out}^T(t)$, i.e., $\mathtt{out}^T(t) = z'$ iff $[\![t]\!]^T = q'(z')$.

The *image* $\tau_q^T(t)$ of the tree t is then defined by $\tau_q^T(t) = z' \cdot z$ iff $[\![t]\!]_q^T = q'(z)$ for some state q' with $F(q') = z'$. Again, we omit the subscript q if the tree t does not contain the variable y. The *transformation described* by DBTT T is the partial function

$$\tau^T : \mathcal{T}_\Sigma \dashrightarrow \mathcal{T}_\Delta \ .$$

Example 4.3. Consider again the transformation translating passive into active voice. In Figure 4.1 the results of the subtrees of the root of the input tree in Figure 1.1(a) are given. Since the four states correspond to the states in the transition given in Figure 4.2, this transition can be applied to the root of the input tree. By the definition, we get the result of the whole tree, by replacing

the variables x_1, x_2 and x_4 by the corresponding outputs of the first, the second, and the forth subtree, respectively. The result is the state $\langle$Sentence$\rangle$ together with the output tree in Figure 1.1(b). ◁

We say that two transducers T and T' are *equivalent* if they describe the same transformation, i.e., for all $u \in \mathcal{T}_\Sigma$, $\tau^T(u)$ is defined iff $\tau^{T'}(u)$ is defined and if they are defined $\tau^T(u)$ and $\tau^{T'}(u)$ are equal. We also use the following notations: The *language* $\mathcal{L}^T(q)$ of a state q is the set of all ground input trees by which q is reached, i.e.,

$$\mathcal{L}^T(q) = \{u \mid \exists v \in \mathcal{T}_\Delta : [\![u]\!]^T = q(v)\} .$$

A context $c \in \mathcal{C}_\Sigma$ (cf. definition on Page 21) is a *context* of a state q if $\tau^T_q(c)$ is defined. The set of all contexts of state q is denoted by $\mathcal{C}^T(q)$:

$$\mathcal{C}^T(q) = \{c \mid \exists z \in \mathcal{T}_\Delta(y) : \tau^T_q(c) = z\}$$

Example 4.4. The tree $\mathtt{e}$ is an input tree of the transducer T in Example 4.2. The result of this tree is $[\![\mathtt{e}]\!]^T = q_{\text{even}}(\mathtt{E})$ and its image is $\tau^T(\mathtt{e}) = \mathtt{Even}$ because $F(q_{\text{even}}) = \mathtt{Even}$. Note that the produced output $\mathtt{out}^T(\mathtt{e}) = \mathtt{E}$ disappears. However, the output is used to compute the result and output of other trees containing $\mathtt{e}$ as subtree. Consider the tree $\mathtt{b}(\mathtt{e})$ here, for example. With the transition $\mathtt{b}(q_{\text{even}}) \to q_{\text{odd}}(\mathtt{B}(x_1))$ and the result of $\mathtt{e}$ we get

$$[\![\mathtt{b}(\mathtt{e})]\!]^T = q_{\text{odd}}(\mathtt{B}(x_1)[\mathtt{E}/x_1]) = q_{\text{odd}}(\mathtt{B}(\mathtt{E})) .$$

For this tree, the output is also used in the image $\tau^T(\mathtt{b}(\mathtt{e}))$. The final function with $F(q_{\text{odd}}) = \mathtt{Odd}(y)$ leads to

$$\tau^T(\mathtt{b}(\mathtt{e})) = \mathtt{Odd}(y) \cdot \mathtt{B}(\mathtt{E}) = \mathtt{Odd}(\mathtt{B}(\mathtt{E})) .$$

In the end, we get that T describes the transformation τ of Example 4.2, i.e., $\tau = \tau^T$. The languages of the states of T are

$$\mathcal{L}^T(q_{\text{even}}) = \{\mathtt{b}^n(\mathtt{e}) \mid n \in \mathbb{N}_0 \text{ is even}\} ,$$
$$\mathcal{L}^T(q_{\text{odd}}) = \{\mathtt{b}^n(\mathtt{e}) \mid n \in \mathbb{N}_0 \text{ is odd}\} .$$

Now consider the context $\mathtt{b}(\mathtt{b}(y))$. The result of this context starting at q_{odd} is deduced as follows

$$\begin{aligned}
[\![y]\!]^T_{q_{\text{odd}}} &= q_{\text{odd}}(y) \\
[\![\mathtt{b}(y)]\!]^T_{q_{\text{odd}}} &= q_{\text{even}}(\mathtt{B}(x_1)[y]) = q_{\text{even}}(\mathtt{B}(y)) && \text{with } \mathtt{b}(q_{\text{odd}}) \to q_{\text{even}}(\mathtt{B}(x_1)) \\
[\![\mathtt{b}(\mathtt{b}(y))]\!]^T_{q_{\text{odd}}} &= q_{\text{odd}}(\mathtt{B}(x_1)[\mathtt{B}(y)]) = q_{\text{odd}}(\mathtt{B}(\mathtt{B}(y))) && \text{with } \mathtt{b}(q_{\text{even}}) \to q_{\text{odd}}(\mathtt{B}(x_1))
\end{aligned}$$

Thus, the image of this context starting at q_{odd} is $\tau^T_{q_{\text{odd}}}(\mathtt{b}(\mathtt{b}(y))) = \mathtt{Odd}(\mathtt{B}(\mathtt{B}(y)))$. The context is element of $\mathcal{C}^T(q_{\text{odd}})$, but we also get that $\mathtt{b}(\mathtt{b}(y)) \in \mathcal{C}^T(q_{\text{even}})$. The sets of contexts of the two states are the same:

$$\mathcal{C}^T(q_{\text{even}}) = \{\mathtt{b}^n(y) \mid n \in \mathbb{N}_0\} = \mathcal{C}^T(q_{\text{odd}})$$ ◁

4.1 Trim Transducers

Transducers may contain useless transitions or states that we want to eliminate, while preserving the described transformation. A state q of a DBTT is *reachable* if the language $\mathcal{L}^T(q)$ is non-empty. A state q is *meaningful* if q has at least one context, i.e., $\mathcal{C}_T(q)$ is non-empty. Furthermore, the output at state q is *potentially useful* if there is a context c of q such that the image $\tau_q^T(c)$ contains the variable y. Otherwise, the output at q is called *useless*.

Definition 4.2 (Trim)**.** A bottom-up tree transducer T is called *trim* if T has the following properties:

1. Every state is reachable.
2. Every state is meaningful.
3. If the output at a state q is useless, then for each transition of the form $\mathsf{a}(q_1, \ldots, q_m) \to q(z)$ leading into state q, $z = *$.

In this definition, $*$ is a special output symbol, which does not occur in any image produced by T. It is easy to show that each deterministic bottom-up tree transducer is equivalent to a trim DBTT.

Proposition 4.1. *For every deterministic bottom-up tree transducer T a deterministic bottom-up tree transducer T' can be constructed in polynomial time with the following properties:*

1. *T' is equivalent to T*
2. *$|T'| \leq |T|$*
3. *T' is trim.*

Proof. For convenience, we recall the construction. Let $T = (Q, \Sigma, \Delta, R, F)$ be a DBTT. First, we compute the set Reach of reachable states of T as the least solution of the following inequation over the complete lattice of subsets of Q:

$$\mathsf{Reach} \supseteq \{q \in Q \mid \exists\, m \in \mathbb{N}_0, \mathsf{a} \in \Sigma^{(m)}, q_1, \ldots, q_m \in \mathsf{Reach}, z \in \mathcal{T}_\Delta(\mathcal{X}_m) : R(\mathsf{a}, q_1 \ldots q_m) = q(z)\}$$

Since the set Q is finite, the least solution can be computed by standard fixpoint iteration in polynomial time. Analogously, we define the set Mean of reachable and meaningful states of T as the least solution of the following inequations over the complete lattice of subsets of Reach:

$$\begin{aligned} \mathsf{Mean} &\supseteq \{q \in \mathsf{Reach} \mid F(q) \text{ is defined}\} \\ \mathsf{Mean} &\supseteq \{q_1, \ldots, q_m \in \mathsf{Reach} \mid \exists\, m \in \mathbb{N}, \mathsf{a} \in \Sigma^{(m)}, q \in \mathsf{Mean}, z \in \mathcal{T}_\Delta(\mathcal{X}_m) : R(\mathsf{a}, q_1 \ldots q_m) = q(z)\} \end{aligned}$$

Again, since the set Reach is finite, the least solution can be computed by standard fixpoint iteration in polynomial time.

Additionally, we define the set $\mathsf{Use} \subseteq \mathsf{Mean}$ of reachable and meaningful states of T for which output is potentially useful as the least solution of the following inequations:

$$\begin{array}{lcl} \mathsf{Use} & \supseteq & \{\, q \in \mathsf{Mean} \mid F(q) \in \hat{\mathcal{T}}_\Delta(y)\} \\ \mathsf{Use} & \supseteq & \{q_i \in \mathsf{Mean} \mid \exists\, m, i \in \mathbb{N}, \mathsf{a} \in \Sigma^{(m)}, q_1, \ldots, q_m \in \mathsf{Mean}, q \in \mathsf{Use}, \\ & & \qquad z \in \mathcal{T}_\Delta(\mathcal{X}_m) :\ R(\mathsf{a}, q_1 \ldots q_m) = q(z) \text{ and } x_i \text{ occurs in } z\} \end{array}$$

The DBTT $T' = (Q', \Sigma, \Delta, R', F')$ to be then constructed is given by:

- $Q' = \mathsf{Mean}$
- $F' = F|_{Q'}$
- $R'(\mathsf{a}, q_1 \ldots q_m) = q(s')$ iff $R(\mathsf{a}, q_1 \ldots q_m) = q(s)$ and $q_1, \ldots, q_m, q \in Q'$ where $s' = *$ if $q \notin \mathsf{Use}$ and $s' = s$ otherwise.

By construction, $|T'| \leq |T|$. The proofs that T' is trim and equivalent to T are omitted. □

In the remainder of this part we consider trim transducers only. For a trim transducer T with set Q of states, we denote by Q_* the set of states with useless output, i.e., for which the output is always $*$.

The transducer of Example 4.2 illustrated in Figure 4.4 is already trim. Both states are reachable and meaningful (cf. Example 4.4). Since every possible subtree is a subtree of an input tree with an odd number of b-labeled nodes, its output is potentially useful.

Chapter 5

Partition

A deterministic bottom-up tree transducer describes a partial tree transformation. In general, this transformation is not subtree-closed, i.e., there may be a tree, which is not in the domain of the transformation τ^T, but it is a subtree of some other tree in the domain of τ^T. However, the transducer produces some output for such a subtree (but no image). For instance, a DBTT translating sentences in passive voice into their counterparts in active voice only accepts input trees with `Sentence passive` as root label. However, as Figure 3.1 illustrates, such a transducer produces some output for a subtree with root label `Object`. In Figure 4.1 some more output trees of input subtrees are given. Since a deterministic bottom-up tree transducer produces exactly one output tree for every input subtree, we get a function, which produces an output for every subtree of input trees. To get a characterization of the whole transformation, it remains to map the context of the subtree, too (Figure 3.1). We will show that this mapping is also implicitly given by a DBTT (Section 5.1). Motivated by these induced functions of DBTTs, we consider general tree functions and define *partitions* of such functions. A partition consists of a subtree-closed function and mappings of contexts. In the next section, we will show that every DBTT induces such a partition in a natural way. Moreover, every tree function can be defined by a partition.

In this chapter, we consider such function partitions independent of transducers. A partition of a function τ consists of a *subtree function* defined on all subtrees of input trees of τ and a *residual function*, which comprises the remaining information about possible context of a subtree. By that, we get a function on subtrees of input trees without losing information about the whole function. Such a pair of a subtree function and a residual function has to fulfill some properties to describe a tree transformation. The two partial functions of a partition have the same subtree-closed domain. A residual function maps trees to context functions. These context functions are partial functions from contexts to trees with arbitrarily many occurrences of the variable y. For instance, a context function φ maps, amongst others, the context of the `Object`-subtree in Figure 3.1 to the tree $\texttt{Sentence active}(y, v_{\texttt{VP}}, v_{\texttt{Obj}})$ on the right-hand side of this

figure (where we abbreviated the second and the third subtree by $v_{\texttt{VP}}$ and $v_{\texttt{Obj}}$, respectively). The residual function then maps the input subtree `Object`$(\ldots)$ to this context function φ.

Formally, a *context function* φ is a partial function from contexts (over Σ) to trees (over Δ), possibly containing the variable y: $\varphi : \mathcal{C}_\Sigma(y) \dashrightarrow \mathcal{T}_\Delta(y)$. A *residual function* r is, in addition, a partial function, which maps trees to context functions:

$$r : \mathcal{T}_\Sigma \dashrightarrow \mathcal{C}_\Sigma(y) \dashrightarrow \mathcal{T}_\Delta(y) \; .$$

The context function $r(u)$ of a tree $u \in \mathcal{T}_\Sigma$ is called *residual* of u. A residual function defines an equivalence relation.

Definition 5.1 (r-Equivalence)**.** Assume r is a residual function. Two trees $u, u' \in \mathrm{dom}(r)$ are r-equivalent if they have the same residual:

$$u \equiv_r u' \quad \text{iff} \quad r(u) = r(u') \; .$$

For a tree $u \in \mathrm{dom}(r)$, we write $[u]_r$ for the equivalence class of u w.r.t. $\equiv_r$, i.e., $[u]_r = \{u' \mid u \equiv_r u',\ u' \in \mathrm{dom}(r)\}$.

With these definitions, we get the definition of a function partition, i.e., a pair of a subtree function and a residual function:

Definition 5.2 (Partition)**.** Let (σ, r) be a pair of partial functions

$$\sigma : \mathcal{T}_\Sigma \dashrightarrow \mathcal{T}_\Delta \qquad \text{and} \qquad r : \mathcal{T}_\Sigma \dashrightarrow \mathcal{C}_\Sigma(y) \dashrightarrow \mathcal{T}_\Delta(y) \; .$$

Then the pair (σ, r) is a *function partition* (or *partition*) if

- the functions have the same domain, i.e., $\mathrm{dom}(\sigma) = \mathrm{dom}(r)$,
- the domain is subtree-closed, and
- the pair is *consistent*, i.e., for all trees $u, u' \in \mathrm{dom}(\sigma)$ and all contexts $c, c' \in \mathcal{C}_\Sigma(y)$ with $c \cdot u = c' \cdot u'$ holds: If $c \in \mathrm{dom}(r(u))$ then $c' \in \mathrm{dom}(r(u'))$ and

$$r(u)(c) \cdot \sigma(u) = r(u')(c') \cdot \sigma(u') \; . \tag{5.1}$$

The *domain* of a partition (σ, r) is the domain of σ and r. The *tree function defined by the partition* (σ, r) is the partial function $\tau : \mathcal{T}_\Sigma \dashrightarrow \mathcal{T}_\Delta$ given by

$$\tau(u) = \begin{cases} r(u)(y) \cdot \sigma(u) & u \in \mathrm{dom}(\sigma),\ y \in \mathrm{dom}(r(u)) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

On the other hand, we say that (σ, r) is a *partition of the tree function* τ and σ is a *subtree function* of τ. Two partitions are *equivalent* if they define the same tree function. The *index of a partition* is the index of the r-equivalence, i.e., the number of equivalence classes. By Equation 5.1, we get a more general property of the tree function τ of a partition (σ, r):

Lemma 5.1. *Let (σ, r) be a partition of tree transformation τ. For a context $c \in \mathcal{C}_\Sigma(y)$ and a tree $u \in \mathcal{T}_\Sigma$ with $c \cdot u \in dom(\tau)$ holds*

$$\tau(c \cdot u) = r(u)(c) \cdot \sigma(u) \ .$$

Proof. Let $c \in \mathcal{C}_\Sigma(y)$, $u \in \mathcal{T}_\Sigma$, and $c \cdot u \in \mathrm{dom}(\tau)$. By definition of τ we get

$$\tau(c \cdot u) = r(c \cdot u)(y) \cdot \sigma(c \cdot u) \ .$$

Furthermore, since the partition is consistent and $y \cdot c \cdot u = c \cdot u$:

$$r(c \cdot u)(y) \cdot \sigma(c \cdot u) = r(u)(c) \cdot \sigma(u)$$

Thus, it follows $\tau(c \cdot u) = r(u)(c) \cdot \sigma(u)$. □

In general, there are more than one partition for a tree function τ, and for a subtree function σ there are different residual functions r such that (σ, r) is a partition of τ. In particular, for every tree function τ, the *canonical partition* (σ_*, r_*) exists:

$$\sigma_*(u) = \begin{cases} * & \text{if } u \in \mathtt{Subtrees}(\mathrm{dom}(\tau)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$r_*(u)(c) = \begin{cases} \tau(c \cdot u) & \text{if } c \cdot u \in \mathrm{dom}(\tau) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Here, $*$ is an arbitrary symbol of the output alphabet. Since the image of every residual is in $\mathcal{T}_\Delta$ (the trees do not contain the variable y), the output of σ_* is never used in τ. In general, the r_*-equivalence of this partition has infinitely many equivalence classes, i.e., its index of $\equiv_{r_*}$ is infinite.

As already said, every deterministic bottom-up tree transducer defines a partition. This will be defined more precisely in the next section. Often, this partition is more significant then the canonical partition.

Example 5.1. Consider again the transducer $T = (Q, \Sigma, \Delta, R, F)$ of Example 4.2, which defines the transformation

$$\tau : \mathtt{b}^n(\mathtt{e}) \mapsto \begin{cases} \mathtt{Odd}(\mathtt{B}^n(\mathtt{E})) & \text{for all odd } n \in \mathbb{N}_0 \\ \mathtt{Even} & \text{for all even } n \in \mathbb{N}_0 \end{cases}$$

At each state, the transducer produces an output tree for every input subtree, which reaches this state. Thus, the states of T describe functions:

$$\begin{aligned} \mathtt{b}^n(\mathtt{e}) &\mapsto \mathtt{B}^n(\mathtt{E}) \quad \text{if } \mathtt{b}^n(\mathtt{e}) \in \mathcal{L}^T(q_{\mathrm{even}}) \\ \mathtt{b}^n(\mathtt{e}) &\mapsto \mathtt{B}^n(\mathtt{E}) \quad \text{if } \mathtt{b}^n(\mathtt{e}) \in \mathcal{L}^T(q_{\mathrm{odd}}) \end{aligned}$$

Since the state reached by a tree is unambiguous and the transducer is deterministic, we get the subtree-closed function

$$\mathtt{out}^T : \mathtt{b}^n(\mathtt{e}) \mapsto \mathtt{B}^n(\mathtt{E}) \quad \text{for all } n \in \mathbb{N}_0 \ .$$

This function, however, differs from the given transformation $\tau^T = \tau$. To get the remaining parts of the output trees (or actually the proper output), we have to consider another function, which is given by every state. It is the context function, which maps every possible context of a state to an image:

$$\tau^T_{q_{\text{even}}} : \mathtt{b}^k(y) \mapsto \begin{cases} \mathtt{Odd}(\mathtt{B}^k(y)) & \text{for all odd } k \in \mathbb{N}_0 \\ \mathtt{Even} & \text{for all even } k \in \mathbb{N}_0 \end{cases}$$

$$\tau^T_{q_{\text{odd}}} : \mathtt{b}^k(y) \mapsto \begin{cases} \mathtt{Even} & \text{for all odd } k \in \mathbb{N}_0 \\ \mathtt{Odd}(\mathtt{B}^k(y)) & \text{for all even } k \in \mathbb{N}_0 \end{cases}$$

Since every subtree of an input tree reaches exactly one state of the DBTT, we can define a function r^T, which maps every input subtree to such a function, i.e., we get

$$r^T(\mathtt{b}^n(\mathtt{e}))(\mathtt{b}^k(y)) = \begin{cases} \mathtt{Odd}(\mathtt{B}^k(y)) & \text{for all } n,k \in \mathbb{N}_0 \text{ with } n+k \text{ is odd.} \\ \mathtt{Even} & \text{for all } n,k \in \mathbb{N}_0 \text{ with } n+k \text{ is even.} \end{cases}$$

For the two functions out^T and r^T, we observe the following properties with respect to the function τ^T:

1. They have the same domain, $\mathrm{dom}(\mathsf{out}^T) = \mathrm{dom}(r^T) = \{\mathtt{b}^n(\mathtt{e}) \mid n \in \mathbb{N}_0\}$, which is subtree-closed and contains exactly the subtrees of $\mathrm{dom}(\tau^T)$.
2. Together, they describe τ^T: For every tree $\mathtt{b}^n(\mathtt{e})$ and every decomposition of the form $\mathtt{b}^{n_1}(y) \cdot \mathtt{b}^{n_2}(\mathtt{e})$, it holds:

$$\tau^T(\mathtt{b}^n(\mathtt{e})) = r^T(\mathtt{b}^{n_2}(\mathtt{e}))(\mathtt{b}^{n_1}(y)) \cdot \mathsf{out}^T(\mathtt{b}^{n_2}(\mathtt{e}))$$

Thus, (out^T, r^T) is a partition of τ^T. ◁

In the following, let (σ, r) be a partition with input alphabet Σ and output alphabet Δ and the tree function of (σ, r) is denoted τ. We designate a special output symbol $* \in \Delta$, which does not occur in any image of τ. Thus, $*$ may only occur in the output of σ.

5.1 Partitions of DBTTs

The pair (out^T, r^T) in Example 5.1 is a partition of the transformation τ^T defined by the DBTT T of Figure 4.4. As for this transducer, we will now define a partition for every DBTT.

Every transducer produces some output for every subtree of an input tree. A transducer T provides the subtree-closed function $\mathsf{out}^T : \mathcal{T}_\Sigma \dashrightarrow \mathcal{T}_\Delta$. Additionally, for the remaining part of every input tree of the transformation, we get an image. It is given by the partial function τ^T_q depending on the state q, which is reached by the subtree. More formally, let u be a subtree of an input tree u' of the transformation τ^T of DBTT T, i.e., there is a context $c \in \mathcal{C}_\Sigma(y)$ with $u' = c \cdot u$. Then the state q reached by u induces the function τ^T_q, which

provides an image $\tau_q^T(c)$ of the context. In general, a transducer T defines a partial function

$$r^T : \mathcal{T}_\Sigma \dashrightarrow \mathcal{C}_\Sigma(y) \dashrightarrow \mathcal{T}_\Delta(y)$$

which assigns an image to every context depending on a subtree. Precisely, for a tree u, which reaches state q and a context $c \in \mathcal{C}^T(q)$, the function r^T is defined by

$$r^T(u)(c) = \tau_q^T(c) \ .$$

As for the example transducer, we observe for the two functions out^T and r^T of any transducer T the following properties:

Lemma 5.2. *Let T be a DBTT. Then for the functions out^T and r^T hold:*

1. *They have the same subtree-closed domain, which is the set of subtrees of all input trees of T, i.e.,*

$$dom(\mathsf{out}^T) = dom(r^T) = \mathtt{Subtrees}(dom(\tau^T)) \ .$$

2. *They are consistent and describe τ^T: For every tree $u' \in dom(\tau^T)$ and every decomposition of the form $u' = c \cdot u$ with $u \in \mathcal{T}_\Sigma$ and $c \in \mathcal{C}_\Sigma(y)$ holds:*

$$\tau^T(u') = r^T(u)(c) \cdot \mathsf{out}^T(u)$$

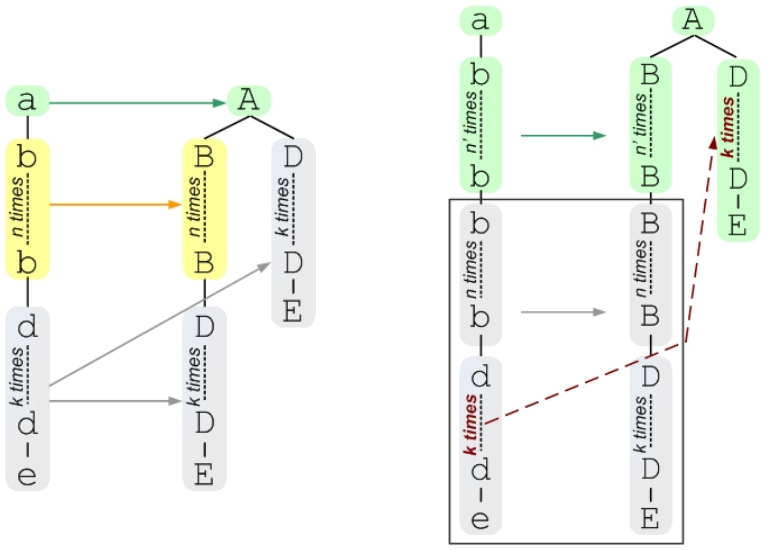
With that, we conclude

Corollary 5.3. *The pair (out^T, r^T) of a transducer T is a partition of τ^T.*

For a DBTT T, we call the pair (out^T, r^T) the *partition of T*. All partitions of DBTTs have some properties in common. In particular, we will observe that their equivalence relations are congruences of finite index and that they are path-finite. The last property describes a restriction of the form of the images of similar contexts. It will be defined in Section 5.1.2. Before that, we consider the index of the equivalence relation $\equiv_{r^T}$ for a DBTT T.

5.1.1 Finite Index

Let T be a DBTT with partition (out^T, r^T). If we consider two trees, which reach the same state in T, we observe that they have the same residual. Continuing the translation at a state, the transducer does not distinguish with which tree it reaches this state. Thus, two trees, which are in the language of the same state of T are r^T-equivalent. It follows that the number of equivalence classes of $\equiv_{r^T}$ is bounded by the number of states of T. Consider for instance the DBTT T in Example 4.2. All trees with an even number of $\mathtt{b}$-labeled nodes reach the state q_{even}. All these trees in $\mathcal{L}^T(q_{\text{even}})$ have the same residual (cf. the partition in Example 5.1). Both states define different residuals. Thus, this partition has index 2. Since a deterministic bottom-up tree transducer has a finite number of states, every partition of a DBTT has finite index.



(a) The function: Which part of the output is a copy of which part of the input.

(b) Subtree $\mathtt{b}^n(\mathtt{d}^k(\mathtt{e}))$ with context $\mathtt{a}(\mathtt{b}^{n'}(y))$ and there images.

Figure 5.1: The function τ of Example 5.2.

Lemma 5.4. *Let $T = (Q, \Sigma, \Delta, R, F)$ be a DBTT and (out^T, r^T) its partition. Then $|\equiv_{r^T}| \leq |Q|$.*

Proof. Let u and u' be two trees, which reach the same state q in T, i.e., $u, u' \in \mathcal{L}^T(q)$. By definition of the residual function r^T, it follows

$$r^T(u) = \tau_q^T = r^T(u') \; .$$

That means $u \equiv_{r^T} u'$. Consequently, $|\equiv_{r^T}| \leq |Q|$. □

Thus, if we want to get a partition, which defines a DBTT, it is a necessary property that the r-equivalence relation of the partition has finite index. The following example presents a tree transformation, which has no partition with equivalence relation of finite index. Hence, this transformation is not definable by a DBTT.

Example 5.2. Let us consider the tree function (Figure 5.1(a))

$$\tau : \mathtt{a}(\mathtt{b}^n(\mathtt{d}^k(\mathtt{e}))) \mapsto \mathtt{A}(\mathtt{B}^n(\mathtt{D}^k(\mathtt{E})), \mathtt{D}^k(\mathtt{E})) \quad \text{for all } n, k \geq 0 \; . \tag{5.2}$$

The input tree has two variable parts. Under the root labeled $\mathtt{a}$, the tree has first a chain of $\mathtt{b}$-labeled nodes, followed by a chain of nodes with label $\mathtt{d}$. The leaf has label $\mathtt{e}$. The transformation copies the whole input tree once (by replacing small letters by capital letters). Additionally, the root of the output tree has

a further subtree, which is a copy of the chain of $\mathtt{d}$'s (Figure 5.1(a)). We can specify the following partition (σ, r) for this transformation (Figure 5.1(b)):

$$\sigma: \begin{cases} \mathtt{b}^n(\mathtt{d}^k(\mathtt{e})) \mapsto \mathtt{B}^n(\mathtt{D}^k(\mathtt{E})) \\ \mathtt{a}(\mathtt{b}^n(\mathtt{d}^k(\mathtt{e}))) \mapsto \mathtt{A}(\mathtt{B}^n(\mathtt{D}^k(\mathtt{E})), \mathtt{D}^k(\mathtt{E})) \end{cases} \quad \text{for all } n, k \geq 0$$

$$r(\mathtt{d}^k(\mathtt{e})): \ \mathtt{a}(\mathtt{b}^n(\mathtt{d}^l(y))) \mapsto \mathtt{A}(\mathtt{B}^n(\mathtt{D}^l(y)), \mathtt{D}^l(y)) \quad \text{for all } n, k, l \geq 0$$

$$r(\mathtt{b}^n(\mathtt{d}^k(\mathtt{e}))): \ \mathtt{a}(\mathtt{b}^m(y)) \mapsto \mathtt{A}(\mathtt{B}^m(y), \mathtt{D}^k(\mathtt{E})) \quad \text{for all } n, k, m \geq 0$$

$$r(\mathtt{a}(\mathtt{b}^n(\mathtt{d}^k(\mathtt{e})))): \ y \mapsto y \quad \text{for all } n, k \geq 0$$

However, this partition has infinitely many equivalence classes. For every $k \neq k'$ the trees $\mathtt{b}^n(\mathtt{d}^k(\mathtt{e}))$ and $\mathtt{b}^n(\mathtt{d}^{k'}(\mathtt{e}))$ have different residuals (Figure 5.1(b)).

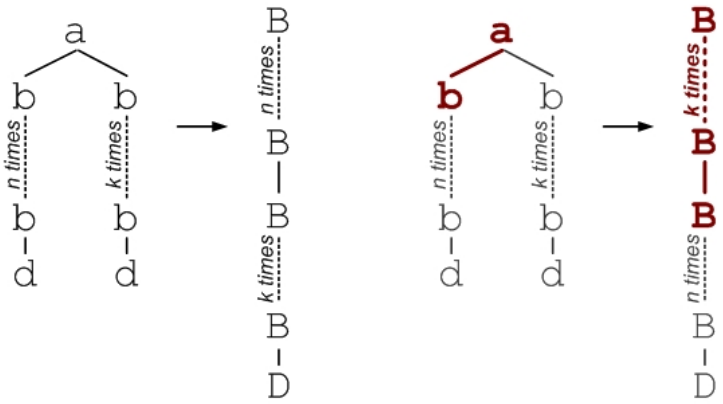
Unfortunately, one can prove that there is no partition with equivalence relation of finite index that describes this transformation. The idea is the following: Every partition can only produce one output tree for an input tree $\mathtt{b}^n(\mathtt{d}^k(\mathtt{e}))$ (for fixed $n, k \geq 1$). However, for the context $\mathtt{a}(y)$, we need two different subtrees dependent of the input tree. Hence, the residual has to produce one of these subtrees. More precisely, the residual has to produce the part of the left subtree, which contains all n $\mathtt{B}$-labeled nodes, or the whole right subtree (with all k $\mathtt{D}$-labeled nodes). For arbitrary n and k, there are infinitely many different such output subtrees, both for the left and the right child of the root. Hence, there are infinitely many different residuals. ◁

This example shows that not every function has a partition with finite index, but we have observed that this is a necessary property to be a function, which is definable by a DBTT. Thus, there is no deterministic bottom-up tree transducer, which defines the tree function τ of Example 5.2.

5.1.2 Path-Finite Partition

Another necessary property is path-finiteness. The idea is the following. A deterministic bottom-up tree transducer applies one rule to every node of an input tree. Thus, for every path p to some node of an input tree, we get a sequence of rules. Although p may belong to infinitely many trees, there are only finitely many different such sequences of rules for p. This property restricts the form of images according to contexts in which p refers to the y node. We will introduce this property as *path-finiteness*, which is a necessary property of a partition to be a partition of a DBTT. Consider for instance the partition illustrated in Figure 3.1. The path $p = (\mathtt{Sentence\ passive}, 4)$ belongs to infinitely many syntax trees of sentences in passive voice. Even though, the images of those contexts with p as y-path only have the y-path $(\mathtt{Sentence\ active}, 1)$.

We give an example, which shows that not every partition is path-finite, nor if its equivalence relation has finite index. Furthermore, path-finiteness is a proper restriction in reference to partitions with equivalence relations of finite index. We will give an example for a tree transformation, which is not definable by a path-finite partition with equivalence relation of finite index.



(a) The transformation τ. (b) Path $(\mathtt{a}, 1)$ induces $(\mathtt{B}, 1)^k$.

Figure 5.2: The transformation of Example 5.3, which is not path-finite.

Let (σ, r) be an arbitrary partition. For an input tree u of σ and a path p, let $C_{u,p}$ be the set of contexts in the domain of $r(u)$ in which p refers to the y-labeled node. The partition is path-finite if, for every pair (u, p) of an input tree and a path, the number of y-paths in images of $C_{u,p}$ under $r(u)$ is finite.

Definition 5.3 (Path-finite)**.** A partition (σ, r) is called *path-finite* if for all $u \in \mathrm{dom}(\sigma)$ and each path $p \in \mathtt{Paths}_\Sigma$ holds

$$\Big|\bigcup\{\mathtt{Paths}_y(r(u)(c)) \mid c \in \mathrm{dom}(r(u)) \wedge p \in \mathtt{Paths}_y(c)\}\Big| < \infty\,. \tag{5.3}$$

That means, for each y-path in input contexts there are only finitely many paths in corresponding output contexts.

Example 5.3. Consider the tree function

$$\tau : \mathtt{a}(\mathtt{b}^n(\mathtt{d}), \mathtt{b}^k(\mathtt{d})) \mapsto \mathtt{B}^{n+k}(\mathtt{D}) \quad \text{for all } n, k \geq 0 \tag{5.4}$$

which maps trees with two subtrees of depth $n+1$ and $k+1$, respectively, to one tree of depth $n + k + 1$ (Figure 5.2(a)). We will show that this transformation has no path-finite partition with finite index (Lemma 5.5). Before that, let us consider some partitions for τ. The canonical partition (σ_*, r_*) (cf. Page 39) is path-finite because the images of contexts in the residuals do not contain any y-paths. However, this partition has infinite index. On the other hand, we can specify the following partition (σ, r) with the domain $\mathtt{Subtrees}(\mathrm{dom}(\tau))$:

$$\sigma = \begin{cases} \mathtt{b}^n(\mathtt{d}) \mapsto \mathtt{B}^n(\mathtt{D}) \\ \mathtt{a}(\mathtt{b}^n(\mathtt{d}), \mathtt{b}^k(\mathtt{d})) \mapsto \mathtt{B}^{n+k}(\mathtt{D}) \end{cases} \qquad \text{for all } n, k \geq 0$$

$$r(\mathtt{b}^n(\mathtt{d})) = \begin{cases} \mathtt{a}(\mathtt{b}^m(y), \mathtt{b}^k(\mathtt{d})) \mapsto \mathtt{B}^{m+k}(y) \\ \mathtt{a}(\mathtt{b}^m(\mathtt{d}), \mathtt{b}^k(y)) \mapsto \mathtt{B}^{m+k}(y) \end{cases} \qquad \text{for all } n,k,m \geq 0$$

$$r(\mathtt{a}(\mathtt{b}^n(\mathtt{d}), \mathtt{b}^k(\mathtt{d}))) = y \mapsto y\ . \qquad \text{for all } n,k \geq 0$$

In contrast to the canonical partition, this partition has finite index. It has two equivalence classes: The one of all input trees of τ and the one of all proper subtrees, i.e.,

$$\{\mathtt{a}(\mathtt{b}^n(\mathtt{d}), \mathtt{b}^k(\mathtt{d})) \mid n,k \geq 0\} \qquad \text{and} \qquad \{\mathtt{b}^n(\mathtt{d}) \mid n \geq 0\}\ .$$

However, the partition (σ, r) is not path-finite: For any tree of the form $\mathtt{b}^n(\mathtt{d})$ and the path $p = (\mathtt{a}, 1)$ we get infinitely many y-paths in the images of all contexts containing p as y-path under $r(\mathtt{b}^n(\mathtt{d}))$ (cf. Figure 5.2(b)). More precisely, let $u_n = \mathtt{b}^n(\mathtt{d})$. Then, we get $C_{u_n,p} = \{\mathtt{a}(y, \mathtt{b}^k(\mathtt{d})) \mid k \geq 0\}$. For the set in Definition 5.3, it follows

$$\begin{aligned} \mathtt{Paths}_y(r(u_n)(C_{u_n,p})) &= \bigcup \{\mathtt{Paths}_y(\mathtt{B}^k(y)) \mid k \geq 0\} \\ &= \{(\mathtt{B}, 1)^k \mid k \geq 0\}\ , \end{aligned}$$

which is an infinite set. As we prove in the following lemma, for this tree function there is no partition, which combines both properties, i.e., a path-finite partition with an equivalence relation of finite index. ◁

The transformation in the Example 5.3 is not definable by a DBTT. The following lemma shows that every partition for this transformation is not path-finite or has an equivalence of infinite index.

Lemma 5.5. *There is a tree function τ, which is not definable by a path-finite partition with finite index.*

Proof. Consider the function τ of Example 5.3 and let (σ, r) be a partition of τ. We show that if (σ, r) has finitely many r-equivalence classes, the partition is not path-finite. For every subtree $\mathtt{b}^n(\mathtt{d}) \in \mathrm{dom}(\sigma)$ and every context $\mathtt{a}(\mathtt{b}^m(y), \mathtt{b}^k(\mathtt{d})) \in \mathrm{dom}(r(\mathtt{b}^n(\mathtt{d})))$ it holds by Lemma 5.1

$$\begin{aligned} & r(\mathtt{b}^n(\mathtt{d}))(\mathtt{a}(\mathtt{b}^m(y), \mathtt{b}^k(\mathtt{d}))) \cdot \sigma(\mathtt{b}^n(\mathtt{d})) \\ &= \tau(\mathtt{a}(\mathtt{b}^{m+n}\mathtt{b}^k(\mathtt{d}))) \\ &= \mathtt{B}^{n+m+k}(\mathtt{D})\ . \end{aligned}$$

Assume $\equiv_r$ has finite index. Then there are infinitely many input trees of the form $\mathtt{b}^n(\mathtt{d})$ with the same residual. Let $\mathtt{b}^n(\mathtt{d})$ and $\mathtt{b}^m(\mathtt{d})$ be two trees with the same residual (for some $n \neq m$). For every $k \geq 0$, there are two possibilities for the residual:

$$r(\mathtt{b}^n(\mathtt{d}))(\mathtt{a}(y, \mathtt{b}^k(\mathtt{d}))) = \mathtt{B}^{k+n}(\mathtt{D}) \tag{5.5}$$

$$r(\mathtt{b}^n(\mathtt{d}))(\mathtt{a}(y, \mathtt{b}^k(\mathtt{d}))) = \mathtt{B}^{i_k}(y) \qquad \text{for some } i_k \geq 0 \tag{5.6}$$

First, assume that for some k the image is ground (Equation 5.5). Then, we get for the output trees

$$\begin{aligned}\mathtt{B}^{k+m}(\mathtt{D}) &= \tau(\mathtt{a}(\mathtt{b}^m(\mathtt{d}), \mathtt{b}^k(\mathtt{d}))) \\ &= r(\mathtt{b}^m(\mathtt{d}))(\mathtt{a}(y, \mathtt{b}^k(\mathtt{d}))) \cdot \sigma(\mathtt{b}^m(\mathtt{d})) \\ &= r(\mathtt{b}^n(\mathtt{d}))(\mathtt{a}(y, \mathtt{b}^k(\mathtt{d}))) \cdot \sigma(\mathtt{b}^m(\mathtt{d})) \\ &= \mathtt{B}^{k+n}(\mathtt{D}) \cdot \sigma(\mathtt{b}^m(\mathtt{d})) \\ &= \mathtt{B}^{k+n}(\mathtt{D}) \ ,\end{aligned}$$

which is not possible because $n \neq m$. Thus, for every k, the residual is defined by Equation 5.6. With that, it follows

$$\begin{aligned}\mathtt{B}^{i_k}(y) \cdot \mathtt{B}^{k+m-i_k}(\mathtt{D}) &= \mathtt{B}^{k+m}(\mathtt{D}) \\ &= \tau(\mathtt{a}(\mathtt{b}^m(\mathtt{d}), \mathtt{b}^k(\mathtt{d}))) \\ &= r(\mathtt{b}^m(\mathtt{d}))(\mathtt{a}(y, \mathtt{b}^k(\mathtt{d}))) \cdot \sigma(\mathtt{b}^m(\mathtt{d})) \\ &= r(\mathtt{b}^n(\mathtt{d}))(\mathtt{a}(y, \mathtt{b}^k(\mathtt{d}))) \cdot \sigma(\mathtt{b}^m(\mathtt{d})) \\ &= \mathtt{B}^{i_k}(y) \cdot \sigma(\mathtt{b}^m(\mathtt{d}))\end{aligned}$$

and with that, $\sigma(\mathtt{b}^m(\mathtt{d})) = \mathtt{B}^{k+m-i_k}(\mathtt{D})$. Since k is arbitrary, but $k+m-i_k$ must be the same for all k, i_k must be variable and dependent of k. In particular, the set $\{i_k \mid k \geq 0\}$ is infinite, but then, every image $r(\mathtt{b}^n(\mathtt{d}))(\mathtt{a}(y, \mathtt{b}^k(\mathtt{d}))) = \mathtt{B}^{i_k}(y)$ has another y-paths. More precisely, $(\mathtt{B}, 1)^{i_k} \in \mathtt{Paths}_y(r(\mathtt{b}^n(\mathtt{d}))(C_{\mathtt{b}^n(\mathtt{d}),(\mathtt{a},1)}))$ for all i_k. Thus, (σ, r) is not path-finite. □

It remains to prove that path-finiteness is a necessary property of a partition to be a partition of a deterministic bottom-up tree transducer, i.e., every partition of a DBTT is path-finite:

Theorem 5.6. *The partition* $(\mathtt{out}^T, r^T)$ *of a deterministic bottom-up tree transducer* T *is path-finite.*

Proof. The idea is given in the beginning of this section: For a node in an input tree the transducer applies one rule. There are finitely many possibilities for each node depending on its subtrees. Thus, for a finite y-path in the input context, there are finitely many different y-paths in its image.

Let $T = (Q, \Sigma, \Delta, R, F)$, $u \in \mathrm{dom}(r^T)$ and $c_u \in \mathrm{dom}(r^T(u))$. We consider the path $p \in \mathtt{Paths}_y(c_u)$ and show by induction over the length of p that there are only finitely many y-paths in the image of the residual of u such that p is a y-path in the context, i.e., that Equation 5.3 holds.

First, assume $p = \epsilon$. Thus, $c_u = y$ and there is no other context $c' \in \mathcal{C}_\Sigma(y)$ with $p \in \mathtt{Paths}_y(c')$. The image $r^T(u)(c_u)$ is defined by the final function F of T, i.e., $r^T(u)(c_u) = F(q)$ for some state $q \in Q$. Thus, the image has only finitely many different paths. Consequently, the set $\mathtt{Paths}_y(r^T(u)(c_u))$ of all y-paths in the image is finite.

Now assume $p = p_1.(\mathtt{a}, i)$ for some label $\mathtt{a} \in \Sigma$ of rank k and $i \leq k$. We have to prove that the set

$$\bigcup\{\mathtt{Paths}_y(r^T(u)(c)) \mid c \in \mathrm{dom}(r^T(u)) \wedge p \in \mathtt{Paths}_y(c)\}$$

is finite. Consider a context $c \in \mathcal{C}_\Sigma(y)$ with $p \in \mathtt{Paths}_y(c)$, then the context c can be written as $c = c_1 \cdot c_\mathtt{a}$ with $p_1 \in \mathtt{Paths}_y(c_1)$ and $(\mathtt{a}, i) \in \mathtt{Paths}_y(c_\mathtt{a})$. If $c \in \mathrm{dom}(r^T(u))$, it follows by the definition of r^T (cf. Page 41) that u reaches a state $q_u \in Q$ and $\tau^T_{q_u} = r^T(u)$. Thus, $c \in \mathcal{C}^T(q_u)$ and $r^T(u)(c) = \tau^T_{q_u}(c)$. The transducer processes the context bottom-up. Thus, there is a state $q_\mathtt{a}$ and some image $z_\mathtt{a}$ with $[\![c_\mathtt{a}]\!]^T_{q_u} = q_\mathtt{a}(z_\mathtt{a})$. The remaining context is also translated to some image: $\tau^T_{q_\mathtt{a}}(c_1) = z_1$. These two images build the image of the context c, i.e., $\tau^T_{q_u}(c) = z_1 \cdot z_\mathtt{a}$. Consequently,

$$r^T(u)(c) = z_1 \cdot z_\mathtt{a} \,.$$

Furthermore, the tree $c_\mathtt{a} \cdot u$ reaches state $q_\mathtt{a}$ with $c_1 \in \mathcal{C}^T(q_\mathtt{a})$. Thus, we have $c_\mathtt{a} \cdot u \in \mathrm{dom}(r^T)$ and $c_1 \in \mathrm{dom}(r^T(c_\mathtt{a} \cdot u))$. By induction, the set

$$\bigcup\{\mathtt{Paths}_y(r^T(c_\mathtt{a} \cdot u)(c_1)) \mid c_1 \in \mathrm{dom}(r^T(c_\mathtt{a} \cdot u)) \wedge p_1 \in \mathtt{Paths}_y(c_1)\}$$

is finite for $c_\mathtt{a} \cdot u$ and p_1. It remains to prove that there are only finitely many y-paths in images of $c_\mathtt{a}$. Since $(\mathtt{a}, 1)$ is the y-path in $c_\mathtt{a}$, the context has the form

$$c_\mathtt{a} = \mathtt{a}(u_1, \ldots, u_{i-1}, y, u_{i+1}, \ldots, u_k)$$

for some subtrees $u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_k \in \mathcal{T}_\Sigma$. Let $u_i = u$. By the definition of $r^T(u)$ we get: There are $q_1, \ldots, q_k \in Q$ and output trees $v_1, \ldots, v_k \in \mathcal{T}_\Delta$ with $[\![u_j]\!]^T = q_j(v_j)$ for all $j \in [k]$. In particular, $q_i = q_u$. In addition, the right-hand side of $(\mathtt{a}, q_1 \ldots q_k)$ is $R(\mathtt{a}, q_1 \ldots q_k) = q_\mathtt{a}(z'_\mathtt{a})$ for some $z'_\mathtt{a} \in \mathcal{T}_\Delta(\mathcal{X}_k)$. The image of $c_\mathtt{a}$ is then given by $z_\mathtt{a} = z'_\mathtt{a}[v_1, \ldots, v_{i-1}, y, v_{i+1}, \ldots, v_k]$. Thus, the y-paths in this image are the x_i-paths in $z'_\mathtt{a}$. However, the number of different images $z'_\mathtt{a}$ is bounded by the number of different sequences of states $q_1 \ldots q_k$. Since Q is finite, there are only finitely many sequences. For each of these images, the number of x_i-paths is finite. Thus, the set of y-paths

$$\bigcup\{\mathtt{Paths}_y(z_\mathtt{a}) \mid u \in \mathcal{L}^T(q_u) \wedge [\![c_\mathtt{a}]\!]^T_{q_u} = q_\mathtt{a}(z_\mathtt{a}) \wedge (\mathtt{a}, 1) \in \mathtt{Paths}_y(c_\mathtt{a})\}$$

is finite. Together, we get for $p = p_1 \cdot (\mathtt{a}, 1)$

$$\begin{aligned}
& |\textstyle\bigcup\{\mathtt{Paths}_y(r^T(u)(c)) \mid c \in \mathrm{dom}(r^T(u)) \ \wedge \ p \in \mathtt{Paths}_y(c)\}| \\
\leq \ & |\textstyle\bigcup\{\mathtt{Paths}_y(r^T(c_\mathtt{a} \cdot u)(c_1)) \mid c_1 \in \mathrm{dom}(r^T(c_\mathtt{a} \cdot u)) \\
& \qquad\qquad \wedge\, p_1 \in \mathtt{Paths}_y(c_1) \ \wedge \ (\mathtt{a}, 1) \in \mathtt{Paths}_y(c_\mathtt{a})\}| \\
& \cdot |\textstyle\bigcup\{\mathtt{Paths}_y(z_\mathtt{a}) \mid u \in \mathcal{L}^T(q_u) \ \wedge \ [\![c_\mathtt{a}]\!]^T_{q_u} = q_\mathtt{a}(z_\mathtt{a}) \wedge \ (\mathtt{a}, 1) \in \mathtt{Paths}_y(c_\mathtt{a})\}| \\
< \ & \infty \,.
\end{aligned}$$

□

Summarizing, we get by Lemma 5.4 and Theorem 5.6 that for every DBTT T, its partition $(\mathtt{out}^T, r^T)$ is path-finite and has finite index.

5.2 Bottom-Up Partition

For every transformation, infinitely many different partitions are possible. First of all, there is the canonical partition (cf. Page 39), which realizes the output in the residual and produces $*$ for every subtree in the subtree function. We can define arbitrarily many partitions for the same transformation by only changing the subtree function of the canonical partition. There are also partitions with different residual functions for one tree function. We figured out that for every transformation, which is definable by a DBTT, there exists a partition (the partition of the transducer), which is path-finite and has an equivalence relation of finite index. However, if a transformation has a path-finite partition with finite index, in general, the transformation has more than one partition with these constraints. For instance, we can separate one input tree and define its output under the subtree function and its residual as in the canonical partition.

In this chapter, we specify a unique partition for every transformation. We are interested only in transformations, which are definable by DBTTs. Thus, if there is no path-finite partition with finite index, we fix the canonical partition as normal form. Otherwise, we present a normal form, which is path-finite and has finite index. This normal form is a unique partition. In Section 6.3, we show that this partition is already the partition of a unique DBTT, describing the given transformation. The normal form has six intrinsic properties:

trim Every subtree produces significant output only if an image of some context uses it, i.e., contains a y. Otherwise, it produces $*$.

proper The subtrees in an equivalence class are mapped to infinitely many different output trees. Otherwise, all trees in this class are mapped to $*$.

earliest The output of such an equivalence class with infinitely many different output trees is produced as early as possible, i.e., there is no common suffix in all images of the residual of a subtree. This common part would be produced by the subtree function.

unified earliest The residual of trees in different equivalence classes cannot be unified by ground trees.

congruent The equivalence relation is a congruence relation.

minimal We consider the partition with the smallest index, which is trim, proper, earliest, unified earliest, and path-finite.

We show that for every path-finite partition with finite index, there exists exactly one partition, which fulfills all these properties and describes the same transformation. We will call this unique partition the *bottom-up partition* of the transformation if it exists (cf. Definition 5.10).

5.2.1 Trim Partition

Similar to trim transducers, we only consider *trim* partitions. Meaning, for an input subtree u, σ outputs a tree $v \neq *$ if and only if it is used in some image of a context of u. Otherwise, it maps u to $*$. Furthermore, every subtree u in $\mathrm{dom}(\sigma)$ has a non-empty residual. Consider, for instance, the transformation of sentences in passive voice into sentences in active voice (Figure 1.1). Figure 4.1 shows parts of a possible subtree function, but if the preposition "by" only occurs in the third subtree of the root node, its output is never used. Changing the output of the subtree $\texttt{Preposition}(\ldots \text{by} \ldots)$ does not change the transformation defined by the suggested partition. Thus, there are infinitely many different partitions for this transformation. To get a unique partition, we fix the output for such subtrees to $*$.

We say that the output of $u \in \mathrm{dom}(\sigma)$ is *useful* in a partition (σ, r) if a context of u exists, whose image under $r(u)$ contains the variable y, i.e., $r(u)(\mathcal{C}_\Sigma(y)) \cap \hat{\mathcal{T}}_\Delta(y) \neq \emptyset$. Otherwise, the output is called *useless*. A trim partition restricts the useless outputs of the subtree function.

Definition 5.4 (Trim)**.** A partition (σ, r) is called *trim* if for each $u \in \mathrm{dom}(\sigma)$ holds $\mathrm{dom}(r(u)) \neq \emptyset$ and

$$r(u)(\mathcal{C}_\Sigma(y)) \subseteq \mathcal{T}_\Delta \Leftrightarrow \sigma(u) = * \,.$$

The non-emptiness of the residuals implies that the domain of a trim partition (σ, r) contains exactly the subtrees of input trees of its tree function τ, i.e., $\mathrm{dom}(\sigma) = \texttt{Subtrees}(\mathrm{dom}(\tau))$. The transformations, which are definable by path-finite partitions with finite index are also definable by trim and path-finite partitions with finite index. If we restrict the domain and replace useless output of the subtree function by $*$, we get an equivalent trim and path-finite partition with the same equivalence relation (restricted to the domain).

Theorem 5.7. *For every path-finite partition (σ, r) with finite index exists an equivalent partition (σ', r'), which is*

- *trim,*
- *path-finite, and*
- *it holds $\mid \equiv_{r'} \mid \leq \mid \equiv_r \mid$.*

Proof. Let τ be the tree function of the partition (σ, r). First, let

$$D = \{u \mid u \in \mathrm{dom}(\sigma), \mathrm{dom}(r(u)) \neq \emptyset\} \,.$$

This set equals the set of subtrees of input trees of the transformation τ, i.e., $D = \texttt{Subtrees}(\mathrm{dom}(\tau))$. We define a subtree function σ' on D, which differs from σ (restricted to D) only on useless outputs. It is defined for every $u \in D$ as follows:

$$\sigma'(u) = \begin{cases} * & r(u)(\mathcal{C}_\Sigma(y)) \subseteq \mathcal{T}_\Delta \\ \sigma(u) & \text{otherwise} \end{cases}$$

Also r' is restricted to D. But apart from that, it equals r on this domain. Now (σ', r') is trim and it describes the same tree function τ as (σ, r):

Let $c \cdot u \in \mathrm{dom}(\tau)$. If $r(u)(c) \in \mathcal{T}_\Delta$ is ground then

$$r'(u)(c) \cdot \sigma'(u) = r(u)(c) \cdot \sigma'(u) = r(u)(c) = r(u)(c) \cdot \sigma(u) = \tau(c \cdot u) \ .$$

Otherwise, $\sigma(u)$ is useful and we get

$$r'(u)(c) \cdot \sigma'(u) = r(u)(c) \cdot \sigma(u) = \tau(c \cdot u) \ .$$

Since the residuals on the new domain D do not change, the number of equivalence classes is again finite, more precise, it holds $| \equiv_{r'} | \leq | \equiv_r |$. Furthermore, if (σ, r) is path-finite, also (σ', r') is path-finite. □

This theorem can also be enhanced to partitions with infinite index. The canonical partition is trim (every output of the subtree function is $*$) and path-finite (there are no y-paths in images). If the domain is finite, it has finite index. With this theorem, we assume in the following that every partition is already trim.

5.2.2 Proper Partition

If there exists a path-finite partition with finite index for a transformation, then there may exist more than one. In particular, these different partitions may have the same index. The next example shows a transformation where two partitions (both are trim and path-finite) have the same index and there is no path-finite partition with a smaller index. Thus, these properties do not suffice to get a unique minimal partition.

Example 5.4. Let us consider the quotient ring of integers modulo 3, i.e. $\mathbb{Z}_3 = \{0, 1, 2\}$, with addition $+_3$ and multiplication $\times_3$ (modulo 3). We use prefix notation. Since the operations are commutative, there are six different (ordered) input pairs. We get the following transformation τ:

$$\begin{array}{ll} +_3(\langle 0,0\rangle) \mapsto 0 & \times_3(\langle 0,0\rangle) \mapsto 0 \\ +_3(\langle 0,1\rangle) \mapsto 1 & \times_3(\langle 0,1\rangle) \mapsto 0 \\ +_3(\langle 0,2\rangle) \mapsto 2 & \times_3(\langle 0,2\rangle) \mapsto 0 \\ +_3(\langle 1,1\rangle) \mapsto 2 & \times_3(\langle 1,1\rangle) \mapsto 1 \\ +_3(\langle 1,2\rangle) \mapsto 0 & \times_3(\langle 1,2\rangle) \mapsto 2 \\ +_3(\langle 2,2\rangle) \mapsto 1 & \times_3(\langle 2,2\rangle) \mapsto 1 \end{array}$$

Seen as a tree transformation, we get input trees with root $+_3$ or $\times_3$ (of rank 1) and leaves (of rank 0) in the set

$$\Sigma^{(0)} = \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 0,2\rangle, \langle 1,1\rangle, \langle 1,2\rangle, \langle 2,2\rangle\}$$

To minimize the number of equivalence classes, we have to subsume input subtrees such that they have the same residual. One possibility is to subsume all

subtrees with the same result with respect to the operation $\times_3$. It is done by the following partition (σ_+, r_+):

$$\begin{array}{lll} \sigma_+(p) = \tau(+_3(p)) & r_+(p) = \left\{ \begin{array}{ll} +_3(y) & \mapsto y \\ \times_3(y) & \mapsto \tau(\times_3(p)) \end{array} \right. & \text{for all pairs } p \in \Sigma^{(0)} \\ \sigma_+(u) = \tau(u) & r_+(u)(y) = y & \text{for all } u \in \mathrm{dom}(\tau) \end{array}$$

The subtree function maps every pair at a leaf to the output if $+_3$ is applied, for instance $\sigma_+(\langle 0, 1\rangle) = 1$. Thus, the residuals of such pairs map $+_3(y)$ to y and use the pre-produced output. For the context $\times_3(y)$, they have to produce the correct output by the residual, for example $r_+(\langle 0, 1\rangle)(\times_3(y)) = 0$. Thus, depending on the output of $\tau(\times_3(p))$, there are three different equivalence classes of pairs $p \in \Sigma^{(0)}$. Additionally, all whole input trees have the same residual. They build one further equivalence class.

On the other hand, we can define the partition $(\sigma_\times, r_\times)$ in the same way. It subsumes the pairs with the same result w.r.t. $+_3$ because this result is realized by the residual function. Whereas the output of $\times_3(p)$ is produced at the leaf p:

$$\begin{array}{lll} \sigma_\times(p) = \tau(\times_3(p)) & r_\times(p) = \left\{ \begin{array}{ll} +_3(y) & \mapsto \tau(+_3(p)) \\ \times_3(y) & \mapsto y \end{array} \right. & \text{for all pairs } p \in \Sigma^{(0)} \\ \sigma_\times(u) = \tau(u) & r_\times(u)(y) = y & \text{for all } u \in \mathrm{dom}(\tau) \end{array}$$

For the pair $\langle 0, 1\rangle$, we get $\sigma_\times(\langle 0, 1\rangle) = 0$ and $r_\times(\langle 0, 1\rangle)(+_3(y)) = 1$. Again, we get four equivalence classes (depending on the output of $\tau(+_3(p))$).

One can prove that there is no partition for this transformation with less than four equivalence classes. Thereto consider the possible different residuals. If a residual maps some context to y, this determines the output of the corresponding subtree function and influences the images of the other contexts. Thus, for every leaf there are only a certain number of possible residuals. Hence, we do not have any unique partition with minimal index for this transformation. ◁

In this example, we have seen that there is a transformation with different trim and path-finite partitions. We have to define a new property, which restricts the number of partitions further. The example shows that it is not sufficient to take the partition with the minimal index because it is not unique. The problem in this example is how to segment a finite set of input subtrees in equivalence classes. To solve that, we postpone finite sets of output trees to the residual. If there remain no finite sets of output trees of the subtree function for sets of equivalent input trees except for $\{*\}$, then we have a *proper* partition.

Definition 5.5 (Proper)**.** A trim partition (σ, r) is called *proper* if for each $u \in \mathrm{dom}(\sigma)$ holds

$$\sigma([u]_r) = \{*\} \quad \text{or} \quad |\sigma([u]_r)| = \infty \,.$$

That means, if there is an input tree u with output tree $\sigma(u)$ different to $*$, there are infinitely many output trees of σ, for which the input tree has the same residual $r(u)$. The residual $r(u)$ of a tree u with $|\sigma([u]_r)| = \infty$ then is called *essential*. Otherwise, if $\sigma(u) = *$, the residual $r(u)$ is called *inessential*.

As the property trim, also this property does not limit the set of transformations which are definable by proper partitions with finite index. For every finite set of output trees of r-equivalent input trees of σ, we postpone the output to the residual. The number of equivalence classes of the new partition is linear in the index of $\equiv_r$.

Theorem 5.8. *For every path-finite partition (σ, r) with finite index, there exists an equivalent partition (σ', r'), which is*

- *proper*
- *path-finite*
- *and has finite index.*

Proof. Let τ be the tree function of (σ, r). We define a partition (σ', r') on the domain of (σ, r). The new subtree function is given by:

$$\sigma'(u) = \begin{cases} \sigma(u) & \text{if } |\sigma([u]_r)| = \infty \\ * & \text{otherwise.} \end{cases}$$

In order of the new partition defining the same tree function, we have to change the residual of subtrees u with $|\sigma([u]_r)| < \infty$, and we have to postpone the output of σ.

$$r'(u)(c) = \begin{cases} r(u)(c) & \text{if } |\sigma([u]_r)| = \infty \\ r(u)(c) \cdot \sigma(u) & \text{otherwise.} \end{cases}$$

Now we prove that this pair (σ', r') is a proper and path-finite partition with finite index, which describe τ:

- The pair (σ', r') describes the same tree function τ as (σ, r) and it is consistent: Let $c \cdot u \in \mathrm{dom}(\tau)$. Then $u \in \mathrm{dom}(r)$ and $c \in \mathrm{dom}(r(u))$. If the set of outputs of r-equivalent trees of u is infinite, i.e., $|\sigma([u]_r)| = \infty$, then
$$r'(u)(c) \cdot \sigma'(u) = r(u)(c) \cdot \sigma(u) = \tau(c \cdot u) \ .$$
Otherwise, if there are only finitely many outputs, $|\sigma([u]_r)| < \infty$, then
$$r'(u)(c) \cdot \sigma'(u) = r(u)(c) \cdot \sigma(u) \cdot * = r(u)(c) \cdot \sigma(u) = \tau(c \cdot u) \ .$$
Thus, (σ', r') is a partition for the transformation τ.
- The partition is trim: The partition (σ, r) is trim. Thus, for every subtree $u \in \mathrm{dom}(r)$ with $|\sigma([u]_r)| = \infty$, the trim condition
$$r(u)(\mathcal{C}_\Sigma(y)) \subseteq \mathcal{T}_\Delta \Leftrightarrow \sigma(u) = *$$
holds also for (σ', r') on these trees. Otherwise, the output of σ' is $*$ and the range of the residual is in $\mathcal{T}_\Delta$ because $\sigma(u) \in \mathcal{T}_\Delta$ and with that also $r'(u)(c) = r(u)(c) \cdot \sigma(u)$ for all $c \in \mathrm{dom}(r'(u))$.

- It is proper: Assume it is not proper, then there is a tree $u \in \mathrm{dom}(r')$ with

$$|\sigma'([u]_{r'})| < \infty \quad \text{and} \quad \sigma'([u]_{r'}) \neq \{*\} .$$

It follows that there must be a r'-equivalent tree $u' \equiv_{r'} u$ with $\sigma'(u') \neq *$. Then, by the construction of σ', the tree u' gets its output by σ, i.e., $\sigma'(u') = \sigma(u')$, which includes $|\sigma([u']_r)| = \infty$. Additionally, every tree u'', which is r-equivalent to u', i.e., $u'' \equiv_r u'$, is in the same infinite equivalence class: $|\sigma([u'']_r)| = \infty$. This implies that for all these trees, σ' and r' are defined as σ and r, respectively, i.e., $\sigma'(u'') = \sigma(u'')$ and $r'(u'') = r(u'')$. Since all these trees are r-equivalent, we get

$$r'(u'') = r(u'') = r(u') = r'(u') .$$

With that, the trees are also equivalent w.r.t r'. Thus, the r'-equivalence class of u is infinite:

$$[u]_{r'} = [u']_{r'} \supseteq [u']_r .$$

It causes a contradiction because the set of output trees is infinite, too:

$$|\sigma'([u]_{r'})| = |\sigma'([u']_{r'})| > |\sigma([u']_r)| = \infty .$$

- The partition is path-finite: Since the residual under r' of a tree u equals the residual under r or it is a subset of the set $\mathcal{T}_\Delta$ of ground terms. Thus, for the set of y-paths in the images we get for every $c \in \mathrm{dom}(r'(u))$:

$$\mathtt{Paths}_y(r'(u)(c)) \subseteq \mathtt{Paths}_y(r(u)(c))$$

If (σ, r) is path-finite, also (σ', r') is path-finite.

- It remains to proof that the index of the equivalence relation $\equiv_{r'}$ is finite. Consider two r-equivalent trees $u \equiv_r u'$. If $|\sigma([u]_r)| = \infty$ then they have the same residual under r': $r'(u) = r(u) = r(u') = r'(u')$. Hence, u and u' are equivalent w.r.t. r', too: $u \equiv_{r'} u'$. For every equivalence class $[u]_r$ with $|\sigma([u]_r)| = \infty$, there is one equivalence class $[u]_{r'}$. Thus, there are only finitely many classes of this form.

Assume that $|\sigma([u]_r)| < \infty$ and $\sigma(u) = \sigma(u')$. For every $c \in \mathrm{dom}(r(u))$, we get

$$r'(u)(c) = r(u)(c) \cdot \sigma(u) = r(u')(c) \cdot \sigma(u') = r'(u')(c)$$

Thus, the two trees u and u' are r'-equivalent, i.e., $u \equiv_{r'} u'$. For every equivalence class $[u]_r$ with $|\sigma([u]_r)| < \infty$, there are at most $|\sigma([u]_r)|$ equivalence classes

$$[u]_{r'} = \{u' \mid \sigma(u) = \sigma(u'), u \equiv_r u'\} .$$

Thus, the number of equivalence classes of $\equiv_{r'}$ is finite. □

Example 5.5. For the transformation τ of Example 5.4 we get the following proper partition (σ, r), which is also path-finite and has finite index. This partition produces the output as late as possible because every output set (of equivalent input trees) is finite. Starting with an arbitrary (path-finite and trim) partition of τ, we get the following:

$$\begin{array}{lll} \sigma(p) = * & r(p) = \left\{ \begin{array}{ll} +_3(y) & \mapsto \tau(+_3(p)) \\ \times_3(y) & \mapsto \tau(\times_3(p)) \end{array} \right. & \text{for all pairs } p \in \Sigma^{(0)} \\ \sigma(u) = * & r(u)(y) = \tau(u) & \text{for all } u \in \mathrm{dom}(\tau) \end{array}$$

This partition is the canonical partition (σ_*, r_*) (Page 39). It has twelve different equivalence classes, but for this transformation, it is the only partition (for a fix symbol $*$), which is proper. ◁

5.2.3 Earliest Partition

In the previous section, we introduced a partition where finite sets of outputs are postponed to the residual, i.e., trivial outputs are produced as late as possible. If we have infinitely many outputs for an equivalence class, there also may be different partitions with the same index, which fulfill the properties mentioned until here. If we postpone infinitely many outputs to the residuals, we get an infinite index. To preserve the finiteness, we contrarily produce the output as early as possible, i.e., by the subtree function instead of the residual.

Example 5.6. Consider the following transformation (Figure 5.3):

$$\tau : \mathtt{b}^n(\mathtt{e}) \mapsto \mathtt{B}^{n+1}(\mathtt{E}) \qquad \text{for all } n \in \mathbb{N}_0$$

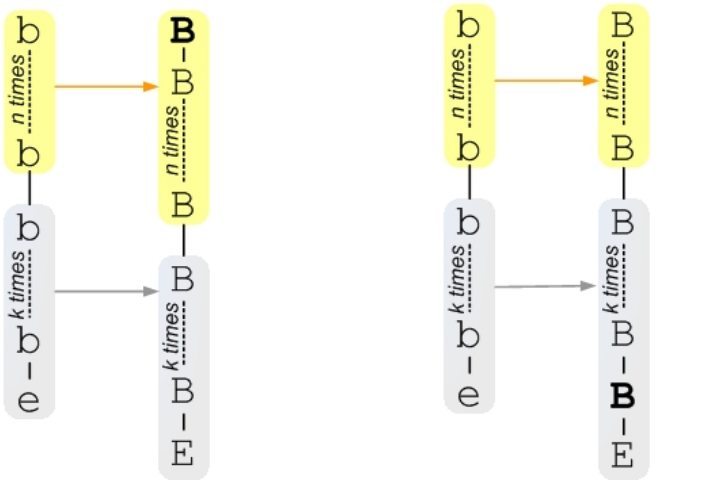
This function copies the input tree (only changing the labels to capital letters), and adds an additional B-labeled node. If we want to define a partition, we have to decide whether we produce this additional node by the subtree function or by the residual. If we add the B-node in the residual (for every tree), we get the partition (σ, r) with

$$\begin{array}{rl} \sigma(\mathtt{b}^n(\mathtt{e})) = \mathtt{B}^n(\mathtt{E}) & \text{for all } n \in \mathbb{N}_0 \\ r(\mathtt{b}^n(\mathtt{e}))(\mathtt{b}^k(y)) = \mathtt{B}^{k+1}(y) & \text{for all } n, k \in \mathbb{N}_0 \end{array}$$

The other way around, we get the partition (σ', r') with

$$\begin{array}{rl} \sigma'(\mathtt{b}^n(\mathtt{e})) = \mathtt{B}^{n+1}(\mathtt{E}) & \text{for all } n \in \mathbb{N}_0 \\ r'(\mathtt{b}^n(\mathtt{e}))(\mathtt{b}^k(y)) = \mathtt{B}^k(y) & \text{for all } n, k \in \mathbb{N}_0 \end{array}$$

which produces the additional node in the subtree function (Figure 5.3). Both partitions fulfill all the former conditions, i.e., they are path-finite, proper and have finite index. In particular, the index is 1. They have both the minimal index. ◁



(a) The "latest" partition (σ, r). (b) The earliest partition (σ', r').

Figure 5.3: The transformation of Example 5.6 with two different partitions.

To get a unique partition in the end, we have to indicate one partition. In the previous section, we have postponed finite sets of outputs at the cost of a minimal index. If we now postpone infinite sets of outputs, we may get an infinite index. Which contradicts the goal of a partition, which is definable by a DBTT. Instead, we will move common parts of images to the subtree function, without increasing the number of equivalence classes. In the given example, we would choose the second partition (σ', r') where the additional `B`-node is produced as early as possible (Figure 5.3(b)).

Formally, for a context function $\varphi : \mathcal{C}_\Sigma(y) \dashrightarrow \mathcal{T}_\Delta(y)$ let $\hat{\mathcal{Z}}(\varphi)$ denote the set of images $z \in \hat{\mathcal{T}}_\Delta(y)$ in the range of φ. Thus, every tree in $\hat{\mathcal{Z}}(\varphi)$ contains an occurrence of the variable y. Then we define the *greatest common suffix* of φ as the greatest common suffix (Page 19) of all trees in $\hat{\mathcal{Z}}(\varphi)$, i.e.,

$$\mathsf{gcs}(\varphi) = \mathsf{gcs}(\hat{\mathcal{Z}}(\varphi)) = \bigsqcup \hat{\mathcal{Z}}(\varphi)$$

w.r.t. the order $\sqsubseteq$ on $\hat{\mathcal{T}}_\Delta(y)_\bot$ from Section 2.3. Note that the least upper bound for the empty set in this order is $\bot$, i.e., $\bigsqcup \emptyset = \bot$. Thus, the greatest common suffices of inessential residuals are defined and equal $\bot$.

Let (σ, r) be a partition of a function τ. The *greatest common suffix* of an input tree $u \in \mathrm{dom}(\sigma)$ with respect to a residual function r is defined as the greatest common suffix of its residual, i.e.,

$$\mathsf{gcs}(u) = \mathsf{gcs}(r(u)) \ .$$

Analogically, we use $\hat{\mathcal{Z}}(u) = \hat{\mathcal{Z}}(r(u))$.

Definition 5.6 (Earliest)**.** A proper partition (σ, r) is called *earliest* if for all trees $u \in \mathrm{dom}(r)$ holds $\mathsf{gcs}(u) \in \{y, \bot\}$.

The greatest common suffix of a tree u in an earliest partition is $\bot$ if all images of contexts under $r(u)$ are ground, i.e., $r(u)$ is inessential. Otherwise, if there exists an image $z = r(u)(c)$ for some context c, then $\mathsf{gcs}(u) = y$. For every subset of $\hat{\mathcal{T}}_\Delta(y)$ the least upper bound w.r.t. $\sqsubseteq$ exists (cf. Section 2.3). Note that the canonical partition is earliest, because all residuals are inessential and, thus, have the greatest common suffix $\bot$. Assume that for a transformation τ a path-finite partition with finite index exists. We show that for such a transformation τ an earliest partition exists, which, in addition, is also path-finite and has finite index:

Theorem 5.9. *For every proper and path-finite partition (σ, r) with finite index, there exists an equivalent partition (σ', r'), which is*

- *earliest,*
- *path-finite and*
- *has finite index.*

Proof. Let τ be the tree function of (σ, r). Since $(\hat{\mathcal{T}}_\Delta(y)_\bot, \sqsubseteq)$ is a complete lattice satisfying the ascending chain condition, the greatest common suffix $\mathsf{gcs}(u)$ (w.r.t. r) exists for every tree $u \in \mathrm{dom}(\sigma)$.

We define a partition (σ', r') on the domain of (σ, r). The new subtree function is given by:

$$\sigma'(u) = \begin{cases} * & \text{if } \mathsf{gcs}(u) = \bot \\ \mathsf{gcs}(u) \cdot \sigma(u) & \text{otherwise.} \end{cases}$$

and the new residual $r'(u)$ of a tree $u \in \mathrm{dom}(r)$ is defined by

$$r'(u)(c) = \begin{cases} r(u)(c) & \text{if } \mathsf{gcs}(u) = \bot \\ z & \text{with } r(u)(c) = z \cdot \mathsf{gcs}(u), \text{ otherwise.} \end{cases}$$

Let $\mathsf{gcs}(u)$ and $\mathsf{gcs}'(u)$ denote the greatest common suffix of trees $u \in \mathrm{dom}(\sigma)$ w.r.t. r and r', respectively. First, we observe the following:

1. For all $u \in \mathrm{dom}(r)$ holds $\mathsf{gcs}(u) = \bot \Leftrightarrow \mathsf{gcs}'(u) = \bot$. This follows by the definition of (σ', r'): If $\mathsf{gcs}(u) = \bot$, the residual is the same for r and r'. Thus, the greatest common suffix is the same. If $\mathsf{gcs}(u) \neq \bot$, then there is a context c such that its image under $r(u)$ is not ground. But then also $r'(u)(c)$, which is determined by $r(u)(c) = r'(u)(c) \cdot \mathsf{gcs}(u)$, is in $\hat{\mathcal{T}}_\Delta(y)$, too. With that, $\mathsf{gcs}'(u) \neq \bot$.
2. For all $u \in \mathrm{dom}(r)$ holds $\sigma(u) = * \Leftrightarrow \sigma'(u) = *$ because

$$\begin{aligned} \sigma(u) = * &\overset{\text{trim}}{\iff} r(u)(\mathcal{C}_\Sigma) \subseteq \mathcal{T}_\Delta \\ &\iff \mathsf{gcs}(u) = \bot \\ &\iff \sigma'(u) = * \,. \end{aligned}$$

3. For all $u, u' \in \mathrm{dom}(r)$ holds $u \equiv_r u' \implies u \equiv_{r'} u'$. It holds because u and u' have the same residual w.r.t. r if they are r-equivalent. Furthermore, if their $\mathsf{gcs}(r(u))$ is $\bot$, then their residual under r' is the same as under r. Thus, they are also r'-equivalent. Otherwise, if $\mathsf{gcs}(r(u)) \neq \bot$, the new image for a context c equals for $r'(u)$ and $r'(u')$:

$$r'(u)(c) \cdot \mathsf{gcs}(u) = r(u)(c) = r(u')(c) = r'(u')(c) \cdot \mathsf{gcs}(u') = r'(u')(c) \cdot \mathsf{gcs}(u)$$

With Proposition 2.1 follows $r'(u)(c) = r'(u')(c)$ because $\mathsf{gcs}(u) \in \hat{\mathcal{T}}_\Delta(y)$.

Now we show that (σ', r') is a trim, proper, path-finite and earliest partition with finite index, which describes τ.

- It is consistent and defines τ: Let $c \cdot u \in \mathrm{dom}(\tau)$. Then $u \in \mathrm{dom}(r)$ and $c \in \mathrm{dom}(r(u))$. If the greatest common suffix of u is $\bot$, i.e., $\mathsf{gcs}(u) = \bot$, then $\hat{\mathcal{Z}}(u) = \emptyset$. That means, all images of contexts of u are ground and since (σ, r) is trim, it follows $\sigma(u) = *$. We get

$$r'(u)(c) \cdot \sigma'(u) = r(u)(c) \cdot * = r(u)(c) \cdot \sigma(u) = \tau(c \cdot u) \ .$$

  Otherwise, if $\mathsf{gcs}(u) \in \hat{\mathcal{T}}_\Delta(y)$, we get

$$r'(u)(c) \cdot \sigma'(u) = r'(u)(c) \cdot \mathsf{gcs}(u) \cdot \sigma(u) = r(u)(c) \cdot \sigma(u) = \tau(c \cdot u) \ .$$

  Consequently, (σ', r') is a partition for the transformation τ.

- The new partition is trim: Let $u \in \mathrm{dom}(\sigma')$. Then it follows:

$$\begin{aligned} \sigma'(u) = * &\Leftrightarrow \sigma(u) = * && \text{Property 2} \\ &\Leftrightarrow r(u)(\mathcal{C}_\Sigma) \subseteq \mathcal{T}_\Delta && (\sigma, r) \text{ is trim} \\ &\Leftrightarrow \mathsf{gcs}(u) = \bot && \text{Def. } \mathsf{gcs} \\ &\Leftrightarrow r'(u)(\mathcal{C}_\Sigma) = r(u)(\mathcal{C}_\Sigma) \subseteq \mathcal{T}_\Delta && \text{Def. } r' \end{aligned}$$

- It is proper: Let $u \in \mathrm{dom}(\sigma')$ be a tree with $\sigma'(u) \neq *$. Then it follows with Property 2 that also $\sigma(u) \neq *$. This implies that there are infinitely many different output trees of r-equivalent input trees, i.e., $|\sigma([u]_r)| = \infty$. All trees $u' \equiv_r u$ have the same $\mathsf{gcs}(u') = \mathsf{gcs}(u) \in \hat{\mathcal{T}}_\Delta(y)$. With Property 3 we know that $[u]_r \subseteq [u]_{r'}$. It follows:

$$\sigma'([u]_{r'}) \supseteq \{\mathsf{gcs}(u) \cdot v \mid v \in \sigma([u]_r)\}$$

  Thus, $|\sigma([u]_{r'})| = \infty$

- The partition is earliest: Let $u \in \mathrm{dom}(r)$. If $\mathsf{gcs}(u) = \bot$, with Property 1 it follows that also $\mathsf{gcs}'(u) = \bot$. Otherwise, assume $\mathsf{gcs}(u) \neq \bot$. Then, we know for every context c with $r'(u)(c) \in \hat{\mathcal{T}}_\Delta(y)$ that $r'(u)(c) \sqsubseteq \mathsf{gcs}'(u)$. Furthermore,

$$r(u)(c) = r'(u)(c) \cdot \mathsf{gcs}(u) \sqsubseteq \mathsf{gcs}'(u) \cdot \mathsf{gcs}(u) \ .$$

  Since this holds for every context c with non-ground image $r(u)(c)$, we also get $\mathsf{gcs}(u) \sqsubseteq \mathsf{gcs}'(u) \cdot \mathsf{gcs}(u)$. Thus, $\mathsf{gcs}'(u) = y$.

- It is path-finite: Let $u \in \mathrm{dom}(\sigma')$, $c \in \mathrm{dom}(r'(u))$ and p a y-path in $\mathtt{Paths}_y(r'(u)(c))$. Since p is an y-path in $r'(u)(c)$, this image is in $\hat{\mathcal{T}}_\Delta(y)$. Thus, it fulfills the equation

$$r'(u)(c) \cdot \mathsf{gcs}(u) = r(u)(c) \; .$$

  Also $\mathsf{gcs}(u)$ is in $\hat{\mathcal{T}}_\Delta(y)$ and there is a y-path $p' \in \mathtt{Paths}_y(\mathsf{gcs}(u))$ such that $p \cdot p'$ is a y-path in $r(u)(c)$, i.e., $p \cdot p' \in \mathtt{Paths}_y(r(u)(c))$. With

$$|\mathtt{Paths}_y(r'(u)(\mathcal{C}_\Sigma))| \leq |\mathtt{Paths}_y(r(u)(\mathcal{C}_\Sigma))| \; ,$$

  it follows that (σ', r') is path-finite because (σ, r) is path-finite.

- The equivalence relation $\equiv_{r'}$ has finite index: With Property 3, we get that $\equiv_{r'}$ is a coarsening of $\equiv_r$. Thus, the index of (σ', r') is less or equal the index of (σ, r), which is finite. □

Returning to the example in the beginning of this section, we see that we already have found an earliest partition:

Example 5.7. Let us consider the greatest common suffices of the different residuals in the two partitions of Example 5.6. For the first partition (σ, r) we get the same residual for every input subtree $\mathtt{b}^n(\mathtt{e})$ (for every $n \in \mathbb{N}_0$). The set of images is

$$\hat{\mathcal{Z}}(\mathtt{b}^n(\mathtt{e})) = r(\mathtt{b}^n(\mathtt{e}))(\mathcal{C}_\Sigma) = \{\mathtt{B}^{k+1}(y) \mid k \in \mathbb{N}_0\} \; .$$

Thus, $\mathsf{gcs}(\mathtt{b}^n(\mathtt{e})) = \bigsqcup \hat{\mathcal{Z}}(\mathtt{b}^n(\mathtt{e})) = \mathtt{B}(y)$. This partition is not earliest, but if we apply the construction of the proof, we get exactly the second partition (σ', r') with

$$\mathsf{gcs}(r'(\mathtt{b}^n(\mathtt{e}))) = \bigsqcup r'(\mathtt{b}^n(\mathtt{e}))(\mathcal{C}_\Sigma) = \bigsqcup \{\mathtt{B}^k(y) \mid k \in \mathbb{N}_0\} = y$$

for every $n \in \mathbb{N}_0$. Hence, this partition is earliest. ◁

5.2.4 Unified Partition

In the Example 5.4, we considered a tree function with different partitions, which are not proper. We had to decide how to subsume the leaves to equivalence classes. The obvious possibilities were either by the result of $+_3$ or by the result of $\times_3$. This gave us two different partitions of the same index. We solved that by taking a proper partition. In that case, the involved equivalence classes have finitely many different outputs. Now there is a similar case for equivalence classes with infinitely many outputs. There may be an input subtree u for which we can choose two different output trees v and v', which cause different essential residuals for u. Nevertheless, we want to define the same transformation. Hence, the residuals only differ somehow in the subtrees v and v' in the images, i.e., the two residuals are unifiable by $\langle v, v' \rangle$ (cf. Page 20). Consider the following example, which presents a partition with two unifiable essential residuals.

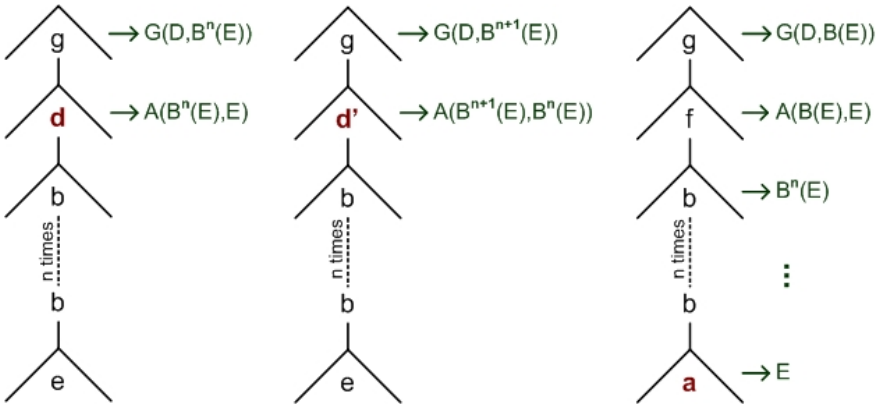


Figure 5.4: The function τ_{mgu} of Example 5.8.

The subtree function σ is defined (for all $n \in \mathbb{N}_0$) by:

$$\begin{aligned} \mathtt{g}(\mathtt{d}(\mathtt{b}^n(\mathtt{e}))) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}^n(\mathtt{E})) & \mathtt{g}(\mathtt{d}'(\mathtt{b}^n(\mathtt{e}))) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}^{n+1}(\mathtt{E})) \\ \mathtt{d}(\mathtt{b}^n(\mathtt{e})) &\mapsto \mathtt{B}^n(\mathtt{E}) & \mathtt{d}'(\mathtt{b}^n(\mathtt{e})) &\mapsto \mathtt{B}^n(\mathtt{E}) \\ \mathtt{b}^n(\mathtt{e}) &\mapsto \mathtt{B}^n(\mathtt{E}) & \mathtt{g}(\mathtt{f}(\mathtt{b}^n(\mathtt{a}))) &\mapsto * \\ \mathtt{b}^n(\mathtt{a}) &\mapsto \mathtt{B}^n(\mathtt{E}) & \mathtt{f}(\mathtt{b}^n(\mathtt{a})) &\mapsto \mathtt{E} \end{aligned}$$

The residual of $\mathtt{b}^n(\mathtt{e})$ (for all $n \in \mathbb{N}_0$) is given by (with $k \in \mathbb{N}_0$):

$$\begin{aligned} \mathtt{d}(\mathtt{b}^k(y)) &\mapsto \mathtt{A}(\mathtt{B}^k(y), \mathtt{E}) & \mathtt{d}'(\mathtt{b}^k(y)) &\mapsto \mathtt{A}(\mathtt{B}^{k+1}(y), \mathtt{B}^k(y)) \\ \mathtt{g}(\mathtt{d}(\mathtt{b}^k(y))) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}^k(y)) & \mathtt{g}(\mathtt{d}'(\mathtt{b}^k(y))) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}^{k+1}(y)) \ . \end{aligned}$$

For $\mathtt{b}^n(\mathtt{a})$ (for all $n \in \mathbb{N}_0$) we have the residual (with $k \in \mathbb{N}_0$)

$$\mathtt{b}^k(y) \mapsto \mathtt{B}^k(y) \qquad \mathtt{f}(\mathtt{b}^k(y)) \mapsto \mathtt{A}(\mathtt{B}(\mathtt{E}), \mathtt{E}) \qquad \mathtt{g}(\mathtt{f}(\mathtt{b}^k(y))) \mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E})) \ .$$

For all other input subtrees, we have the following residual for all $n \in \mathbb{N}_0$:

		$r(u)$	
u	$\sigma(u)$	y	$\mathtt{g}(y)$
$\mathtt{g}(\mathtt{d}(\mathtt{b}^n(\mathtt{e})))$	$\mathtt{G}(\mathtt{D}, \mathtt{B}^n(\mathtt{E}))$	y	
$\mathtt{g}(\mathtt{d}'(\mathtt{b}^n(\mathtt{e})))$	$\mathtt{G}(\mathtt{D}, \mathtt{B}^{n+1}(\mathtt{E}))$	y	
$\mathtt{g}(\mathtt{f}(\mathtt{b}^n(\mathtt{a})))$	$*$	$\mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E}))$	
$\mathtt{d}(\mathtt{b}^n(\mathtt{e}))$	$\mathtt{B}^n(\mathtt{E})$	$\mathtt{A}(y, \mathtt{E})$	$\mathtt{G}(\mathtt{D}, y)$
$\mathtt{d}'(\mathtt{b}^n(\mathtt{e}))$	$\mathtt{B}^n(\mathtt{E})$	$\mathtt{A}(\mathtt{B}(y), y)$	$\mathtt{G}(\mathtt{D}, \mathtt{B}(y))$
$\mathtt{f}(\mathtt{b}^n(\mathtt{a}))$	$\mathtt{E}$	$\mathtt{A}(\mathtt{B}(y), y)$	$\mathtt{G}(\mathtt{D}, \mathtt{B}(y))$

Figure 5.5: The partition (σ, r) for the tree function τ_{mgu} of Example 5.8.

Example 5.8. The Figure 5.4 illustrates the following function τ_{mgu} ($\forall n \in \mathbb{N}_0$):

$$\begin{aligned} \mathtt{g}(\mathtt{d}(\mathtt{b}^n(\mathtt{e}))) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}^n(\mathtt{E})) & \mathtt{g}(\mathtt{f}(\mathtt{b}^n(\mathtt{a}))) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E})) \\ \mathtt{g}(\mathtt{d}'(\mathtt{b}^n(\mathtt{e}))) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}^{n+1}(\mathtt{E})) & \mathtt{f}(\mathtt{b}^n(\mathtt{a})) &\mapsto \mathtt{A}(\mathtt{B}(\mathtt{E}), \mathtt{E}) \\ \mathtt{d}(\mathtt{b}^n(\mathtt{e})) &\mapsto \mathtt{A}(\mathtt{B}^n(\mathtt{E}), \mathtt{E}) & \mathtt{b}^n(\mathtt{a}) &\mapsto \mathtt{B}^n(\mathtt{E}) \\ \mathtt{d}'(\mathtt{b}^n(\mathtt{e})) &\mapsto \mathtt{A}(\mathtt{B}^{n+1}(\mathtt{E}), \mathtt{B}^n(\mathtt{E})) \end{aligned}$$

All input trees are monadic. Note that the input trees on the left-hand side differ pairwise in the node with label $\mathtt{d}$ and $\mathtt{d}'$. On the right-hand side, we have another leaf with label $\mathtt{a}$; the $\mathtt{d}$-labeled node is replaced by a node with label $\mathtt{f}$. The subtrees of the form $\mathtt{b}^n(\mathtt{e})$ on the left-hand side do not have an image. In contrast to that, all subtrees on the right-hand side have an image.

Now let us define a partition, which is earliest, path-finite and has finite index. We first consider the left-hand side. Let $u_n = \mathtt{b}^n(\mathtt{e})$ for all $n \in \mathbb{N}_0$. For this subtrees u_n (for every $n \in \mathbb{N}_0$), the subtree function has to output $\mathtt{B}^n(\mathtt{E})$. This is necessary because this is the common part of the images of $\mathtt{g}(\mathtt{d}(\mathtt{b}^n(\mathtt{e})))$ and $\mathtt{d}(\mathtt{b}^n(\mathtt{e}))$, which depends on the number n of $\mathtt{b}$-labeled nodes in u_n. Otherwise, it contradicts one of the mentioned properties. We get the following subtree function σ for the input trees on the left-hand side ($\forall n \in \mathbb{N}_0$):

$$\begin{aligned} \mathtt{g}(\mathtt{d}(u_n)) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}^n(\mathtt{E})) & \mathtt{g}(\mathtt{d}'(u_n)) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}^{n+1}(\mathtt{E})) \\ \mathtt{d}(u_n) &\mapsto \mathtt{B}^n(\mathtt{E}) & \mathtt{d}'(u_n) &\mapsto \mathtt{B}^n(\mathtt{E}) \\ u_n &\mapsto \mathtt{B}^n(\mathtt{E}) \end{aligned}$$

With the restriction of the mentioned properties, the corresponding residual function r is defined as follows. For the input subtrees u_n for every $n \in \mathbb{N}_0$ we get the residual (for all $k \in \mathbb{N}_0$)

$$\begin{aligned} \mathtt{d}(\mathtt{b}^k(y)) &\mapsto \mathtt{A}(\mathtt{B}^k(y), \mathtt{E}) & \mathtt{d}'(\mathtt{b}^k(y)) &\mapsto \mathtt{A}(\mathtt{B}^{k+1}(y), \mathtt{B}^k(y)) \\ \mathtt{g}(\mathtt{d}(\mathtt{b}^k(y))) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}^k(y)) & \mathtt{g}(\mathtt{d}'(\mathtt{b}^k(y))) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}^{k+1}(y)) \ . \end{aligned}$$

For the other elements in $\mathrm{dom}(\sigma)$ we have for all $n \in \mathbb{N}_0$:

	$r(u)$	
u	y	$\mathtt{g}(y)$
$\mathtt{g}(\mathtt{d}(\mathtt{b}^n(\mathtt{e})))$	y	
$\mathtt{g}(\mathtt{d}'(\mathtt{b}^n(\mathtt{e})))$	y	
$\mathtt{d}(\mathtt{b}^n(\mathtt{e}))$	$\mathtt{A}(y, \mathtt{E})$	$\mathtt{G}(\mathtt{D}, y)$
$\mathtt{d}'(\mathtt{b}^n(\mathtt{e}))$	$\mathtt{A}(\mathtt{B}(y), y)$	$\mathtt{G}(\mathtt{D}, \mathtt{B}(y))$

We get four different residuals and there is no possibility to cut it down to three or less. Now consider the trees on the right-hand side. Again, we have to produce $\mathtt{B}^n(\mathtt{E})$ for the subtrees $\mathtt{b}^n(\mathtt{a})$. The trees $\mathtt{g}(\mathtt{f}(\mathtt{b}^n(\mathtt{a})))$ have only the context y. Thus, we produce $\mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E}))$ by the residual and output $*$ by the subtree function, to get a proper partition.

However, for the tree $\mathtt{f}(\mathtt{b}^n(\mathtt{a}))$, there are different possibilities. First, let us consider the residual if

$$\sigma(\mathtt{f}(\mathtt{b}^n(\mathtt{a}))) = \mathtt{B}(\mathtt{E}) \; .$$

We get

	$r(u)$	
u	y	$\mathtt{g}(y)$
$\mathtt{f}(\mathtt{b}^n(\mathtt{a}))$	$\mathtt{A}(y, \mathtt{E})$	$\mathtt{G}(\mathtt{D}, y)$

This residual equals the residual of $\mathtt{d}(\mathtt{b}^n(\mathtt{e}))$. Thus, we do not get a new equivalence class. This partition (σ, r) is completely given in Figure 5.5. On the other hand, one can choose

$$\sigma(\mathtt{f}(\mathtt{b}^n(\mathtt{a}))) = \mathtt{E} \; .$$

Then the residual equals the residual of $\mathtt{d}'(\mathtt{b}^n(\mathtt{e}))$:

	$r(u)$	
u	y	$\mathtt{g}(y)$
$\mathtt{f}(\mathtt{b}^n(\mathtt{a}))$	$\mathtt{A}(\mathtt{B}(y), y)$	$\mathtt{G}(\mathtt{D}, \mathtt{B}(y))$

We have the same number of equivalence classes as before. Both of these partitions fulfill the former properties. ◁

This example shows that the properties path-finite, earliest, and having finite index do not suffice to get a unique partition. The problem for the transformation of the example is that the two residuals of $u = \mathtt{d}(\mathtt{b}(\mathtt{e}))$ and $u' = \mathtt{d}'(\mathtt{b}(\mathtt{e}))$ have a unifier. That means, the pair $\langle \mathtt{B}(\mathtt{E}), \mathtt{E} \rangle$ fulfills the following condition. For all contexts $c \in \mathrm{dom}(r(u)) = \mathrm{dom}(r(u'))$ holds:

$$r(u)(c) \cdot \mathtt{B}(\mathtt{E}) = r(u')(c) \cdot \mathtt{E} \tag{5.7}$$

In Section 2.3, the most-general unifier of two trees is introduced and it is extended to the most-general unifier of a set of pairs of trees (Page 21). Here, we consider the most-general unifier of two residuals. It is the most-general unifier of the set of pairs of images corresponding to the same context.

Formally, for context functions $\varphi, \varphi' : \mathcal{C}_\Sigma(y) \dashrightarrow \mathcal{T}_\Delta(y)$ let

$$C(\varphi, \varphi') = \{(\varphi(c), \varphi'(c)) \mid c \in \mathrm{dom}(\varphi)\}$$

if $\mathrm{dom}(\varphi) = \mathrm{dom}(\varphi')$ and $C(\varphi, \varphi')$ is undefined, otherwise. Then the most-general unifier of two context functions φ and φ' is

$$\mathsf{mgu}(\varphi, \varphi') = \begin{cases} \bot & \text{if } C(\varphi, \varphi') \text{ is undefined} \\ \mathsf{mgu}(C(\varphi, \varphi')) & \text{otherwise.} \end{cases}$$

As for trees, we say that two context functions are *unifiable* if the most-general unifier is not $\bot$. For example, Equation 5.7 shows that the residuals $r(\mathtt{d}(\mathtt{b}(\mathtt{e})))$ and $r(\mathtt{d}'(\mathtt{b}(\mathtt{e})))$ are unifiable with the unifier $\langle \mathtt{B}(\mathtt{E}), \mathtt{E} \rangle$. This is already the most-general unifier of $r(\mathtt{d}(\mathtt{b}(\mathtt{e})))$ and $r(\mathtt{d}'(\mathtt{b}(\mathtt{e})))$, i.e.,

$$\mathsf{mgu}(r(\mathtt{d}(\mathtt{b}(\mathtt{e}))), r(\mathtt{d}'(\mathtt{b}(\mathtt{e})))) = \langle \mathtt{B}(\mathtt{E}), \mathtt{E} \rangle \; .$$

Let (σ, r) be a path-finite and earliest partition with finite index. The most-general unifier $\mathsf{mgu}(r(u_1), r(u_2)) = \langle z_1, z_2 \rangle$ for unifiable residuals $r(u_1), r(u_2)$ of input trees $u_1, u_2 \in \mathrm{dom}(r)$ has the following properties:

- If $r(u_i)$ is inessential, then $r(u_i)(\mathcal{C}_\Sigma) \in \mathcal{T}_\Delta$. Therefore, $z_i = \top$.
- Moreover, z_1 contains y iff z_2 contains y. If both z_1 and z_2 contain y, the mgu must equal $\langle y, y \rangle$, otherwise the partition would not be earliest.

If a ground term v is in the set of outputs $\sigma([u]_r)$ of an equivalence class with residual $r(u)$, then v is called *realizable* in $r(u)$. Note that the ground terms z occurring in most-general unifiers of residuals are, however, not necessarily realizable.

Definition 5.7 (Unified Earliest)**.** An earliest partition (σ, r) is called *unified earliest* if no ground term in most-general unifiers of residuals of r is realizable.

Assume for a path-finite and earliest partition with finite index, the most-general unifiers for pairs of residuals are known. Then an equivalent unified earliest partition can be constructed, which is again path-finite and has finite index.

Theorem 5.10. *For each earliest and path-finite partition (σ, r) with finite index, there exists an equivalent partition (σ', r'), which is*

- *unified earliest,*
- *path-finite and*
- *has finite index.*

Proof. Let τ be the tree function of (σ, r) and Z the set of all subtrees of ground most-general unifiers, i.e.,

$$Z = \mathtt{Subtrees}\{z \in \mathcal{T}_\Delta \mid u, u' \in \mathrm{dom}(r), \mathsf{mgu}(r(u), r(u')) = \langle z, z' \rangle \text{ for some } z'\} \,.$$

Then we define the new partition (σ', r') for the same domain as follows.

$$\sigma'(u) = * \quad \text{and} \quad r'(u)(c) = r(u)(c) \cdot \sigma(u) \qquad \text{if } \sigma(u) \in Z \tag{5.8}$$

$$\sigma'(u) = \sigma(u) \quad \text{and} \quad r'(u)(c) = r(u)(c) \qquad \text{if } \sigma(u) \notin Z \tag{5.9}$$

Since the new definition of $r'(u)$ only depends on $r(u)$ and $\sigma(u)$, it is a well-defined partition. First, we observe:

1. If the new residual under r' of a tree u do not equals the former residual $r(u)$, then it is inessential (because all $\sigma(u) \in Z$ are ground).
2. The number of new residuals is finite. This follows because $\equiv_r$ has finite index. Hence, the number of different most-general unifiers is finite, too. A new residual $r'(u)$ for a tree u is composed by an old residual $r(u)$ and a ground subterm of a most-general unifier.

Now we have to prove that (σ', r') is unified earliest and path-finite, has finite index, and is equivalent to (σ, r).

- The partition (σ', r') defines τ: Let $u \in \mathrm{dom}(\sigma)$. Assume the output of u is in Z, i.e., $\sigma(u) \in Z$. Then we get:

  $$\tau(c \cdot u) = r(u)(c) \cdot \sigma(u) = r'(u)(c) = r'(u)(c) \cdot * = r'(u)(c) \cdot \sigma'(u)$$

  Otherwise, if $\sigma(u) \notin Z$, it is the same as for (σ, r), i.e.,

  $$\tau(c \cdot u) = r(u)(c) \cdot \sigma(u) = r'(u)(c) \cdot \sigma'(u) \; .$$

  Thus, the tree function of (σ', r') is τ.

- (σ', r') is trim: If $\sigma'(u) \neq *$ then $r'(u)(\mathcal{C}_\Sigma(y)) = r(u)(\mathcal{C}_\Sigma(y))$, which is no subset of $\mathcal{T}_\Delta$ because (σ, r) is trim. If $\sigma'(u) = *$ then

  $$r'(u)(\mathcal{C}_\Sigma(y)) = r(u)(\mathcal{C}_\Sigma(y)) \cdot \sigma(u) \subseteq \mathcal{T}_\Delta \; .$$

- It is proper: This follows directly by Property 2. New outputs under σ' are $*$ only. In addition, the number of outputs that are changed is finite. Thus, the infinite sets of equivalence classes with essential residual are only reduced by a finite number.

- The partition (σ', r') is earliest and path-finite: For essential residuals nothing changes (Property 1). Thus, their greatest common suffices remain y and the number of y-paths is the same as in r. The greatest common suffix of an inessential residual is $\bot$. The set of y-paths in the images is empty.

- The partition is unified earliest. Assume there exists a tree $u \in \mathrm{dom}(\sigma')$ with $\mathsf{mgu}(r'(u), r'(u')) = \langle \sigma'(u), z \rangle$ for some z. Then $r'(u)$ is not inessential and with Property 1, it equals $r(u)$ and $\sigma'(u) = \sigma(u)$. Assume $r'(u') = r(u')$, then $\langle \sigma'(u), z \rangle$ is also the most-general unifier of $r(u)$ and $r(u')$. However, then $\sigma'(u) = \sigma(u)$ is in Z and $\sigma'(u) = *$ cannot be part of the most-general unifier. Otherwise, if $r'(u') \neq r(u')$, it is inessential and we get for every $c \in \mathrm{dom}(r(u'))$:

  $$r(u)(c) \cdot \sigma(u) = r'(u)(c) \cdot \sigma'(u) = r'(u')(c) = r(u')(c) \cdot \sigma(u')$$

  Thus, $\mathsf{mgu}(r(u), r(u')) = \langle \sigma(u), \sigma(u') \rangle$ and with that, $\sigma(u)$ is in Z. This contradicts the assumption that $r'(u)$ is essential.

- The equivalence relation $\equiv_{r'}$ has finite index: Since r has finite index and we only get finitely many new residuals (Property 2), the index of r' is finite, too. □

We have seen on Page 61 that the partition (σ, r) in Example 5.8 is not unified earliest. In the following example, we apply the construction of the proof to that partition.

Example 5.9. Consider again the tree function τ_{mgu} of the Example 5.8. We want to define a unified earliest partition for this function. Continuing along the construction in the proof (Equations 5.8 and 5.9), we start with the partition (σ, r) of Example 5.8. The complete partition (σ, r), which is not unified earliest, was given in Figure 5.5.

The residual function r maps for arbitrary $n, m \in \mathbb{N}_0$ the trees $u_n = \mathtt{d}(\mathtt{b}^n(\mathtt{e}))$ and $u'_m = \mathtt{d}'(\mathtt{b}^m(\mathtt{e}))$ to unifiable residuals. As seen on Page 61, for those trees holds for every context $c \in \mathrm{dom}(r(u_n))$: $r(u_n)(c) \cdot \mathtt{B}(\mathtt{E}) = r(u'_m)(c) \cdot \mathtt{E}$. It follows

$$\mathsf{mgu}(r(u_n), r(u'_m)) = \langle \mathtt{B}(\mathtt{E}), \mathtt{E} \rangle \ .$$

Additionally, there is another non-trivial most-general unifier (of the residuals in the first and third lines of the table on Page 59):

$$\mathsf{mgu}(r(\mathtt{g}(\mathtt{d}(\mathtt{b}^n(\mathtt{e})))), r(\mathtt{g}(\mathtt{f}(\mathtt{b}^n(\mathtt{a}))))) = \langle \mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E})), \top \rangle$$

Thus, the set of subtrees of ground unifiers as described in the proof is

$$Z = \{\mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E})),\ \mathtt{D},\ \mathtt{B}(\mathtt{E}),\ \mathtt{E}\} \ .$$

To get a unified earliest partition, we have to change the output and the residual of every tree u with $\sigma(u) \in Z$. For instance, we have $\sigma(\mathtt{e}) = \mathtt{E}$. Thus, we get $\sigma'(\mathtt{e}) = *$ and the residual $r'(\mathtt{e})$ is defined by

$$\begin{aligned} \mathtt{d}(\mathtt{b}^k(y)) &\mapsto \mathtt{A}(\mathtt{B}^k(\mathtt{E}), \mathtt{E}) & \mathtt{d}'(\mathtt{b}^k(y)) &\mapsto \mathtt{A}(\mathtt{B}^{k+1}(\mathtt{E}), \mathtt{B}^k(\mathtt{E})) \\ \mathtt{g}(\mathtt{d}(\mathtt{b}^k(y))) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}^k(\mathtt{E})) & \mathtt{g}(\mathtt{d}'(\mathtt{b}^k(y))) &\mapsto \mathtt{G}(\mathtt{D}, \mathtt{B}^{k+1}(\mathtt{E})) \ . \end{aligned}$$

For the sake of completeness, we present the unified earliest partition in Figure 5.6. The proof provides that this partition is also path-finite and has finite index. ◁

5.2.5 Congruence Relation

Having a closer look at the partition constructed in the proof of Theorem 5.10, we observe that the equivalence relation $\equiv_{r'}$ is a congruence relation, i.e., for equivalent trees the same context results in equivalent trees (cf. Page 21). In general, this is not implicitly given if a partition is unified earliest. The following Example 5.10 presents a further partition for the tree function τ_{mgu} of Example 5.8. Even though this partition is unified earliest, path-finite, and has finite index, the partition's equivalence relation is not a congruence.

Example 5.10. Consider the tree function τ_{mgu} in Figure 5.4. The following partition $(\hat{\sigma}, \hat{r})$ is unified earliest, path-finite, and has finite index. It differs from the partition (σ, r) of Example 5.8 only for those trees, which realize a ground most-general unifier. In contrast to that, the partition (σ', r') of Example 5.9 never outputs any ground subtree of a most-general unifier, although if it is not involved.

u	$\sigma'(u)$	$r'(u)$	
		y	$\mathtt{g}(y)$
$\mathtt{g(d(b}^n\mathtt{(e)))}$	$\mathtt{G(D,B}^n\mathtt{(E))}$	y	
$\mathtt{g(d(b(e)))}$	$*$	$\mathtt{G(D,B(E))}$	
$\mathtt{g(d(e))}$	$\mathtt{G(D,E)}$	y	
$\mathtt{g(d'(b}^n\mathtt{(e)))}$	$\mathtt{G(D,B}^{n+1}\mathtt{(E))}$	y	
$\mathtt{g(d'(b(e)))}$	$\mathtt{G(D,B}^2\mathtt{(E))}$	y	
$\mathtt{g(d'(e))}$	$*$	$\mathtt{G(D,B(E))}$	
$\mathtt{g(f(b}^m\mathtt{(a)))}$	$*$	$\mathtt{G(D,B(E))}$	
$\mathtt{d(b}^n\mathtt{(e))}$	$\mathtt{B}^n\mathtt{(E)}$	$\mathtt{A}(y,\mathtt{E})$	$\mathtt{G(D},y\mathtt{)}$
$\mathtt{d(b(e))}$	$*$	$\mathtt{A(B(E),E)}$	$\mathtt{G(D,B(E))}$
$\mathtt{d(e)}$	$*$	$\mathtt{A(E,E)}$	$\mathtt{G(D,E)}$
$\mathtt{d'(b}^n\mathtt{(e))}$	$\mathtt{B}^n\mathtt{(E)}$	$\mathtt{A(B}(y),y\mathtt{)}$	$\mathtt{G(D,B}(y)\mathtt{)}$
$\mathtt{d'(b(e))}$	$*$	$\mathtt{A(B}^2\mathtt{(E),B(E))}$	$\mathtt{G(D,B}^2\mathtt{(E))}$
$\mathtt{d'(e)}$	$*$	$\mathtt{A(B(E),E)}$	$\mathtt{G(D,B(E))}$
$\mathtt{f(b}^m\mathtt{(a))}$	$*$	$\mathtt{A(B(E),E)}$	$\mathtt{G(D,B(E))}$

u	$\sigma'(u)$	$r'(u)$			
		$\mathtt{d(b}^k(y)\mathtt{)}$	$\mathtt{d'(b}^k(y)\mathtt{)}$	$\mathtt{g(d(b}^k(y)\mathtt{)))}$	$\mathtt{g(d'(b}^k(y)\mathtt{)))}$
$\mathtt{b}^n\mathtt{(e)}$	$\mathtt{B}^n\mathtt{(E)}$	$\mathtt{A(B}^k(y),\mathtt{E)}$	$\mathtt{A(B}^{k+1}(y),\mathtt{B}^k(y)\mathtt{)}$	$\mathtt{G(D,B}^k(y)\mathtt{)}$	$\mathtt{G(D,B}^{k+1}(y)\mathtt{)}$
$\mathtt{b(e)}$	$*$	$\mathtt{A(B}^{k+1}\mathtt{(E),E)}$	$\mathtt{A(B}^{k+2}\mathtt{(E),B}^{k+1}\mathtt{(E))}$	$\mathtt{G(D,B}^{k+1}\mathtt{(E))}$	$\mathtt{G(D,B}^{k+2}\mathtt{(E))}$
$\mathtt{e}$	$*$	$\mathtt{A(B}^k\mathtt{(E),E)}$	$\mathtt{A(B}^{k+1}\mathtt{(E),B}^k\mathtt{(E))}$	$\mathtt{G(D,B}^k\mathtt{(E))}$	$\mathtt{G(D,B}^{k+1}\mathtt{(E))}$

u	$\sigma'(u)$	$r'(u)$		
		$\mathtt{b}^k(y)$	$\mathtt{f(b}^k(y)\mathtt{)}$	$\mathtt{g(f(b}^k(y)\mathtt{)))}$
$\mathtt{b}^n\mathtt{(a)}$	$\mathtt{B}^n\mathtt{(E)}$	$\mathtt{B}^k(y)$	$\mathtt{A(B(E),E)}$	$\mathtt{G(D,B(E))}$
$\mathtt{b(a)}$	$*$	$\mathtt{B}^{k+1}\mathtt{(E)}$	$\mathtt{A(B(E),E)}$	$\mathtt{G(D,B(E))}$
$\mathtt{a}$	$*$	$\mathtt{B}^k\mathtt{(E)}$	$\mathtt{A(B(E),E)}$	$\mathtt{G(D,B(E))}$

Figure 5.6: The unified earliest partition (σ', r') of Example 5.9. ($m \in \mathbb{N}_0$, $n \geq 2$)

In comparison to the partition (σ, r), the residual and the output of $\mathtt{b}^n(\mathtt{e})$ and $\mathtt{b}^n(\mathtt{a})$ (for all $n \in \mathbb{N}_0$) does not change (Figure 5.5), i.e.,

$$\hat{r}(\mathtt{b}^n(\mathtt{e})) = r(\mathtt{b}^n(\mathtt{e})) \qquad \hat{\sigma}(\mathtt{b}^n(\mathtt{e})) = \mathtt{B}^n(\mathtt{E})$$
$$\hat{r}(\mathtt{b}^n(\mathtt{a})) = r(\mathtt{b}^n(\mathtt{a})) \qquad \hat{\sigma}(\mathtt{b}^n(\mathtt{a})) = \mathtt{B}^n(\mathtt{E}) \ .$$

Apart from that, the subtree function $\hat{\sigma}$ and the residual function $\hat{r}$ are defined by

		$\hat{r}(u)$		
u	$\hat{\sigma}(u)$	y	$\mathtt{g}(y)$	
$\mathtt{g}(\mathtt{d}(\mathtt{b}^n(\mathtt{e})))$	$\mathtt{G}(\mathtt{D}, \mathtt{B}^n(\mathtt{E}))$	y		$n \in \mathbb{N}_0$
$\mathtt{g}(\mathtt{d}'(\mathtt{b}^n(\mathtt{e})))$	$\mathtt{G}(\mathtt{D}, \mathtt{B}^{n+1}(\mathtt{E}))$	y		$n \in \mathbb{N}_0$
$\mathtt{g}(\mathtt{f}(\mathtt{b}^n(\mathtt{a})))$	$*$	$\mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E}))$		$n \in \mathbb{N}_0$
$\mathtt{d}(\mathtt{b}^n(\mathtt{e}))$	$\mathtt{B}^n(\mathtt{E})$	$\mathtt{A}(y, \mathtt{E})$	$\mathtt{G}(\mathtt{D},y)$	$n \neq 1$
$\mathtt{d}'(\mathtt{b}^n(\mathtt{e}))$	$\mathtt{B}^n(\mathtt{E})$	$\mathtt{A}(\mathtt{B}(y),y)$	$\mathtt{G}(\mathtt{D}, \mathtt{B}(y))$	$n \neq 0$
$\mathtt{d}(\mathtt{b}(\mathtt{e}))$	$*$	$\mathtt{A}(\mathtt{B}(\mathtt{E}), \mathtt{E})$	$\mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E}))$	
$\mathtt{d}'(\mathtt{e})$	$*$	$\mathtt{A}(\mathtt{B}(\mathtt{E}), \mathtt{E})$	$\mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E}))$	
$\mathtt{f}(\mathtt{b}^n(\mathtt{a}))$	$*$	$\mathtt{A}(\mathtt{B}(\mathtt{E}), \mathtt{E})$	$\mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E}))$	$n \in \mathbb{N}_0$

The two trees $\mathtt{e}$ and $\mathtt{b}(\mathtt{e})$ are equivalent. However, if we enhance the trees by the context $\mathtt{d}(y)$, we get non-equivalent trees $\mathtt{d}(\mathtt{e})$ and $\mathtt{d}(\mathtt{b}(\mathtt{e}))$. Thus, this partition is not congruent, but it is unified earliest, path-finite and has finite index. ◁

This example shows that congruence is not implied by the previous properties. However, the proof of Theorem 5.10 provides an equivalent partition with congruence relation for every path-finite partition with finite index. To prove this, we first define the image of a subcontext c of a context $c' \cdot c$ of a tree u. This (unique) image does not exists in every partition. We will show for every unified earliest, path-finite partition with finite index holds: If for every pair c, u there is a unique image of c, depending on c and u only, the equivalence relation is already a congruence relation (and vice versa).

Definition 5.8. Let (σ, r) be a partition, $u \in \mathrm{dom}(\sigma)$, and $c \in \mathcal{C}_\Sigma(y)$ with $c \cdot u \in \mathrm{dom}(\sigma)$. Let $Z_{u,c} \subseteq \mathcal{T}_\Delta(y)$ be the set of possible images of c w.r.t. u, i.e., $z \in Z_{u,c}$ iff for all contexts $c' \in \mathrm{dom}(r(c \cdot u))$ holds

$$r(c \cdot u)(c') \cdot z = r(u)(c' \cdot c) \ .$$

The *dedicated image* $\mathsf{di}(u)(c) \in \mathcal{T}_\Delta(y)$ of c w.r.t. u is then defined by:

- If $r(c \cdot u)$ is essential and $Z_{u,c}$ is a singleton, it is the unique image $z \in Z_{u,c}$.
- If $r(c \cdot u)$ is inessential and $Z_{u,c} \neq \emptyset$, it equals $*$.
- Otherwise, it is undefined.

Note that in this definition, it is not required that c is a context of u w.r.t. r, i.e., c is not necessarily in $\mathrm{dom}(r(u))$. Furthermore, $c \in \mathrm{dom}(r(u))$ does not imply $\mathsf{di}(u)(c) = r(u)(c)$.

If the dedicated image of $c \in \mathcal{C}_\Sigma(y)$ w.r.t. $u \in \mathrm{dom}(\sigma)$ is defined, the following holds for all contexts $c' \in \mathrm{dom}(r(c \cdot u))$:

$$r(c \cdot u)(c') \cdot \mathsf{di}(u)(c) = r(u)(c' \cdot c) \tag{5.10}$$

Example 5.11. Consider again the tree function in Figure 5.4 and the partition (σ', r') in Example 5.9: For all trees $u \in \mathrm{dom}(\sigma')$, $c \in \mathcal{C}_\Sigma(y)$ with $c \cdot u \in \mathrm{dom}(\sigma')$ the dedicated image $\mathsf{di}(u)(c)$ with respect to the partition (σ', r') in Example 5.9 exists. Consider for instance $u = \mathtt{b}(\mathtt{e})$ and $c = \mathtt{d}(y)$. The residual $r'(\mathtt{d}(\mathtt{b}(\mathtt{e})))$ is inessential and for every $c' \in \mathrm{dom}(r'(\mathtt{b}(\mathtt{e})))$ holds the Equation 5.10:

$$r'(\mathtt{d}(\mathtt{b}(\mathtt{e})))(y) \cdot * = \mathtt{A}(\mathtt{B}(\mathtt{E}), \mathtt{E}) = r'(\mathtt{b}(\mathtt{e}))(\mathtt{d}(y))$$
$$r'(\mathtt{d}(\mathtt{b}(\mathtt{e})))(\mathtt{g}(y)) \cdot * = \mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E})) = r'(\mathtt{b}(\mathtt{e}))(\mathtt{g}(\mathtt{d}(y)))$$

Thus, the dedicated image $\mathsf{di}(\mathtt{e})(\mathtt{b}(y))$ equals $*$.

With respect to the partition $(\hat{\sigma}, \hat{r})$ of Example 5.10, the dedicated image does not exist for every pair u, c. Consider the same pair: $u = \mathtt{b}(\mathtt{e})$ and $c = \mathtt{d}(y)$. For every z holds:

$$\hat{r}(\mathtt{d}(\mathtt{b}(\mathtt{e})))(y) \cdot z = \mathtt{A}(\mathtt{B}(\mathtt{E}), \mathtt{E}) \neq \mathtt{A}(y, \mathtt{E}) = \hat{r}(\mathtt{b}(\mathtt{e}))(\mathtt{d}(y))$$
$$\hat{r}(\mathtt{d}(\mathtt{b}(\mathtt{e})))(\mathtt{g}(y)) \cdot z = \mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E})) \neq \mathtt{G}(\mathtt{D}, y) = \hat{r}(\mathtt{b}(\mathtt{e}))(\mathtt{g}(\mathtt{d}(y))) \qquad \triangleleft$$

This example shows that not for every unified earliest, path-finite partition with finite index the dedicated images are defined. However, another property of the counterexample partition $(\hat{\sigma}, \hat{r})$ is that its equivalence relation is not a congruence (cf. Example 5.10). We will show that we can generalize this observation, i.e., for every unified earliest, path-finite partition with finite index, these properties are equivalent.

Lemma 5.11. *For every unified earliest and path-finite partition (σ, r) with finite index, the following are equivalent*

1. *The equivalence $\equiv_r$ is a congruence relation.*
2. *For all $u \in dom(\sigma)$ and every $c \in \mathcal{C}_\Sigma(y)$ with $c \cdot u \in dom(\sigma)$, the dedicated image $\mathsf{di}(u)(c)$ is defined.*

Proof. Let τ be the tree function defined by (σ, r). First, assume that $\equiv_r$ is a congruence, i.e., for all $c \in \mathcal{C}_\Sigma(y)$ with $c \cdot u \in \mathrm{dom}(\sigma)$ holds:

$$u \equiv_r u' \implies c \cdot u \equiv_r c \cdot u' \,.$$

Let $u \in \mathrm{dom}(\sigma)$ and $c \in \mathcal{C}_\Sigma(y)$ with $c \cdot u \in \mathrm{dom}(\sigma)$. For every $c' \in \mathrm{dom}(r(c \cdot u))$ holds

$$r(c \cdot u)(c') \cdot \sigma(c \cdot u) = r(u)(c' \cdot c) \cdot \sigma(u) \,.$$

Assume $r(c \cdot u)$ is inessential but $r(u)$ is essential. Then there is a tree $u' \equiv_r u$ with $\sigma(u') \neq \sigma(u)$. Furthermore, it holds $c \cdot u \equiv_r c \cdot u'$. Thus, we get for all $c' \in \mathrm{dom}(r(c \cdot u))$

$$\begin{aligned} r(u)(c' \cdot c) \cdot \sigma(u) &= r(c \cdot u)(c') \\ &= r(c \cdot u')(c') \\ &= r(u')(c' \cdot c) \cdot \sigma(u') \\ &= r(u)(c' \cdot c) \cdot \sigma(u') \ . \end{aligned}$$

Since $\sigma(u) \neq \sigma(u')$, for all $c' \in \mathrm{dom}(r(c \cdot u))$ holds $r(u')(c' \cdot c) = r(u)(c' \cdot c) \in \mathcal{T}_\Delta$. It follows that for all $c' \in \mathrm{dom}(r(c \cdot u))$ holds $r(c \cdot u)(c') \cdot * = r(u)(c' \cdot c)$. Thus, the dedicated image $\mathsf{di}(u)(c)$ equals $*$. Analogously, $\mathsf{di}(u)(c) = *$ follows also if $r(c \cdot u)$ and $r(u)$ are inessential.

Now assume $r(c \cdot u)$ is essential. If $r(u)$ is inessential, it follows for all $c' \in \mathrm{dom}(r(c \cdot u))$

$$r(c \cdot u)(c') \cdot \sigma(c \cdot u) = r(u)(c' \cdot c) \ .$$

The dedicated image is defined, i.e., $\mathsf{di}(u)(c) = \sigma(c \cdot u)$. Thus, assume $r(u)$ and $r(c \cdot u)$ are essential. It follows that there is a tree $u' \equiv_r u$ with $\sigma(u) \neq \sigma(u')$. Since the relation is congruent, also $c \cdot u' \equiv_r c \cdot u$. We get

$$\begin{aligned} r(c \cdot u)(c') \cdot \sigma(c \cdot u) &= r(u)(c' \cdot c) \cdot \sigma(u) \\ r(c \cdot u)(c') \cdot \sigma(c \cdot u') &= r(u)(c' \cdot c) \cdot \sigma(u') \end{aligned}$$

By Proposition 2.1, there are three possibilities. Since $r(c \cdot u)$ is essential, there is a context c' with $r(c \cdot u)(c') \notin \mathcal{T}_\Delta$. Thus, there is a tree $s_{c'} \in \hat{\mathcal{T}}_\Delta(y)$ with

$$r(c \cdot u)(c') \cdot s_{c'} = r(u)(c' \cdot c) \tag{5.11}$$

$$\text{or} \qquad r(c \cdot u)(c') = r(u)(c' \cdot c) \cdot s_{c'} \tag{5.12}$$

If the first equation holds, it follows that $s_{c'} \cdot \sigma(u) = \sigma(c \cdot u)$ and with that, for all context $\bar{c}$ holds the equivalent equation. Thus, $s_{c'} = \mathsf{di}(u)(c)$ is unique. If Equation 5.12 holds, then it holds consequently for all contexts $\bar{c}$ of $c \cdot u$ for the same tree $s_{c'}$. Since the partition is earliest, $s_{c'} = y$ and with that, we can state

$$r(c \cdot u)(c') \cdot y = r(u)(c' \cdot c) \ .$$

The dedicated image then is $\mathsf{di}(u)(c) = y$.

Now assume that Property 2. of the lemma holds, i.e., the dedicated images are defined. We show that $\equiv_r$ is a congruence. Assume the opposite. Then there are subtrees $u, u' \in \mathrm{dom}(\sigma)$ with $u \equiv_r u'$ and $c \in \mathcal{C}_\Sigma(y)$ with $c \cdot u \in \mathrm{dom}(r)$ but

1. $c \cdot u' \notin \mathrm{dom}(r)$ or
2. $c \cdot u \not\equiv_r c \cdot u'$.

First, assume there exists such a triple u, u', c with $c \cdot u' \notin \mathrm{dom}(r)$. Since the partition is trim, there is a context $c' \in \mathrm{dom}(r(c \cdot u))$. Thus, $c' \cdot c$ is a context

of u, i.e., $c' \cdot c \in \mathrm{dom}(r(u)) = \mathrm{dom}(r(u'))$. However, then $c' \cdot c \cdot u'$ is an input tree of τ. Thus, if $u \equiv_r u'$ and $c \cdot u \in \mathrm{dom}(r)$ then $c \cdot u' \in \mathrm{dom}(r)$, too.

Now assume there is a triple u, u', c, which contradicts the second condition, i.e., it exists a context $c' \in \mathrm{dom}(r(c \cdot u)) = \mathrm{dom}(r(c \cdot u'))$ with different images $r(c \cdot u)(c') \neq r(c \cdot u')(c')$. Since for every context c and every tree u with $c \cdot u \in \mathrm{dom}(\sigma)$, the dedicated image $\mathsf{di}(u)(c)$ exists, we get for u, u' that for all $\bar{c} \in \mathrm{dom}(r(c \cdot u))$ holds

$$r(c \cdot u)(\bar{c}) \cdot \mathsf{di}(u)(c) = r(u)(\bar{c} \cdot c)$$
$$r(c \cdot u')(\bar{c}) \cdot \mathsf{di}(u')(c) = r(u')(\bar{c} \cdot c)$$

Since $u \equiv_r u'$ we get for all $\bar{c} \in \mathrm{dom}(r(c \cdot u))$

$$r(c \cdot u)(\bar{c}) \cdot \mathsf{di}(u)(c) = r(c \cdot u')(\bar{c}) \cdot \mathsf{di}(u')(c)$$

Thus, $\langle \mathsf{di}(u)(c), \mathsf{di}(u')(c) \rangle$ is a unifier of $r(c \cdot u), r(c \cdot u')$. The most-general unifier of these residuals then is either

- $\langle y, y \rangle$, $\langle \top, \top \rangle$, or
- $\langle \mathsf{di}(u)(c), \mathsf{di}(u')(c) \rangle$, $\langle \mathsf{di}(u)(c), \top \rangle$, or $\langle \top, \mathsf{di}(u')(c) \rangle$ with ground dedicated images $\mathsf{di}(u)(c) \in \mathcal{T}_\Delta$ and $\mathsf{di}(u')(c) \in \mathcal{T}_\Delta$, respectively.

In the first case, the residuals are equivalent, i.e., $c \cdot u \equiv_r c \cdot u'$. In the second case, assume (w.l.o.g.) the dedicated image $\mathsf{di}(u)(c) \in \mathcal{T}_\Delta$ is a ground term and $\mathsf{mgu}(r(c \cdot u), r(c \cdot u')) = \langle \mathsf{di}(u)(c), z' \rangle$ with $z' \in \{\mathsf{di}(u')(c), \top\}$. It follows:

$$r(c \cdot u)(\bar{c}) \cdot \mathsf{di}(u)(c) = r(u)(\bar{c} \cdot c) \cdot \sigma(u) = r(c \cdot u)(\bar{c}) \cdot \sigma(c \cdot u)$$

Thus, $\mathsf{di}(u)(c) = \sigma(c \cdot u)$, which is realizable in $r(c \cdot u)$. This contradicts the fact that (σ, r) is unified earliest. — The equivalence relation is a congruence. □

Finally, by this lemma and Theorem 5.10 we get for every earliest, path-finite partition with finite index an equivalent partition, which is unified earliest, path-finite and defines a *congruence* relation of finite index. This partition is the one given in the proof of Theorem 5.10. We will show that for this partition the dedicated images exist.

Theorem 5.12. *For every earliest and path-finite partition (σ, r) with finite index, there exists a partition (σ', r') such that*

- *(σ', r') is unified earliest and path-finite*
- *$\equiv_{r'}$ is a congruence of finite index.*

Proof. Let τ be the tree function defined by (σ, r) and let (σ', r') be the partition defined in Equations 5.8 and 5.9 in the proof of Theorem 5.10. In that proof, we have already shown that this partition is unified earliest and path-finite. It is also proven that the index of the equivalence relation $\equiv_{r'}$ is finite. It remains to proof that the equivalence relation is already a congruence. Thereto, we show

that for all $u \in \mathrm{dom}(\sigma')$ and every $c \in \mathcal{C}_\Sigma(y)$ with $c \cdot u \in \mathrm{dom}(\sigma')$ the dedicated image $\mathsf{di}(u)(c)$ is defined. Then it follows with Lemma 5.11 that the equivalence relation $\equiv_{r'}$ is a congruence.

For every $u \in \mathrm{dom}(\sigma')$, $c \in \mathcal{C}_\Sigma(y)$ with $c \cdot u \in \mathrm{dom}(\sigma')$, and every context $c' \in \mathrm{dom}(r'(c \cdot u))$, we know

$$r'(c \cdot u)(c') \cdot \sigma'(c \cdot u) = r'(u)(c' \cdot c) \cdot \sigma'(u) \; . \tag{5.13}$$

Thus, let $u \in \mathrm{dom}(\sigma')$ and $c \in \mathcal{C}_\Sigma(y)$ with $c \cdot u \in \mathrm{dom}(\sigma')$. We consider the different possibilities whether the residuals $r'(c \cdot u)$ and $r'(u)$ are each essential or inessential.

- First, assume for all $c' \in \mathrm{dom}(r'(c \cdot u))$ the image of $c' \cdot c$ under $r'(u)$ is ground, i.e., $r'(u)(c' \cdot c) \in \mathcal{T}_\Delta$. Then $r'(c \cdot u)(c') \cdot \sigma'(c \cdot u) = r'(u)(c' \cdot c)$. If $r'(c \cdot u)$ is essential, we get with $\mathsf{di}(u)(c) = \sigma'(c \cdot u)$ the dedicated image of c w.r.t. u. Otherwise, if the residual $r'(c \cdot u)$ is inessential, we get $r'(c \cdot u)(c') = r'(u)(c' \cdot c)$ for all $c' \in \mathrm{dom}(r'(c \cdot u))$ and define the dedicated image by $\mathsf{di}(u)(c) = *$.

- Now assume $r'(c \cdot u)$ is essential and it exists $c' \in \mathrm{dom}(r'(c \cdot u))$ with $r'(u)(c' \cdot c) \in \hat{\mathcal{T}}_\Delta(y)$. Since the partition is proper, there are infinitely many different outputs of equivalent trees, i.e., $|\sigma'([u]_{r'})| = \infty$. The number of equivalence classes is finite. Thus, there are two trees $u_1, u_2 \in [u]_{r'}$ with different outputs for which $c \cdot u_1$ and $c \cdot u_2$ are equivalent. Summarizing, we have

  - $u_1 \equiv_{r'} u_2 \equiv_{r'} u$
  - $\sigma'(u_1) \neq \sigma'(u_2)$
  - $c \cdot u_1 \equiv_{r'} c \cdot u_2$.

  We get with Equation 5.13

  $$\begin{aligned} r'(c \cdot u_1)(c') \cdot \sigma'(c \cdot u_1) &= r'(u_1)(c' \cdot c) \cdot \sigma'(u_1) \\ r'(c \cdot u_2)(c') \cdot \sigma'(c \cdot u_2) &= r'(u_2)(c' \cdot c) \cdot \sigma'(u_2) \; . \end{aligned}$$

  Since the residuals of u_1 and u_2 are equivalent and so are the residuals of $c \cdot u_1$ and $c \cdot u_2$, it follows:

  - $\sigma'(c \cdot u_1) \neq \sigma'(c \cdot u_2)$

  By Proposition 2.1, there are two possibilities:

  $$r'(c \cdot u_1)(c') \cdot z = r'(u_1)(c' \cdot c) \tag{5.14}$$

  $$\text{or} \qquad r'(c \cdot u_1)(c') = r'(u_1)(c' \cdot c) \cdot z \tag{5.15}$$

  for some $z \in \hat{\mathcal{T}}_\Delta(y)$. Case (a) of the Proposition 2.1, i.e., $r'(u_1)(c' \cdot c)$ and $r'(u)(c' \cdot c)$ are equal and ground, is impossible because we assumed $r'(u_1)(c' \cdot c) = r'(u)(c' \cdot c) \in \hat{\mathcal{T}}_\Delta(y)$.

If Equation 5.14 holds, for both $i \in \{1,2\}$ follows $z \cdot \sigma'(u_i) = \sigma'(c \cdot u_i)$. Then Proposition 2.1 provides that for all $\bar{c} \in \mathrm{dom}(r'(c \cdot u_i))$ the same equation as for c' holds:

$$r'(c \cdot u_1)(\bar{c}) \cdot z = r'(u_1)(\bar{c} \cdot c)$$

If Equation 5.15 holds, it follows analogically for both $i \in [2]$ and all $\bar{c} \in \mathrm{dom}(r'(c \cdot u_i))$:

- $\sigma'(u_i) = z \cdot \sigma'(c \cdot u_i)$
- $r'(c \cdot u_1)(\bar{c}) = r'(u_1)(\bar{c} \cdot c) \cdot z$.

First, assume that Equation 5.14 holds. The trees u_1 and u are equivalent. This implies, $r'(c \cdot u_1)(\bar{c}) \cdot z = r'(u)(\bar{c} \cdot c)$ holds for all $\bar{c} \in \mathrm{dom}(r'(c \cdot u_1))$. Thus, for all contexts $\bar{c} \in \mathrm{dom}(r'(c \cdot u_1))$ holds

$$r'(c \cdot u)(\bar{c}) \cdot \sigma'(c \cdot u) = r'(u)(\bar{c} \cdot c) \cdot \sigma'(u) = r'(c \cdot u_1)(\bar{c}) \cdot z \cdot \sigma'(u) \ .$$

Hence, $r'(c \cdot u)$ and $r'(c \cdot u_1)$ are unifiable. Since $r'(c \cdot u)$ is essential and $\sigma'(c \cdot u)$ is realizable in $r'(c \cdot u)$, the most-general unifier is

$$\mathsf{mgu}(r'(c \cdot u), r'(c \cdot u_1)) = \langle y, y \rangle \ .$$

However, that means that the two trees $c \cdot u$ and $c \cdot u_1$ are equivalent and it follows for all $\bar{c} \in \mathrm{dom}(r'(c \cdot u))$

$$r'(c \cdot u)(\bar{c}) \cdot z = r'(c \cdot u_1)(\bar{c}) \cdot z = r'(u_1)(\bar{c} \cdot c) = r'(u_1)(\bar{c} \cdot c) \ .$$

With $\mathsf{di}(u)(c) = z$ we get the dedicated image of c under u.

On the other hand, assume that the second equation (Equation 5.15) holds, i.e., for all $\bar{c} \in \mathrm{dom}(r'(c \cdot u_i))$ holds

$$r'(c \cdot u_1)(\bar{c}) = r'(u_1)(\bar{c} \cdot c) \cdot z \ .$$

Since the (σ', r') is earliest and z is a common suffix of all images in $r'(c \cdot u_1)$, it follows that $z = y$. Again, for all contexts $\bar{c} \in \mathrm{dom}(r'(c \cdot u_1))$ holds

$$r'(c \cdot u)(\bar{c}) \cdot \sigma'(c \cdot u) = r'(u)(\bar{c} \cdot c) \cdot \sigma'(u) = r'(c \cdot u_1)(\bar{c}) \cdot \sigma'(u) \ .$$

Thus, $r'(c \cdot u)$ and $r'(c \cdot u_1)$ are unifiable. Analogically as in the first case, it follows that $r'(c \cdot u)(\bar{c}) \cdot \mathsf{di}(u)(c) = r'(u)(\bar{c} \cdot c)$ with $\mathsf{di}(u)(c) = y$.

- Finally, assume $r'(c \cdot u)$ is inessential and it exists $c' \in \mathrm{dom}(r'(c \cdot u))$ with $r'(u)(c' \cdot c) \in \hat{\mathcal{T}}_\Delta(y)$. Then $r'(u)$ is essential and there are infinitely many trees $u' \equiv_{r'} u$ with $\sigma'(u') \neq \sigma'(u)$.

Assume the residual $r'(c \cdot u')$ is inessential for all these u'. With Equation 5.13 follows

$$\begin{aligned} r'(c \cdot u)(c') &= r'(u)(c' \cdot c) \cdot \sigma'(u) \\ &\neq r'(u')(c' \cdot c) \cdot \sigma'(u') \\ &= r'(c \cdot u')(c') \ . \end{aligned}$$

Thus, $c \cdot u \not\equiv_{r'} c \cdot u'$. Since the partition has finite index, there is a tree $u' \in [u]_{r'}$ with essential $r'(c \cdot u')$ and $\sigma'(u') \neq \sigma'(u)$. It follows by the previous parts of this proof that the dedicated images $\mathsf{di}(u')(c)$ exist with

$$r'(c \cdot u')(c') \cdot \mathsf{di}(u')(c) = r'(u')(c' \cdot c)$$

for all $c' \in \mathrm{dom}(r'(c \cdot u'))$. Now we get with Equation 5.13 and the equivalence of u and u' that for all $c' \in \mathrm{dom}(r'(c \cdot u))$ holds:

$$r'(c \cdot u')(c') \cdot \mathsf{di}(u')(c) \cdot \sigma'(u) = r'(u')(c' \cdot c) \cdot \sigma'(u') = r'(c \cdot u)(c')$$

Thus, the most-general unifier of the residuals of $c \cdot u'$ and $c \cdot u$ is

$$\mathsf{mgu}(r'(c \cdot u'), r'(c \cdot u)) = \langle \mathsf{di}(u')(c) \cdot \sigma'(u), \top \rangle \ .$$

Then, a subtree of a ground most-general unifier is realizable in some tree (either $\sigma'(u)$ or $\mathsf{di}(u')(c)$ equals $\sigma'(u')$). This does not contradicts the property earliest unified, but the partition (σ', r') in the proof of Theorem 5.10 has no such most-general unifier by construction. □

5.2.6 Minimal Partition

Let τ be a tree function with a partition, which is path-finite and has finite index and let (σ, r) be a partition of τ, which is path-finite and has finite index, but additionally it is unified earliest and $\equiv_r$ is a congruence. We have seen that such a partition exists. Unfortunately, this partition is not necessarily unique. For instance, for the partition in Figure 5.6, which fulfills these properties, we can increase the index without suffering the loss of one of the previous properties. If we change the definition such that $\mathsf{a}, \mathsf{b}(\mathsf{a})$, and $\mathsf{b}^n(\mathsf{a})$ for all $n \in \mathbb{N}$ are equivalent, the properties are preserved, but the index is smaller.

In this section, we show that the partition for τ with minimal index, which preserves all afore introduced properties, is unique. To prove that, let $\sim'_r$ denote the smallest equivalence relation over $\mathrm{dom}(r)$ with the following properties:

- If $\mathsf{mgu}(r(u), r(u')) = \langle y, y \rangle$ or $\mathsf{mgu}(r(u), r(u')) = \langle \top, \top \rangle$ then $u \sim'_r u'$;
- Assume that $\mathsf{mgu}(r(u), r(u_1)) = \langle \top, v_1 \rangle$ for some ground term $v_1 \in \mathcal{T}_\Delta$. If for all u_2 with $\mathsf{mgu}(r(u), r(u_2)) = \langle \top, v_2 \rangle$ for some ground term $v_2 \in \mathcal{T}_\Delta$, $\mathsf{mgu}(r(u_1), r(u_2)) = \langle y, y \rangle$ holds then $u \sim'_r u_1$.

The relation $\sim_r$ is the greatest congruence relation, which is a refinement of $\sim'_r$, i.e., $u_1 \sim_r u_2$ whenever for every $c \in \mathcal{C}_\Sigma(y)$ holds $c \cdot u_1 \sim'_r c \cdot u_2$. Two trees $u_1, u_2 \in \mathcal{T}_\Sigma$ are called *similar* w.r.t. r if $u_1 \sim_r u_2$.

Definition 5.9 (Minimal)**.** A unified earliest partition (σ, r) is called *minimal* if similar trees are equivalent, i.e., $\equiv_r$ and $\sim_r$ define the same congruence relation.

If the most-general unifier is known for every pair of residuals, a minimal partition for τ can be defined.

Theorem 5.13. *For every unified earliest and path-finite partition (σ, r) with finite index and $\equiv_r$ is a congruence, there exists an equivalent partition (σ', r'), which is*

- *minimal,*
- *path-finite, and*
- *$\equiv_{r'}$ is a congruence relation of finite index.*

Additionally, the index of (σ', r') is less or equal the index of (σ, r).

Proof. Let τ be the tree function of (σ, r). We define the partition (σ', r') on the same domain as follows. Let $u \in \mathrm{dom}(r)$. Assume there is a tree $u' \in \mathrm{dom}(r)$ with $u' \sim_r u$ and $\mathsf{mgu}(r(u'), r(u)) = \langle v', \top \rangle$ for some $v' \in \mathcal{T}_\Delta$. Note that this implies that $r(u)$ is inessential. We define

$$\sigma'(u) = v' \quad \text{and} \quad r'(u)(c) = r(u')(c) \tag{5.16}$$

Otherwise, if no such $u' \sim_r u$ with non trivial most-general unifier exists, the output and the residual of u do not change, i.e.,

$$\sigma'(u) = \sigma(u) \quad \text{and} \quad r'(u)(c) = r(u)(c) \tag{5.17}$$

Let $u \in \mathrm{dom}(r)$ and $c \in \mathrm{dom}(r(u))$. If there is a tree $u' \in \mathrm{dom}(r)$ with $u' \sim_r u$ and $\mathsf{mgu}(r(u'), r(u)) = \langle v', \top \rangle$ for some $v' \in \mathcal{T}_\Delta$, then Equation 5.16 provides

$$r'(u)(c) \cdot \sigma'(u) = r(u')(c) \cdot v' = r(u)(c) = r(u)(c) \cdot \sigma(u) = \tau(c \cdot u) \; .$$

Otherwise, we get by Equation 5.17

$$r'(u)(c) \cdot \sigma'(u) = r(u)(c) \cdot \sigma(u) = \tau(c \cdot u) \; .$$

Thus, the pair is a partition of τ. We observe the following properties for all trees $u, u' \in \mathrm{dom}(r)$:

1. If $r(u)$ is essential, then $r'(u)$ is essential and equals $r(u)$.
2. Contrapositive, if $r'(u)$ is inessential, then $r(u)$ is inessential.
3. If $u' \sim_r u$, then u becomes r'-equivalent to u', i.e., $u \equiv_{r'} u'$.
4. No new residuals are generated, i.e., there exists a tree $u'' \in \mathrm{dom}(r)$ such that $r'(u) = r(u'')$. In particular, $u \sim_r u''$.

First, we show that the partition is minimal. Thereto, we prove, on the one hand, that $\equiv_{r'}$ is a congruence. Then, $\equiv_{r'}$ is a refinement of $\sim'_{r'}$ (cf. Page 72) and with that $\equiv_{r'} \subseteq \sim_{r'}$.

Let $u_1 \equiv_{r'} u_2$. The most-general unifier of $r'(u_1)$ and $r'(u_2)$ is either $\langle \top, \top \rangle$ or $\langle y, y \rangle$. In the first case, both residuals are inessential and with Property 2, we get also $\mathsf{mgu}(r(u_1), r(u_2)) = \langle \top, \top \rangle$. That implies $u_1 \equiv_r u_2$. The equivalence $\equiv_r$ is a congruence. Hence, $u_1 \sim_r u_2$. Now assume $\mathsf{mgu}(r'(u_1), r'(u_2)) = \langle y, y \rangle$. Property 4 provides that there are trees u'_1, u'_2 with $r(u'_i) = r'(u_i)$ for $i \in [2]$. Thus, we get

$$u_1 \sim_r u'_1 \equiv_r u'_2 \sim_r u_2 \ .$$

The equivalence $\equiv_r$ is a congruence. Hence, $u'_1 \sim_r u'_2$ and with that $u_1 \sim_r u_2$. Thus, in both cases, we get that u_1 and u_2 are similar w.r.t. r. Since $\sim_r$ is a congruence, it follows that for all $c \in \mathcal{C}_\Sigma(y)$ holds $c \cdot u_1 \sim_r c \cdot u_2$. In addition, Property 3 implies that for all $c \in \mathcal{C}_\Sigma(y)$ holds $c \cdot u_1 \equiv_{r'} c \cdot u_2$.

On the other hand, we have to show that $\sim_{r'} \subseteq \equiv_{r'}$. First, we show that for all trees u_1, u_2, which are similar in (σ', r'), holds $u_1 \sim'_r u_2$, i.e:

$$u_1 \sim_{r'} u_2 \implies u_1 \sim'_r u_2 \ . \tag{5.18}$$

Thereto, let $u_1 \sim_{r'} u_2$. The most-general unifier is

1. $\mathsf{mgu}(r'(u_1), r'(u_2)) = \langle \top, \top \rangle$,
2. $\mathsf{mgu}(r'(u_1), r'(u_2)) = \langle y, y \rangle$,
3. $\mathsf{mgu}(r'(u_1), r'(u_2)) = \langle v_1, \top \rangle$ for some $v_1 \in \mathcal{T}_\Delta$, or
4. $\mathsf{mgu}(r'(u_1), r'(u_2)) = \langle \top, v_2 \rangle$ for some $v_2 \in \mathcal{T}_\Delta$.

In the first case, also $\mathsf{mgu}(r(u_1), r(u_2)) = \langle \top, \top \rangle$ (by Property 2) and with that $u_1 \sim'_r u_2$. In the second case, we know by Property 3 that there is a tree u_3 such that $r(u_3) = r'(u_1) = r'(u_2)$ and $u_1 \sim_r u_3 \sim_r u_2$.

Let us now assume the most-general unifier equals $\langle v_1, \top \rangle$ for some ground term $v_1 \in \mathcal{T}_\Delta$. Additionally, for every tree u_3 holds (by definition of $\sim'_{r'}$)

$$\mathsf{mgu}(r'(u_3), r'(u_2)) = \langle v_3, \top \rangle \text{ with } v_3 \in \mathcal{T}_\Delta \implies u_1 \sim_{r'} u_3 \ . \tag{5.19}$$

Since u_1 and u_3 have both essential residuals under r', they must be r'-equivalent. For u_2 holds $r'(u_2) = r(u_2)$ is inessential (Property 2).

First, assume $r(u_1) = r'(u_1)$ is essential. Then $\mathsf{mgu}(r(u_1), r(u_2)) = \langle v_1, \top \rangle$ and for all u_3 with $\mathsf{mgu}(r(u_3), r(u_2)) = \langle v_3, \top \rangle$ and $v_3 \in \mathcal{T}_\Delta$ holds with Property 1 also $\mathsf{mgu}(r'(u_3), r'(u_2)) = \langle v_3, \top \rangle$. It follows with Equation 5.19 that

$$r(u_3) = r'(u_3) = r'(u_1) = r(u_1) \ .$$

Thus, $u_1 \sim'_r u_2$ (by definition of $\sim'_r$).

Otherwise, assume $r(u_1)$ is inessential and different to $r'(u_1)$. There is some tree u_4 with $r'(u_1) = r(u_4)$ (Property 4) and $\mathsf{mgu}(r(u_1), r(u_4)) = \langle \top, v_4 \rangle$ for some $v_4 \in \mathcal{T}_\Delta$ and $u_4 \sim_r u_1$. With Property 1 we get $r'(u_4) = r'(u_1)$. Then

$$\mathsf{mgu}(r'(u_4), r'(u_2)) = \langle v_1, \top \rangle$$

for u_4 with essential residual $r(u_4)$. Analog to the last assumption, we get $u_4 \sim'_r u_2$. It follows $u_1 \sim'_r u_4 \sim'_r u_2$.

The last case that the most-general unifier equals $\langle \top, v_2 \rangle$ for some $v_2 \in \mathcal{T}_\Delta$ is analog. Summarizing, we get Implication 5.18 for all trees u_1, u_2. Since $\sim_{r'}$ is a congruence, we get:

$$\begin{aligned}
u_1 \sim_{r'} u_2 &\implies \text{for all } c \in \mathcal{C}_\Sigma(y) :\ c \cdot u_1 \sim_{r'} c \cdot u_2 && \text{(Congruence)} \\
&\implies \text{for all } c \in \mathcal{C}_\Sigma(y) :\ c \cdot u_1 \sim'_r c \cdot u_2 && \text{(Implication 5.18)} \\
&\implies u_1 \sim_r u_2 \\
&\implies u_1 \equiv_{r'} u_2 && \text{(Property 3)}
\end{aligned}$$

It remains to prove that the partition is unified earliest. Then, we get that the partition is minimal. Thus, we have to show that (σ', r') is a trim, proper, earliest, unified earliest, and path-finite partition, and has finite index.

- It is trim and proper: We only have to consider trees where the residual under r' differs from the residual under r. In that case, it equals a residual of a tree u', i.e., $r'(u) = r(u') = r'(u')$ (Property 1), which is essential. Thus, $\sigma'(u) = s' \neq *$ implies that $r'(u)$ is essential. — It is trim.

  In addition, for every essential residual, the set of outputs only increases, i.e., $\sigma([u]_r) \subseteq \sigma'([u]_{r'})$. — It is proper.

- The partition is earliest, path-finite, and has finite index: With Property 4, these properties are directly inherited from (σ, r).

- It is unified earliest: Assume there is a unifiable pair of residuals in the range of r' with realizable ground unifier, i.e., there are trees u_1, u_2 with $\mathsf{mgu}(r'(u_1), r'(u_2)) = \langle v_1, v_2 \rangle$ where $\sigma'(u_1) = v_1 \in \mathcal{T}_\Delta$ is realizable in $r'(u_1)$ and $v_2 \in \mathcal{T}_\Delta \cup \{\top\}$. Property 4 implies that there are trees u'_1, u'_2 in $\mathrm{dom}(r)$ with $r(u'_i) = r'(u_i)$ and $u_i \sim_r u'_i$ (for $i \in \{1, 2\}$). Thus, for the most-general unifier we get

  $$\mathsf{mgu}(r(u'_1), r(u'_2)) = \langle v_1, v_2 \rangle \ .$$

  Since (σ, r) is unified earliest, $u_1 \not\equiv_r u'_1$. Otherwise, $\sigma(u_1) = \sigma'(u_1) = v_1$ would be realizable in $r(u_1)$. Hence, the residual $r(u'_1)$ is inessential and we get $\mathsf{mgu}(r(u_1), r(u'_1)) = \langle \top, v_1 \rangle$. Additionally, we have for the residuals of u_1 and u'_2: $\mathsf{mgu}(r(u_1), r(u'_2)) = \langle \top, v_2 \rangle$. If $v_2 = \top$, $u_1 \sim_r u'_2 \sim_r u_2$ and with Property 3: $u_1 \equiv_{r'} u_2$. This contradicts the assumption that the most-general unifier $\mathsf{mgu}(r'(u_1), r'(u_2))$ is different from $\langle y, y \rangle$ and $\langle \top, \top \rangle$. Otherwise, if $v_2 \neq \top$, we get that $u'_1 \sim_r u'_2$ (Definition of $\sim_r$). It follows

with Property 3: $u_1' \equiv_r u_2'$. Since $r(u_1')$ and $r(u_2')$ are essential, we also have (with Property 1) $u_1' \equiv_{r'} u_2'$. Furthermore, we know for both $i \in [2]$ that u_i' is r'-equivalent to u_i, i.e., we get

$$u_1 \equiv_{r'} u_1' \equiv_{r'} u_2' \equiv_{r'} u_2 \ .$$

Again, this contradicts the assumption for $\mathsf{mgu}(r'(u_1), r'(u_2))$. □

In Example 5.9 we have defined a unified earliest partition for the example transformation τ_{mgu}. Since we get this partition by the construction of the proof of Theorem 5.10, we know by the proof of Theorem 5.12 that the relation of this partition is a congruence relation. In the following example, we build the minimal partition for the transformation τ_{mgu} as described in the proof of Theorem 5.13:

Example 5.12. Let τ_{mgu} be the tree function of Example 5.8 and (σ, r) the unified earliest partition with congruence relation of Example 5.9 (Figure 5.6). Now we apply the construction of the previous proof. Thereto, we have to consider the most-general unifiers to deduce the congruence relation $\sim_r$. We are only interested in the unifiers of an inessential and an essential residual. For every residual, we choose the first tree occurring in the table on Page 65 as representative. Let $u = \mathtt{b}^2(\mathtt{e})$. We get the following:

$$\mathsf{mgu}(r(\mathtt{g}(\mathtt{d}(u))), r(\mathtt{g}(\mathtt{d}(\mathtt{b}(\mathtt{e}))))) = \langle \mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E})), \top \rangle \tag{5.20}$$

$$\mathsf{mgu}(r(\mathtt{d}(u)), r(\mathtt{d}(\mathtt{b}(\mathtt{e})))) = \langle \mathtt{B}(\mathtt{E}), \top \rangle \tag{5.21}$$

$$\mathsf{mgu}(r(\mathtt{d}'(u)), r(\mathtt{d}(\mathtt{b}(\mathtt{e})))) = \langle \mathtt{E}, \top \rangle \tag{5.22}$$

$$\mathsf{mgu}(r(\mathtt{d}(u)), r(\mathtt{d}(\mathtt{e}))) = \langle \mathtt{E}, \top \rangle \tag{5.23}$$

$$\mathsf{mgu}(r(\mathtt{d}'(u)), r(\mathtt{d}'(\mathtt{b}(\mathtt{e})))) = \langle \mathtt{B}(\mathtt{E}), \top \rangle \tag{5.24}$$

$$\mathsf{mgu}(r(u), r(\mathtt{b}(\mathtt{e}))) = \langle \mathtt{B}(\mathtt{E}), \top \rangle \tag{5.25}$$

$$\mathsf{mgu}(r(u), r(\mathtt{e})) = \langle \mathtt{E}, \top \rangle \tag{5.26}$$

$$\mathsf{mgu}(r(\mathtt{b}^2(\mathtt{a})), r(\mathtt{b}(\mathtt{a}))) = \langle \mathtt{B}(\mathtt{E}), \top \rangle \tag{5.27}$$

$$\mathsf{mgu}(r(\mathtt{b}^2(\mathtt{a})), r(\mathtt{a})) = \langle \mathtt{E}, \top \rangle \tag{5.28}$$

Now we see in the first equation that tree $\mathtt{g}(\mathtt{d}(\mathtt{b}(\mathtt{e})))$ with inessential residual is potential similar to $\mathtt{g}(\mathtt{d}(u))$, i.e., $\mathtt{g}(\mathtt{d}(\mathtt{b}(\mathtt{e}))) \sim_r' \mathtt{g}(\mathtt{d}(u))$. Their residuals have a most-general unifier of the form $\langle v, \top \rangle$ with ground tree $v \in \mathcal{T}_\Delta$ and there is no other essential residual in the range of r, which has such a most-general unifier with $r(\mathtt{g}(\mathtt{d}(\mathtt{b}(\mathtt{e}))))$. Since y is the only context in $\mathrm{dom}(r(\mathtt{g}(\mathtt{d}(u))))$, the trees are similar:

$$\mathtt{g}(\mathtt{d}(\mathtt{b}(\mathtt{e}))) \sim_r \mathtt{g}(\mathtt{d}(u)) \ .$$

Let us consider the tree $t = \mathtt{d}(\mathtt{b}(\mathtt{e}))$. Also the residuals of $\mathtt{d}(u)$ and t have a most-general unifier of the form $\langle s, \top \rangle$ with ground tree $s \in \mathcal{T}_\Delta$ (Equation 5.21). On the other hand, also the residuals of $\mathtt{d}'(u)$ and t have such a unifier (Equation 5.22). The trees $\mathtt{d}(u)$ and $\mathtt{d}'(u)$ are not equivalent, hence, $\mathtt{d}(\mathtt{b}(\mathtt{e}))$ is similar

to none of the trees with essential residual. In particular,

$$\mathtt{d}(\mathtt{b}(\mathtt{e})) \not\sim_r \mathtt{d}(u) \ .$$

This difference causes that $\mathtt{b}(\mathtt{e})$ and u are not similar, as well as $\mathtt{e}$ and $\mathtt{b}(\mathtt{e})$. Also u and $\mathtt{e}$ are not similar. Altogether, we observe the following similarities (of representatives of the equivalence classes):

$$\begin{aligned} \mathtt{g}(\mathtt{d}(u)) &\sim_r \mathtt{g}(\mathtt{d}(\mathtt{b}(\mathtt{e}))) \\ \mathtt{d}(u) &\sim_r \mathtt{d}(\mathtt{e}) \\ \mathtt{d}'(u) &\sim_r \mathtt{d}'(\mathtt{b}(\mathtt{e})) \\ \mathtt{b}^2(\mathtt{a}) &\sim_r \mathtt{b}(\mathtt{a}) \sim_r \mathtt{a} \end{aligned}$$

Additionally, for all trees $u, u' \in \mathrm{dom}(r)$ with $u \equiv_r u'$ holds $u \sim_r u'$. We have to adapt the output under σ of these trees with inessential residuals to the essential residuals of the similar trees. The bottom line is the partition (σ', r') given in Figure 5.7. This congruence relation $\equiv_{r'}$ equals $\sim_{r'}$. It has eight equivalence classes. ◁

It follows that for every path-finite partition of finite index, an equivalent minimal and path-finite partition with finite index exists. Moreover, we will see that this partition is unique for its transformation.

Theorem 5.14. *Let τ be a tree function with path-finite partition of finite index, then the equivalent minimal and path-finite partition with finite index is unique.*

Proof. Let (σ, r) and (σ', r') be two minimal and path-finite partitions with finite index of the same tree function τ. We show that $\sigma = \sigma'$ and $r = r'$.

If there is a tree $u \in \mathrm{dom}(r)$ and context $c \in \mathrm{dom}(r(u))$, then $c \cdot u$ is in the domain of τ and with that both $u \in \mathrm{dom}(r')$ and $c \in \mathrm{dom}(r'(u))$. The partitions have the same domain and also the residuals of a tree $u \in \mathrm{dom}(r)$ under r and r' have the same domain, i.e., $\mathrm{dom}(r(u)) = \mathrm{dom}(r'(u))$. We prove that for all trees $u \in \mathrm{dom}(r)$ holds

$$[u]_r = [u]_{r'} \quad \text{and} \quad r(u) = r'(u) \ .$$

It follows immediately that $\sigma = \sigma'$. Let $u \in \mathrm{dom}(r)$. For every $c \in \mathrm{dom}(r(u))$ holds:

$$r(u)(c) \cdot \sigma(u) = \tau(c \cdot u) = r'(u)(c) \cdot \sigma'(u) \tag{5.29}$$

First assume that $r(u)$ and $r'(u)$ are inessential. It follows that $r(u) = r'(u)$. For every tree $u' \in [u]_r$ with $r'(u')$ is inessential, too, the residual under r' equals the residual of u, i.e., $r'(u') = r'(u)$. It follows $u' \in [u]_{r'}$.

Now assume that $r(u)$ is essential. We will show the following:

$$\exists\ u_1 \equiv_r u \text{ with } r(u) = r(u_1) = r'(u_1) \tag{5.30}$$

u	$\sigma'(u)$	$r'(u)$	
		y	$\mathtt{g}(y)$
$\mathtt{g}(\mathtt{d}(\mathtt{b}^m(\mathtt{e})))$	$\mathtt{G}(\mathtt{D},\mathtt{B}^m(\mathtt{E}))$	y	
$\mathtt{g}(\mathtt{d}'(\mathtt{b}^m(\mathtt{e})))$	$\mathtt{G}(\mathtt{D},\mathtt{B}^{m+1}(\mathtt{E}))$	y	
$\mathtt{g}(\mathtt{f}(\mathtt{b}^m(\mathtt{a})))$	$\mathtt{G}(\mathtt{D},\mathtt{B}(\mathtt{E}))$	y	
$\mathtt{d}(\mathtt{b}^n(\mathtt{e}))$	$\mathtt{B}^n(\mathtt{E})$	$\mathtt{A}(y,\mathtt{E})$	$\mathtt{G}(\mathtt{D},y)$
$\mathtt{d}(\mathtt{b}(\mathtt{e}))$	$*$	$\mathtt{A}(\mathtt{B}(\mathtt{E}),\mathtt{E})$	$\mathtt{G}(\mathtt{D},\mathtt{B}(\mathtt{E}))$
$\mathtt{d}(\mathtt{e})$	$\mathtt{E}$	$\mathtt{A}(y,\mathtt{E})$	$\mathtt{G}(\mathtt{D},y)$
$\mathtt{d}'(\mathtt{b}^n(\mathtt{e}))$	$\mathtt{B}^n(\mathtt{E})$	$\mathtt{A}(\mathtt{B}(y),y)$	$\mathtt{G}(\mathtt{D},\mathtt{B}(y))$
$\mathtt{d}'(\mathtt{b}(\mathtt{e}))$	$\mathtt{B}(\mathtt{E})$	$\mathtt{A}(\mathtt{B}(y),y)$	$\mathtt{G}(\mathtt{D},\mathtt{B}(y))$
$\mathtt{d}'(\mathtt{e})$	$*$	$\mathtt{A}(\mathtt{B}(\mathtt{E}),\mathtt{E})$	$\mathtt{G}(\mathtt{D},\mathtt{B}(\mathtt{E}))$
$\mathtt{f}(\mathtt{b}^m(\mathtt{a}))$	$*$	$\mathtt{A}(\mathtt{B}(\mathtt{E}),\mathtt{E})$	$\mathtt{G}(\mathtt{D},\mathtt{B}(\mathtt{E}))$

u	$\sigma'(u)$	$r'(u)$			
		$\mathtt{d}(\mathtt{b}^k(y))$	$\mathtt{d}'(\mathtt{b}^k(y))$	$\mathtt{g}(\mathtt{d}(\mathtt{b}^k(y)))$	$\mathtt{g}(\mathtt{d}'(\mathtt{b}^k(y)))$
$\mathtt{b}^n(\mathtt{e})$	$\mathtt{B}^n(\mathtt{E})$	$\mathtt{A}(\mathtt{B}^k(y),\mathtt{E})$	$\mathtt{A}(\mathtt{B}^{k+1}(y),\mathtt{B}^k(y))$	$\mathtt{G}(\mathtt{D},\mathtt{B}^k(y))$	$\mathtt{G}(\mathtt{D},\mathtt{B}^{k+1}(y))$
$\mathtt{b}(\mathtt{e})$	$*$	$\mathtt{A}(\mathtt{B}^{k+1}(\mathtt{E}),\mathtt{E})$	$\mathtt{A}(\mathtt{B}^{k+2}(\mathtt{E}),\mathtt{B}^{k+1}(\mathtt{E}))$	$\mathtt{G}(\mathtt{D},\mathtt{B}^{k+1}(\mathtt{E}))$	$\mathtt{G}(\mathtt{D},\mathtt{B}^{k+2}(\mathtt{E}))$
$\mathtt{e}$	$*$	$\mathtt{A}(\mathtt{B}^k(\mathtt{E}),\mathtt{E})$	$\mathtt{A}(\mathtt{B}^{k+1}(\mathtt{E}),\mathtt{B}^k(\mathtt{E}))$	$\mathtt{G}(\mathtt{D},\mathtt{B}^k(\mathtt{E}))$	$\mathtt{G}(\mathtt{D},\mathtt{B}^{k+1}(\mathtt{E}))$

u	$\sigma'(u)$	$r'(u)$		
		$\mathtt{b}^k(y)$	$\mathtt{f}(\mathtt{b}^k(y))$	$\mathtt{g}(\mathtt{f}(\mathtt{b}^k(y)))$
$\mathtt{b}^m(\mathtt{a})$	$\mathtt{B}^m(\mathtt{E})$	$\mathtt{B}^k(y)$	$\mathtt{A}(\mathtt{B}(\mathtt{E}),\mathtt{E})$	$\mathtt{G}(\mathtt{D},\mathtt{B}(\mathtt{E}))$

Figure 5.7: The minimal partition (σ', r') of Example 5.12. ($m \in \mathbb{N}_0$, $n \geq 2$)

There are infinitely many different trees in $[u]_r$ with different outputs under σ, i.e., $|\sigma([u]_r)| = \infty$. Since the index of $\equiv_{r'}$ is finite, there is an infinite set of such trees, which are r'-equivalent, i.e., there is a tree $u' \equiv_r u$ and the set $S = [u]_r \cap [u']_{r'}$ is infinite and additionally $|\sigma(S)| = \infty$. Let $u_1, u_2 \in S$ with $\sigma(u_1) \neq \sigma(u_2)$. Note that the two trees u_1 and u_2 are equivalent with respect to both r and r'. Assume the residual of u_1 under r' is inessential. There is a context $c \in \mathrm{dom}(r(u_1))$ with $r(u_1)(c) \in \hat{\mathcal{T}}_\Delta(y)$. We get

$$r(u_1)(c) \cdot \sigma(u_1) = r'(u_1)(c) = r'(u_2)(c) = r(u_1)(c) \cdot \sigma(u_2)$$

which is a contradiction to $\sigma(u_1) \neq \sigma(u_2)$. Thus, the residual $r'(u_1)$ is essential. With Proposition 2.1 and Equation 5.29 we will show that $r(u_1) = r'(u_1)$: For every context $c \in \mathrm{dom}(r(u_1))$ holds

$$\begin{aligned} r(u_1)(c) \cdot \sigma(u_1) &= r'(u_1)(c) \cdot \sigma'(u_1) \\ r(u_2)(c) \cdot \sigma(u_2) &= r'(u_2)(c) \cdot \sigma'(u_2) \end{aligned}$$

For every context c, either $r(u_1)(c) = r'(u_1)(c) \in \mathcal{T}_\Delta$ is ground or there is a tree $z_c \in \hat{\mathcal{T}}_\Delta(y)$ with $r(u_1)(c) \cdot z_c = r'(u_1)(c)$ or $r(u_1)(c) = r'(u_1)(c) \cdot z_c$ (Proposition 2.1). Assume there are c and c' with different trees $z_c \neq z_{c'}$ and $r(u_1)(c) \cdot z_c = r'(u_1)(c)$ and $r(u_1)(c') \cdot z_{c'} = r'(u_1)(c')$, respectively. It follows that $\sigma(u_i) = z_c \cdot \sigma'(u_i) = z_{c'} \cdot \sigma'(u_i)$ (for both $i \in [2]$). It follows, again with Proposition 2.1 that $z_c = z_{c'}$. Otherwise, assume there are c and c' with different trees z_c and $z_{c'}$ and

$$\begin{aligned} r(u_1)(c) \cdot z_c &= r'(u_1)(c) \\ r(u_1)(c') &= r'(u_1)(c') \cdot z_{c'} \end{aligned}$$

Then $\sigma(u_i) = z_c \cdot \sigma'(u_i)$ and $z_{c'} \cdot \sigma(u_i) = \sigma'(u_i)$ for both $i \in [2]$. It follows that $z_c = z_{c'} = y$. Summarizing, there is a tree $z \in \hat{\mathcal{T}}_\Delta(y)$ with $r(u_1)(c) \cdot z = r'(u_1)(c)$ for every context c or $r(u_1)(c) = r'(u_1)(c) \cdot z$ for every context c. Since r and r' are earliest, z must equal y. Consequently, $r(u_1) = r'(u_1)$. This gives us Equation 5.30.

Now assume $r(u)$ is essential and $r'(u)$ is inessential. There is a tree $u_1 \equiv_r u$ with essential residual $r(u) = r(u_1) = r'(u_1)$ (cf. Equation 5.30). For all context $c \in \mathrm{dom}(r(u))$, we get

$$r'(u_1)(c) \cdot \sigma(u) = r(u)(c) \cdot \sigma(u) = r'(u)(c) \; .$$

Thus, these residuals have the following most-general unifier:

$$\mathsf{mgu}(r'(u_1), r'(u)) = \langle \sigma(u), \top \rangle$$

Since (σ', r') is minimal, the trees are not similar, i.e., $u_1 \not\sim_{r'} u$. There is some context $c \in \mathcal{C}_\Sigma(y)$ (possibly y) such that $c \cdot u_1 \not\sim'_{r'} c \cdot u$.

Let us consider the possible most-general unifiers for such contexts c: Since the relation $\equiv_{r'}$ is a congruence, we know (by Lemma 5.11) that for every

context c with $c \cdot u_1 \in \mathrm{dom}(r')$ the dedicated image $\mathsf{di}(c)(u_1) \in \mathcal{T}_\Delta(y)$ exist. For every context $c' \in \mathrm{dom}(r'(c \cdot u_1))$ holds

$$r'(c \cdot u_1)(c') \cdot \mathsf{di}(c)(u_1) = r'(u_1)(c' \cdot c) \; .$$

Furthermore, it follows for every such context c and every $c' \in \mathrm{dom}(r'(c \cdot u_1))$:

$$\begin{aligned} r'(c \cdot u_1)(c') \cdot \mathsf{di}(c)(u_1) \cdot \sigma(u) &= r'(u_1)(c' \cdot c) \cdot \sigma(u) \\ &= r(u)(c' \cdot c) \cdot \sigma(u) \\ &= r'(u)(c' \cdot c) && (r'(u) \text{ is inessential}) \\ &= r'(c \cdot u)(c') \cdot \sigma'(c \cdot u) \end{aligned}$$

Thus, we get for the most-general unifier (w.r.t. the ordering $\geq$ on Page 20):

$$\mathsf{mgu}(r'(c \cdot u_1), r'(c \cdot u)) \geq \langle \mathsf{di}(c)(u_1) \cdot \sigma(u), \sigma'(c \cdot u) \rangle$$

However, (σ', r') is unified earliest and $\sigma'(c \cdot u)$ is realizable in $r'(c \cdot u)$. Hence, the most-general unifier is in $\{\langle \mathsf{di}(c)(u_1) \cdot \sigma(u), \top \rangle, \langle \top, \top \rangle, \langle y, y \rangle\}$.

If $u_1 \not\sim_{r'} u$, then there is some context $c \in \mathcal{C}_\Sigma(y)$ such that $c \cdot u_1 \not\sim'_{r'} c \cdot u$. More precisely, we get that

$$\mathsf{mgu}(r'(c \cdot u_1), r'(c \cdot u)) = \langle \mathsf{di}(c)(u_1) \cdot \sigma(u), \top \rangle \; .$$

Otherwise, the trees would be r'-equivalent. Thus, $r'(c \cdot u)$ is inessential. Additionally, for every tree u_2 (including $c \cdot u_1$) with $\mathsf{mgu}(r'(u_2), r'(c \cdot u)) = \langle v_2, \top \rangle$ for some ground term $v_2 \in \mathcal{T}_\Delta$, there is a tree u'_2 with $r'(u_2) = r'(u'_2) = r(u'_2)$ (Property 5.30). Thus,

$$\mathsf{mgu}(r(u'_2), r'(c \cdot u)) = \langle v_2, \top \rangle \; . \tag{5.31}$$

On the other hand, we get for every context $c' \in \mathrm{dom}(r(c \cdot u))$:

$$\begin{aligned} r(c \cdot u)(c') \cdot \sigma(c \cdot u) &= r(u)(c' \cdot c) \cdot \sigma(u) && (\text{consistent}) \\ &= r'(u)(c' \cdot c) \cdot \sigma'(u) && (\text{equivalence}) \\ &= r'(c \cdot u)(c') \cdot \sigma'(c \cdot u) && (\text{consistent}) \\ &= r'(c \cdot u)(c') && (\text{inessential}) \end{aligned}$$

It follows that $\mathsf{mgu}(r(c \cdot u), r'(c \cdot u)) \in \{\langle \top, \top \rangle, \langle \sigma(c \cdot u), \top \rangle\}$. First, assume that $r(c \cdot u)$ is inessential and equals $r'(c \cdot u)$. Since $u_1 \equiv_r u$ and $\equiv_r$ is a congruence, we get that also $r(c \cdot u_1) = r'(c \cdot u)$. With Equation 5.29 it follows:

$$\mathsf{mgu}(r'(c \cdot u_1), r'(c \cdot u)) = \mathsf{mgu}(r'(c \cdot u_1), r(c \cdot u_1)) = \langle \sigma'(c \cdot u_1), \bot \rangle$$

This contradicts the fact that (σ', r') is unified earliest because $\sigma'(c \cdot u_1)$ is realizable in $r'(c \cdot u_1)$. Thus, we get $\mathsf{mgu}(r(c \cdot u), r'(c \cdot u)) = \langle \sigma(c \cdot u), \top \rangle$ and with Equation 5.31

$$\mathsf{mgu}(r(c \cdot u), r(u'_2)) \geq \langle \sigma(c \cdot u), v_2 \rangle \; .$$

Since $\sigma(c \cdot u)$ is realizable in $r(c \cdot u)$ and (σ, r) is unified earliest, the most-general unifier must equal $\langle y, y\rangle$ or $\langle \top, \top\rangle$, i.e., the trees are r-equivalent. We get $r(c \cdot u) = r(u_2') = r'(u_2') = r'(u_2)$. In particular, also $r(c \cdot u) = r'(c \cdot u_1)$. Summarizing, for every tree u_2 with $\mathsf{mgu}(r'(u_2), r'(c \cdot u)) = \langle v_2, \top\rangle$ for some ground $v_2 \in \mathcal{T}_\Delta$, we get $u_2 \equiv_{r'} c \cdot u_1$. This is a contradiction to $c \cdot u_1 \not\sim'_{r'} c \cdot u$. Thus, $u_1 \sim_{r'} u$ and since (σ', r') is minimal, $u_1 \equiv_{r'} u$. However, we assumed $r'(u)$ is inessential, whereas $r'(u_1)$ is essential. — It follows that there are no trees u with $r(u)$ essential and $r'(u)$ inessential.

Finally, assume that $r(u)$ and $r'(u)$ are both essential, but different. Then, we know that for all $c \in \mathrm{dom}(r(u))$ holds $r(u)(c) \cdot \sigma(u) = r'(u)(c) \cdot \sigma'(u)$. Since the residuals are different and essential, the most-general unifier is

$$\mathsf{mgu}(r(u), r'(u)) = \langle \sigma(u), \sigma'(u)\rangle \ .$$

With Property 5.30 there is a tree $u' \equiv_{r'} u$ with $r(u') = r'(u)$. Thus,

$$\mathsf{mgu}(r(u), r(u')) = \langle \sigma(u), \sigma'(u)\rangle \ .$$

However, $\sigma(u)$ is realizable in $r(u)$. — Contradiction to (σ, r) is unified earliest.

Altogether, the two residual functions r and r' are the same. Thus, the minimal partition is unique. □

Summarizing the results from the previous sections, we obtain from the Theorems 5.7, 5.8, 5.9, 5.10, 5.12, 5.13, and 5.14:

Theorem 5.15. *For every tree function τ with path-finite partition of finite index, a minimal, path-finite partition of finite index exists, which is unique.* □

We call this unique partition *bottom-up partition* if it exists:

Definition 5.10 (Bottom-up Partition)**.** A partition (σ, r) of a function τ is called the *bottom-up partition* of τ if it is minimal, path-finite, and has finite index.

We write (σ_τ, r_τ) for the bottom-up partition of τ. In the following we will see that there exists a bottom-up tree transducer for a tree function if and only if this unique minimal partition (which is path-finite and has finite index) exists for τ.

Chapter 6

Myhill-Nerode Theorem

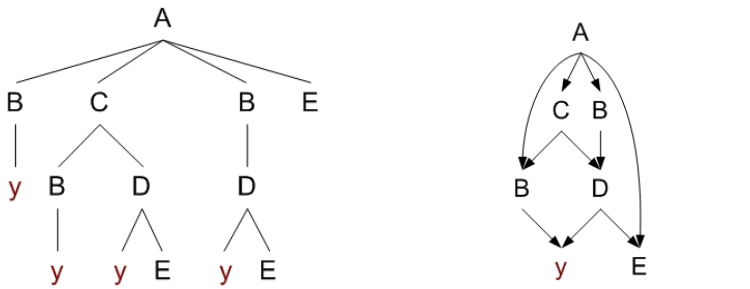
In Section 5.1, we have seen a correspondence between partitions and DBTTs: Every deterministic bottom-up tree transducer defines a partition, which is path-finite and has finite index. Not only that the properties path-finiteness and finite index are necessary to define the transformation of a DBTT, but also they are sound. More generally, we get the following Myhill-Nerode style theorem. It says that for every tree function, which is definable by a DBTT, the bottom-up partition exists. Furthermore, the DBTT described by the bottom-up partition is the unique minimal bottom-up transducer of this function.

Theorem 6.1. *The following three properties are equivalent for a partial tree function* $\tau : \mathcal{T}_\Sigma \dashrightarrow \mathcal{T}_\Delta$*:*

1. *There is a DBTT* T *with* $\tau^T = \tau$*.*
2. *There is a unique minimal DBTT* T_{min} *with* $\tau^{T_{min}} = \tau$*.*
3. *The bottom-up partition* (σ_τ, r_τ) *exists.*

We will prove this theorem in this chapter. In the first section, we show that a unique minimal transducer can be constructed for every deterministic bottom-up tree transducer. The idea of the construction is inspired by the properties of the bottom-up normal form for a partition. We get a unified earliest normal form and prove that there is a unique minimal transducer. For a given "proper" deterministic bottom-up tree transducer, i.e., one where every state produces either none or infinitely many outputs, the corresponding unique minimal transducer can be constructed in polynomial time.

In a second step, we will show that the partition $(\mathsf{out}^{T_{min}}, r^{T_{min}})$ given by the minimal DBTT T_{min} (cf. Section 5.1) is the bottom-up partition of its transformation $\tau^{T_{min}}$. Finally, we show that (3.) implies (1.): Given a bottom-up partition, we construct a DBTT, which describes the same tree function. Additionally, we will show that this transducer is already the minimal DBTT.



(a) An expanded tree and ... (b) its compacted representation.

Figure 6.1: The compacted representation of a tree.

6.1 Minimization of Bottom-Up Transducers

In this section, we prove the first part of the Theorem 6.1, i.e., we show that for every deterministic bottom-up tree transducer, there is a unique equivalent DBTT in normal form. Even though the idea is similar to [EMS09], obtaining the normal form is quite different for DBTTs than for top-down transducers. Generating the "earliest" normal form for a given trim DBTT proceeds in four steps. The properties of a DBTT induced by these steps are similar to some of the properties of a bottom-up partition.

1. First, we make the transducer *proper*, i.e., we remove all output from states, which only produce finitely many different outputs. The output for such states is postponed until the root node of the input tree. This is similar to the "proper normal form" of [AU71, EM03c] (which removes states that produce finitely many outputs, using regular look-ahead).

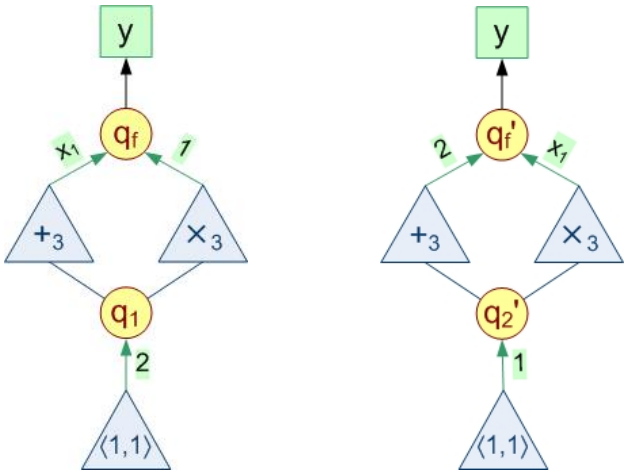
2. We make the transducer *earliest*, i.e., every state produces output as early as possible during translation.

3. We remove pairwise *ground context unifiers* (this is a technical property, achievable in quadratic time on the transducer).

4. Last, we *minimize* in the usual way (by merging isomorphic states).

Steps (2)–(4) can be done in polynomial time, i.e., given a proper transducer, its unique unified earliest transducer is constructed in polynomial time. As a positive side effect, equivalence checking for proper transducers can be done in polynomial time. Constructing a proper transducer (Step 1) takes double-exponential time in the worst case.

Also in Steps 2 and 3, the constructions could result in an exponential blow-up if we consider trees as compositions of irreducible terms. To avoid that, we assume that terms are given in a *compacted representation*. Thereto, isomorphic subterms are shared as depicted in Figure 6.1. Considering trees in this compacted representation, the composition of two trees is only a substitution in one



(a) The detail of T_+. (b) The detail of $T_\times$.

Figure 6.2: Details depending on input label $\langle 1, 1\rangle$ of the DBTTs of Example 6.1.

occurrence of the variable y. Thus, we have no multiple copying of subterms and the size of the new terms in Steps 2 and 3 is restricted.

6.1.1 Proper Transducers

For a given trim transducer, there is not necessarily a unique minimal equivalent transducer. Recall the Example 5.4 of the quotient ring of integers modulo 3. The two intuitive partitions are not proper. For the transformation of that example, we can also define different transducers, which are equivalent and have the same size.

Example 6.1. Assume that Σ and Δ are defined as in Example 5.4. Consider the transducers $T_+ = (Q_+, \Sigma, \Delta, R_+, F_+)$ and $T_\times = (Q_\times, \Sigma, \Delta, R_\times, F_\times)$ defined below (Figure 6.2). Both describe the tree function τ of that example in Section 5.2.2. The sets of states are $Q_+ = \{q_f, q_0, q_1, q_2\}$ and $Q_\times = \{q'_f, q'_0, q'_1, q'_2\}$. The final functions are only defined for q_f and q'_f, respectively, by $F_+(q_f) = y$ and $F_\times(q'_f) = y$. Finally, the set of rules R_+ for all $a, b \in \{0, 1, 2\}$ is given by

$$\begin{aligned} \langle a, b\rangle &\mapsto q_i(j) \qquad \text{for } i = a \times_3 b, \quad j = a +_3 b \\ +_3(q_a) &\mapsto q_f(x_1) \\ \times_3(q_a) &\mapsto q_f(a)\ , \end{aligned}$$

whereas the set of rules $R_\times$ for all $a, b \in \{0, 1, 2\}$ is given by

$$\begin{aligned} \langle a, b\rangle &\mapsto q_i(j) \qquad \text{for } i = a +_3 b, \quad j = a \times_3 b \\ \times_3(q_a) &\mapsto q_f(x_1) \\ +_3(q_a) &\mapsto q_f(a)\ . \end{aligned}$$

The transducer T_+ produces the output $\tau(+_3(p))$ for every pair p at the node labeled by p. It stores the output of $\times_3(p)$ in the state reached by p. At the root, it suffices either to pass the produced output (if the label of the root is $+_3$) or produces the correct output depending on the state (with root label $\times_3$). On the other side, $T_\times$ produces the output of $\times_3(p)$ at the leaf and stores the output of $+_3(p)$ in the state. Accordingly, the transitions for the root nodes are swapped, too. In Figure 6.2, the difference of these transducers is illustrated. Both parts of the figure show the transitions, which are necessary to translate a tree containing a leaf with label $\langle 1, 1\rangle$ for the deterministic bottom-up tree transducers T_+ and $T_\times$, respectively.

Both transducers are trim and describe the same transformation τ. It is not clear how a unique normal form for the transformation with less than four states could look like. ◁

Considering the partitions of these DBTTs (cf. Section 5.1), we get exactly the partitions of Example 5.4, i.e., (σ_+, r_+) and $(\sigma_\times, r_\times)$, respectively. Likewise, we introduce the property "proper" for DBTTs.

Let T denote a trim transducer with set of states Q. A state $q \in Q$ is called *essential* if the set of results $\{[\![t]\!]^T \mid t \in \mathcal{L}^T(q)\}$ for input trees reaching q is infinite. Otherwise, q is called *inessential*. Note that all states of the transducers in Example 6.1 are inessential. A proper transducer postpones outputs at inessential states.

Definition 6.1 (Proper)**.** The trim transducer T is called *proper* if every inessential state does not produce any output, i.e., is in Q_*.

For every trim bottom-up tree transducer exists an equivalent proper transducer:

Proposition 6.2. *[AU71, EM03c] For every trim DBTT T, a DBTT T' can be constructed with the following properties:*

1. *T' is equivalent to T*
2. *T' is proper*
3. *$|T'| \leq \Gamma \cdot |T|$*

where Γ is the sum of sizes of all outputs produced for inessential states of T.

In the worst case, an inessential state may have exponentially many outputs – even if the input alphabet has maximal rank 1. In case that both input and output alphabets have symbols of ranks greater than 1, doubly exponentially many outputs of inessential states are possible.

Proof of Proposition 6.2. Let $T = (Q, \Sigma, \Delta, R, F)$ be a trim DBTT. First, we determine the set $\mathsf{Finite} \subseteq Q$ of inessential states of T. For that, we construct the *dependence* graph $G_T = (Q, E_T)$ of T. The set of nodes of G_T is given by the set Q of states of T, and the set E_T of edges is given by:

$$E_T = \{(q_i, q) \mid \mathsf{a}(q_1, \ldots, q_k) \to q(z) \in R \text{ and } x_i \text{ occurs in } z\}$$

An edge $(q_i, q) \in E_T$ is called essential if there is a transition $\mathsf{a}(q_1, \ldots, q_k) \to q(z)$ in R where z contains x_i, but is different from x_i. A strongly connected component of G_T is called essential if it contains an essential edge. Otherwise, it is called inessential. Then a state $q \in Q$ is essential iff it is reachable from some essential strongly connected component. Since the set of all strongly connected components can be computed in linear time [Tar72], also the set of states within all essential strongly connected components can be computed in linear time. Therefore, both the set of all essential states, as well as the set Finite of inessential states, can be computed in linear time.

Consider an inessential state $q \in \mathsf{Finite}$. We partition q into new states $\langle q, z\rangle$ where $q(z)$ is a possible result for some input tree $t \in \mathcal{L}^T(q)$. If q occurs on a left-hand side of a rule as the state for the i-th argument where the state in the right-hand side is essential, a new rule is generated where q is replaced with $\langle q, z\rangle$ and the corresponding variable x_i is replaced with z. Also, the final function F' should be modified accordingly for inessential states.

Formally, let Finite denote the set of all inessential states of T. Let the set of new inessential states $Q_f \subseteq \mathsf{Finite} \times \mathcal{T}_\Delta$ be defined as the least set satisfying the inequation:

$$Q_f \supseteq \{\langle q, z[v_1/x_{j_1}, \ldots, v_r/x_{j_r}]\rangle \mid \mathsf{a}(q_1, \ldots, q_m) \to q(z) \in R, q \in \mathsf{Finite}, \\ x_{j_1}, \ldots, x_{j_r} \text{ occur in } z,\ \langle q_{j_1}, v_1\rangle, \ldots, \langle q_{j_r}, v_r\rangle \in Q_f\}$$

Then we construct a transducer $T' = (Q', \Sigma, \Delta, R', F')$ by

- $Q' = (Q \backslash \mathsf{Finite}) \cup Q_f$;
- Assume that $F(q) = z$. If $q \notin \mathsf{Finite}$, then $F'(q) = z$. If $\langle q, v\rangle \in Q_f$, then $F'(\langle q, v\rangle) = z \cdot v$.
- Assume that $\mathsf{a}(q_1, \ldots, q_m) \to q(z) \in R$. Then T' has transitions of the form $\mathsf{a}(q'_1, \ldots, q'_m) \to q'(z')$ with the following properties:
  - If $q_i \notin \mathsf{Finite}$, then $q'_i = q_i$. If $q_i \in \mathsf{Finite}$, then $q'_i = \langle q_i, v_i\rangle \in Q_f$. Let z_i be equal to x_i if $q_i \notin \mathsf{Finite}$ and $z_i = v_i$ otherwise.
  - If $q \notin \mathsf{Finite}$, then $q' = q$ and $z' = z[z_1, \ldots, z_m]$. If $q \in \mathsf{Finite}$, then $q' = \langle q, z[z_1, \ldots, z_m]\rangle$ and $z' = *$.

This deterministic bottom-up tree transducer T' is trim and proper. For the size, we observe that a transition of T is duplicated for every sequence of possible outputs of the inessential states on the left-hand side. Furthermore, x_i-nodes in right-hand sides are replaced by outputs of inessential states. Thus, the size of T' is bounded by $\Gamma \cdot |T|$, where Γ is the sum of sizes of all outputs produced for inessential states of T. □

With this theorem we get a proper deterministic bottom-up tree transducer for the transformation of Example 5.4. Thereto, we apply the construction of the proof to one of the two transducers T_+ and $T_\times$ of Example 6.1.

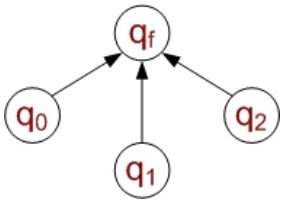


Figure 6.3: The dependency graph G_{T_+} for DBTT T_+.

Example 6.2. Consider again the transducer T_+ of Example 6.1. The dependence graph is

$$G_{T_+} = (\{q_f, q_0, q_1, q_2\}, \{(q_0, q_f), (q_1, q_f), (q_2, q_f)\})$$

with only inessential edges (Figure 6.3). We determine that all states are inessential. The equivalent proper DBTT $T'_+ = (Q'_+, \Sigma, \Delta, R'_+, F'_+)$ has the following set of states:

$$Q'_+ = \{\langle q_f, i\rangle, \langle q_0, i\rangle, \langle q_1, 1\rangle, \langle q_1, 2\rangle, \langle q_2, 0\rangle \mid 0 \le i \le 2\} .$$

Since every state of Q'_+ is inessential, the output is postponed to the final function, whereas the right-hand sides of the transitions R'_+ are of the form $\langle q, z\rangle(*)$. More precisely, for all pairs $\langle a, b\rangle \in \Sigma^{(0)}$, the transitions have the form

$$\langle a, b\rangle \to \langle q_i, j\rangle(*)$$

where $i = a \times_3 b$ and $j = a +_3 b$. Additionally, we have for states $\langle q_a, b\rangle \in Q'_+$ with $a \neq f$ the transitions:

$$+_3(\langle q_a, b\rangle) \to \langle q_f, b\rangle(*)$$
$$\times_3(\langle q_a, b\rangle) \to \langle q_f, a\rangle(*)$$

The final function is given by

$$F'_+(\langle q_f, a\rangle) = a \quad \text{for all } a \in \{0, 1, 2\} .$$

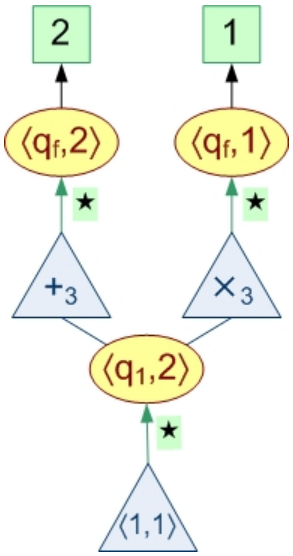


Figure 6.4: The details depending on label $\langle 1, 1\rangle$ of the DBTT of Example 6.2.

Figure 6.4 represents the transitions of DBTT T'_+, which are potential applied to nodes of trees with leaf labeled $\langle 1, 1\rangle$. It is the same part as we have seen for the two transducers T_+ and $T_\times$ in Figure 6.2. If we construct $T'_\times$ for the second transducer $T_\times$ of Example 6.1, we get an isomorphic transducer to T'_+. Both deterministic bottom-up tree transducers $T'_\times$ and T'_+ are proper and realize the transformation τ. ◁

6.1.2 Earliest Transducers

Assume that we are given a proper DBTT T. Analog as for partitions, we now want this transducer to produce the output at essential states as *early* as possible. Thereto, we compute the *greatest common suffix* of all non-ground images of contexts for a state q and produce it at q directly.

For an essential state q, let $\hat{\mathcal{Z}}(q)$ denote the set of images $z \in \hat{\mathcal{T}}_\Delta(y)$ produced for contexts of q, i.e, if q is reachable by a tree u then $\hat{\mathcal{Z}}(q) = \hat{\mathcal{Z}}(u)$ w.r.t. the partition of the DBTT (cf. Page 55 for $\hat{\mathcal{Z}}(u)$). Thus, every tree in $\hat{\mathcal{Z}}(q)$ contains an occurrence of the variable y. The *greatest common suffix* of all trees in $\hat{\mathcal{Z}}(q)$ is denoted by $\mathsf{gcs}(q)$, i.e.,

$$\mathsf{gcs}(q) = \bigsqcup \hat{\mathcal{Z}}(q)$$

with respect to the order $\sqsubseteq$ on $\hat{\mathcal{T}}_\Delta(y)_\bot$ from Page 19. It equals the greatest common suffix $\mathsf{gcs}(u)$ for all $u \in \mathcal{L}^T(q)$. In contrast to partitions, for every proper DBTT, the greatest common suffices of the states can effectively be computed:

Lemma 6.3. *For a proper DBTT T, the trees $\mathsf{gcs}(q)$ for all essential states q of T can be computed in polynomial time.*

Proof. Assume that $z \in \mathcal{T}_\Delta(\mathcal{X}_k)$ and x_i occurs in z. Then $\mathsf{suff}_i(z)$ denotes the largest subtree z_i of $z[y/x_i]$ with the following properties:

- y is the only variable occurring in z_i, i.e., $z_i \in \hat{\mathcal{T}}_\Delta(y)$;
- $z[y/x_i] = z' \cdot z_i$ for some z', i.e., $z' \in \hat{\mathcal{T}}_{\Delta \cup \mathcal{X}_k \setminus \{x_i\}}(y)$.

Then the trees $\mathsf{gcs}(q)$ are the least solution of the inequations

$$\begin{array}{lcll} \mathsf{gcs}(q_i) & \sqsupseteq & \mathsf{suff}_i(\mathsf{gcs}(q) \cdot z), & \mathsf{a}(q_1, \ldots, q_k) \to q(z) \in R \text{ and } x_i \text{ occurs in } z, \\ \mathsf{gcs}(q) & \sqsupseteq & z, & F(q) = z \text{ and } y \text{ occurs in } z. \end{array}$$

Since T is proper, this system contains inequations only for essential states q. The system has a unique least solution because the right-hand sides are monotonic. In addition, since the complete lattice $\hat{\mathcal{T}}_\Delta(y)_\bot$ satisfies the ascending chain condition, this least solution can effectively be computed. Using a standard worklist algorithm, it can be shown that each inequation is evaluated at most $\mathcal{O}(|T|)$ times. If we represent elements from $\hat{\mathcal{T}}_\Delta(y)$ as compositions of irreducible trees in the compact representation, then each right-hand side also can be evaluated in polynomial time. This proves the complexity bound stated in the proposition. □

The greatest common suffix is different from $\bot$ only for essential states. Since for these states the set $\hat{\mathcal{Z}}(q)$ is not empty, the least upper bound is always in $\hat{\mathcal{T}}_\Delta(y)$.

Definition 6.2. A proper bottom-up tree transducer T is called *earliest* if the greatest common suffix of every essential state q equals y.

With Lemma 6.3 we know that the $\mathsf{gcs}(q)$ are computable in polynomial time. Thus, we can construct an earliest DBTT in polynomial time by moving these common suffices to the output of the trees reaching the state.

Theorem 6.4. *For each proper tree transducer T, a tree transducer T' can be constructed in polynomial time with the following properties:*

- *T is equivalent to T'*
- *T' is earliest.*

Proof. Let $T = (Q, \Sigma, \Delta, R, F)$ be a proper tree transducer. According to Lemma 6.3, we can compute the greatest common suffix $\mathsf{gcs}(q)$ for every essential state q of T. The corresponding earliest transducer $T' = (Q, \Sigma, \Delta, R', F')$ has the same set of states as T as well as the same input and output alphabets, but only differs in the transition function and the final function.

Let us first construct the final function F' of the new transducer T'. Let q be a state of T'. Then $F'(q)$ is defined iff $F(q)$ is defined. If q is an inessential state, then $F'(q) = F(q)$. Now assume that q is essential and $F(q) = z$. Then the greatest common suffix $\mathsf{gcs}(q)$ of q is a suffix of z, i.e., $z = u \cdot \mathsf{gcs}(q)$ for some $u \in \hat{\mathcal{T}}_\Delta(y)$. Since we assume that $\mathsf{gcs}(q)$ has already been output, we set $F'(q) = u$.

We now construct the transition function R'. Then $R'(\mathsf{a}, q_1 \dots q_m)$ is defined iff $R(\mathsf{a}, q_1 \dots q_m)$ is defined. Assume that $\mathsf{a}(q_1, \dots, q_m) \to q(z)$ is a transition in R. If q is inessential, then $z = *$ and R' contains the transition $\mathsf{a}(q_1, \dots, q_m) \to q(*)$ as well. Now assume q is essential. Then we construct the output of the corresponding transition in R' in two steps. First, we add the greatest common suffix corresponding to q to z, i.e., we define $\bar{z} = \mathsf{gcs}(q) \cdot z$. Then we remove from $\bar{z}$ the greatest common suffices of all variables occurring in (z and thus also in) $\bar{z}$. Let $x_{i_1}, \dots, x_{i_r}$ be an enumeration of the variables occurring in $\bar{z}$. Then $\bar{z}$ can be uniquely decomposed into:

$$\bar{z} = u[\mathsf{gcs}(q_{i_1})/x_{i_1}, \dots, \mathsf{gcs}(q_{i_r})/x_{i_r}]$$

where $u \in \mathcal{T}_\Delta(\{x_{i_1}, \dots, x_{i_r}\})$. Then R' has the transition $\mathsf{a}(q_1, \dots, q_m) \to q(u)$.

Due to the one-to-one correspondence of the final functions and transition functions, T' is trim. In order to prove the equivalence of T and T', it suffices to verify by induction on the structure of an input tree $t \in \mathcal{T}_\Sigma$ and every essential state q,

$$[\![t]\!]^T = q(z) \qquad \text{iff} \qquad [\![t]\!]^{T'} = q(\mathsf{gcs}(q) \cdot z) \ .$$

Moreover, for every inessential state of T, we have:

$$[\![t]\!]^T = q(*) \quad \text{iff} \qquad [\![t]\!]^{T'} = q(*)$$

In particular, this invariant implies that T' is still proper. In order to prove that T' is earliest, it suffices to verify for every context $c \in \hat{\mathcal{T}}_\Sigma(y)$ of an essential state q that the following holds

$$\tau_q^T(c) = \tau_q^{T'}(c) \cdot \mathsf{gcs}(q) \ .$$

This invariant can again be proven by induction on the length of the context c (using the first invariant).

Given the trees $\mathsf{gcs}(q)$, the construction can be performed in polynomial time. We need to care, however, not to expand the representation of suffices $\mathsf{gcs}(q)$: Such an expansion could result in an exponential blow-up of the sizes of resulting trees. The factorization of output trees, which is necessary for constructing the tree u in transitions of R' reaching essential states, however, can also be performed with the compacted representation directly. □

Example 6.3. Assume that $\Sigma = \{\mathtt{a}^{(2)}, \mathtt{b}^{(1)}, \mathtt{c}^{(0)}, \mathtt{d}^{(0)}\}$ and $\Delta = \{\mathtt{D}^{(2)}, \mathtt{E}^{(0)}\}$. Consider the proper DBTT $T = (Q, \Sigma, \Delta, R, F)$, with set of (essential) states $Q = \{q_1, q_2\}$ where the final function is $F = \{q_1 \mapsto \mathtt{D}(\mathtt{D}(y, \mathtt{E}), \mathtt{D}(y, \mathtt{E}))\}$ and the transition function R is given by:

$$\begin{array}{rlrl} \mathtt{a}(q_1, q_2) & \to q_1(\mathtt{D}(x_2, \mathtt{D}(x_1, \mathtt{E}))) & \mathtt{d} & \to q_1(\mathtt{E}) \\ \mathtt{b}(q_2) & \to q_2(\mathtt{D}(x_1, \mathtt{D}(\mathtt{D}(\mathtt{E}, \mathtt{E}), \mathtt{E}))) & \mathtt{c} & \to q_2(\mathtt{E}) \end{array}$$

To compute the greatest common suffices, we consider the following inequations:

$$\begin{array}{llll} \mathsf{gcs}(q_1) & \sqsupseteq & \mathsf{suff}_1(\mathsf{gcs}(q_1) \cdot \mathtt{D}(x_2, \mathtt{D}(x_1, \mathtt{E}))) & = \mathtt{D}(y, \mathtt{E}) \\ \mathsf{gcs}(q_2) & \sqsupseteq & \mathsf{suff}_2(\mathsf{gcs}(q_1) \cdot \mathtt{D}(x_2, \mathtt{D}(x_1, \mathtt{E}))) & = y \\ \mathsf{gcs}(q_2) & \sqsupseteq & \mathsf{suff}_1(\mathsf{gcs}(q_2) \cdot \mathtt{D}(x_1, \mathtt{D}(\mathtt{D}(\mathtt{E}, \mathtt{E}), \mathtt{E}))) & = \mathsf{gcs}(q_2) \cdot \mathtt{D}(y, \mathtt{D}(\mathtt{D}(\mathtt{E}, \mathtt{E}), \mathtt{E})) \\ \mathsf{gcs}(q_1) & \sqsupseteq & F(q_1) & = \mathtt{D}(\mathtt{D}(y, \mathtt{E}), \mathtt{D}(y, \mathtt{E})) \end{array}$$

For q_2, we obtain $\mathsf{gcs}(q_2) = y$. Moreover, since

$$\mathtt{D}(y, \mathtt{E}) \sqcup \mathtt{D}(\mathtt{D}(y, \mathtt{E}), \mathtt{D}(y, \mathtt{E})) = \mathtt{D}(y, \mathtt{E}) \ ,$$

we have $\mathsf{gcs}(q_1) = \mathtt{D}(y, \mathtt{E})$. The final function of the earliest DBTT for T' thus is given by $F' = \{q_1 \mapsto \mathtt{D}(y, y)\}$. In order to construct the new transition function, first consider the right-hand side for $\mathtt{a}(q_1, q_2)$ in R' where the right-hand side in T is $R(\mathtt{a}, q_1 q_2)) = q_1(\mathtt{D}(x_2, \mathtt{D}(x_1, \mathtt{E})))$ (Figure 6.5). We have already computed the greatest common suffix for the state q_1 of the right-hand side (Step 1 in Figure 6.5). In the second step, we construct

$$\bar{z} = \mathsf{gcs}(q_1) \cdot \mathtt{D}(x_2, \mathtt{D}(x_1, \mathtt{E})) = \mathtt{D}(y, \mathtt{E}) \cdot \mathtt{D}(x_2, \mathtt{D}(x_1, \mathtt{E})) = \mathtt{D}(\mathtt{D}(x_2, \mathtt{D}(x_1, \mathtt{E})), \mathtt{E}) \ .$$

From this tree, we remove the suffices for q_1 and q_2 at the variables x_1 and x_2, respectively. This was the third step in the Figure 6.5. It results in the tree $u = \mathtt{D}(\mathtt{D}(x_2, x_1), \mathtt{E})$. Therefore, we obtain in Step 4 the transition

$$\mathtt{a}(q_1, q_2) \to q_1(\mathtt{D}(\mathtt{D}(x_2, x_1), \mathtt{E})) \ .$$

Analogously, we obtain the transitions

$$\begin{array}{rl} \mathtt{d} & \to q_1(\mathtt{D}(\mathtt{E}, \mathtt{E})) \\ \mathtt{b}(q_2) & \to q_2(\mathtt{D}(x_1, \mathtt{D}(\mathtt{D}(\mathtt{E}, \mathtt{E}), \mathtt{E}))) \\ \mathtt{c} & \to q_2(\mathtt{E}) \ . \end{array}$$

For this transducer T', the greatest common suffices for all (essential) states equal y and the transducer is now earliest. ◁

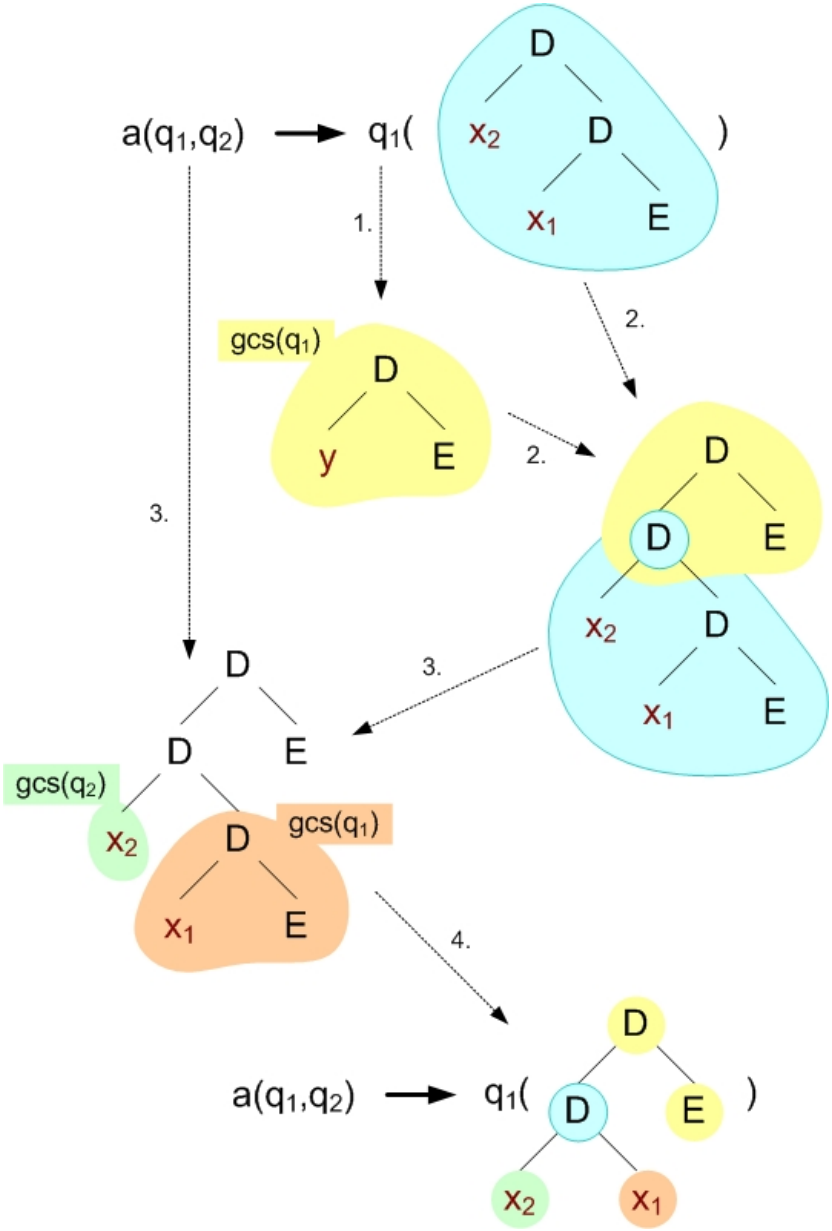


Figure 6.5: Construction of the right-hand side $R'(\mathtt{a}, q_1 q_2)$ in the earliest DBTT T' of Example 6.3.

6.1.3 Unified Transducers

For an earliest DBTT, contexts of states may disagree except for a pair of output trees.

Example 6.4. Recall the tree function τ_{mgu} of Example 5.8 (Figure 5.4). Similar to the two partitions presented in that example, we can construct different deterministic bottom-up tree transducers for τ_{mgu} with the same number of states. The input and output alphabets are

$$\Sigma = \{\mathtt{a}^{(0)}, \mathtt{b}^{(1)}, \mathtt{d}^{(1)}, \mathtt{e}^{(0)}, \mathtt{f}^{(1)}, \mathtt{g}^{(1)}\} \quad \text{and} \quad \Delta = \{\mathtt{A}^{(2)}, \mathtt{B}^{(1)}, \mathtt{D}^{(0)}, \mathtt{E}^{(0)}, \mathtt{G}^{(2)}\} \,,$$

respectively. First, consider the earliest transducer $T_1 = (Q, \Sigma, \Delta, R_1, F)$ with set of states $Q = \{q_1, q_1', q_2, q_2', q_3\}$ and R_1, F given by:

$$\begin{array}{rlll} \mathtt{e} & \to q_1(\mathtt{E}) & \mathtt{b}(q_1) & \to q_1(\mathtt{B}(x_1)) \\ \mathtt{a} & \to q_1'(\mathtt{E}) & \mathtt{b}(q_1') & \to q_1'(\mathtt{B}(x_1)) \\ \mathtt{d}(q_1) & \to q_2(x_1) & \mathtt{g}(q_2) & \to q_3(\mathtt{G}(\mathtt{D}, y)) \\ \mathtt{d}'(q_1) & \to q_2'(x_1) & \mathtt{g}(q_2') & \to q_3(\mathtt{G}(\mathtt{D}, \mathtt{B}(y))) \\ \mathtt{f}(q_1') & \to q_2'(\mathtt{E}) & & \end{array} \qquad \begin{array}{ll} F(q_1') & = y \\ F(q_2) & = \mathtt{A}(y, \mathtt{E}) \\ F(q_2') & = \mathtt{A}(\mathtt{B}(y), y) \\ F(q_3) & = y \end{array}$$

This transducer is illustrated in Figure 6.6. There is an additional dotted arrow from the $\mathtt{f}$-labeled transition to the state q_2, which we ignore for T_1. We observe that the states q_2 and q_2' are both essential and have the same contexts. The images of the context $\mathtt{g}(y)$,

$$\tau_{q_2}^{T_1}(\mathtt{g}(y)) = \mathtt{G}(\mathtt{D}, y) \quad \text{and} \quad \tau_{q_2'}^{T_1}(\mathtt{g}(y)) = \mathtt{G}(\mathtt{D}, \mathtt{B}(y))$$

differ only in the suffix $\mathtt{B}(y)$. The images of the context y are

$$\tau_{q_2}^{T_1}(y) = \mathtt{A}(y, \mathtt{E}) \quad \text{and} \quad \tau_{q_2'}^{T_1}(y) = \mathtt{A}(\mathtt{B}(y), y) \,.$$

If y is substituted by $\mathtt{B}(\mathtt{E})$ in the image at q_2 and $\mathtt{E}$ at q_2', they become equal. Thus, for each context c of q_2, we get

$$\tau_{q_2}^{T_1}(c) \cdot \mathtt{B}(\mathtt{E}) = \tau_{q_2'}^{T_1}(c) \cdot \mathtt{E} \,.$$

The transition with left-hand side $\mathtt{f}(q_1')$ produces $\mathtt{E}$ as output and reaches state q_2'. Thus, we can also define a transducer $T_2 = (Q, \Sigma, \Delta, R_2, F)$, which equals T_1 but $R_2(\mathtt{f}, q_1') = q_2(\mathtt{B}(\mathtt{E}))$ (by the dotted arrow in the figure). Both transducers are earliest and have the same number of states. ◁

We see that earliest deterministic bottom-up tree transducers may have unifiable states. Assume that $T = (Q, \Sigma, \Delta, R, F)$ is an earliest DBTT and that $q_1, q_2 \in Q$ are states. Every state of T defines a context function τ_q^T. We define the *most-general unifier* $\mathsf{mgu}(q_1, q_2)$ of the states q_1, q_2 as the most-general unifier of their context functions (cf. Page 61), i.e.,

$$\mathsf{mgu}(q_1, q_2) = \mathsf{mgu}(\tau_{q_1}^T, \tau_{q_2}^T) \,.$$

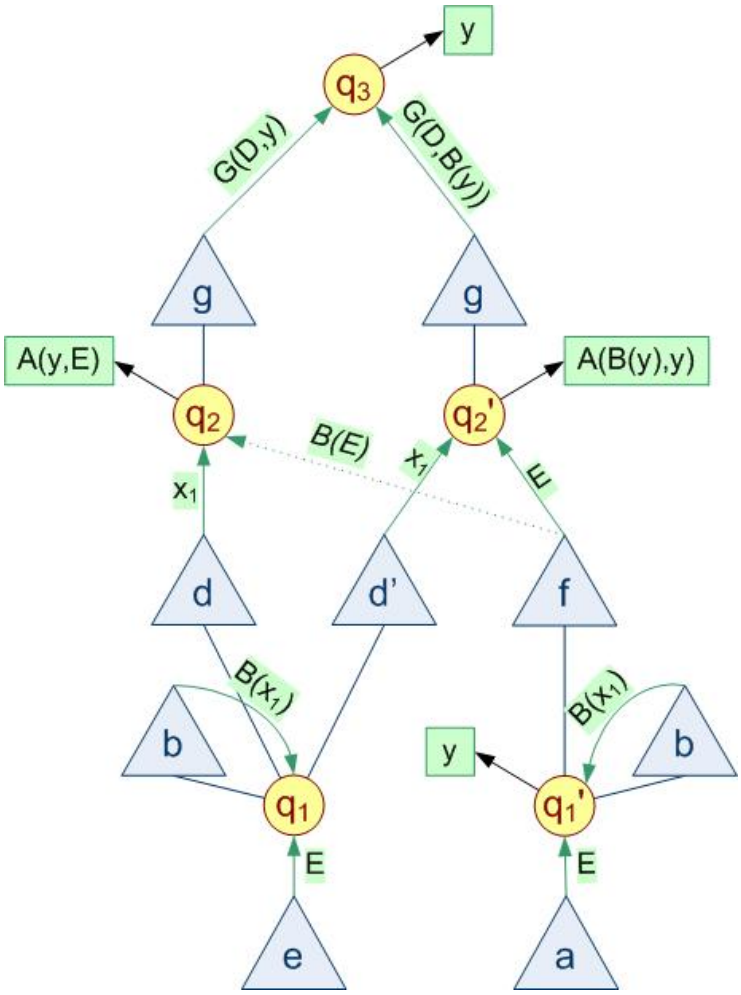


Figure 6.6: The transducers T_1 (without the `B(E)`-labeled arrow from '`f`' to q_2) and T_2 (without the `E`-labeled arrow from '`f`' to q_2') of Example 6.4.

Note that this most-general unifier is a pair if the two states have the same set of contexts, i.e., $\mathcal{C}^T(q_1) = \mathcal{C}^T(q_2)$. Otherwise, it is $\mathsf{mgu}(q_1, q_2) = \bot$. We call states q_1, q_2 *unifiable* if the most-general unifier is not $\bot$.

For unifiable states q_1 and q_2, we observe similar properties for the most-general unifier $\mathsf{mgu}(q_1, q_2) = \langle z_1, z_2 \rangle$ as for the most-general unifier of unifiable residuals:

- If q_i is inessential, then for every context c of q_i holds $\tau^T_{q_i}(c) \in \mathcal{T}_\Delta$. Therefore, $z_i = \top$.
- Moreover, z_1 contains y iff z_2 contains y. If both z_1 and z_2 contain y, the most-general unifier must equal $\langle y, y \rangle$, otherwise, T would not be earliest.

A ground term v is called *realizable* in a state q if v is part of the set of outputs of q. Ground terms v occurring in most-general unifiers of states are not necessarily realizable.

Definition 6.3 (Unified Earliest)**.** The earliest DBTT T is called *unified earliest* if no ground term in most-general unifiers of states of T is realizable.

In the following, we show that for every earliest DBTT, a unified earliest DBTT can be constructed in polynomial time. For this construction, the following result is required.

Theorem 6.5. *Assume that T is an earliest deterministic bottom-up tree transducer. Then all most-general unifiers $\mathsf{mgu}(q_1, q_2)$ can be constructed in polynomial time.*

Proof. Let the transducer $T = (Q, \Sigma, \Delta, R, F)$. We determine the greatest mapping $\mu : Q^2 \to \mathbb{D}_\Delta$ from pairs of states to candidate unifiers, which for all q, q' satisfies the following constraints:

1. $$\mu(q, q') \leq \begin{cases} \langle \top, \top \rangle & F(q) \text{ and } F(q') \text{ are undefined} \\ \bot & F(q) \text{ defined } \Leftrightarrow F(q') \text{ undefined} \\ \mathsf{mgu}(F(q), F(q')) & \text{otherwise} \end{cases}$$

2. Consider $\mathtt{a} \in \Sigma$ of rank m, $i \in [m]$, $q_1 \ldots, q_{i-1}, q_{i+1}, \ldots, q_m \in Q$. If it is not the case that the right-hand side $R(\mathtt{a}, q_1 \ldots q_{i-1} q q_{i+1} \ldots q_m)$ is defined iff $R(\mathtt{a}, q_1 \ldots q_{i-1} q' q_{i+1} \ldots q_m)$ is defined, then $\mu(q, q') \leq \bot$. Therefore, now assume that both right-hand sides $R(\mathtt{a}, q_1 \ldots q_{i-1} q q_{i+1} \ldots q_m)$ and $R(\mathtt{a}, q_1 \ldots q_{i-1} q' q_{i+1} \ldots q_m)$ are defined and equal $q_0(z)$ and $q'_0(z')$, respectively.
   - If $\mu(q_0, q'_0) = \bot$, then $\mu(q, q') \leq \bot$.
   - Assume $\mu(q_0, q'_0) = \langle s, s' \rangle$ where both s and s' contain y, and one of them equals y. If $s' \cdot z = z_1 \cdot u$ and $s \cdot z' = z_1 \cdot u'$ for some trees $u, u' \in \mathcal{T}_\Delta(x_i)$, then $\mu(q, q') \leq \mathsf{mgu}(u[y/x_i], u'[y/x_i])$. If no such decompositions exist, we have $\mu(q, q') \leq \bot$.

- Assume that $\mu(q_0, q_0') = \langle v, v' \rangle$ for ground terms $v, v' \in \mathcal{T}_\Delta$. If a variable $x_j \neq x_i$ occurs in z or z', then we have $\mu(q, q') \leq \bot$. Now assume that $z[y/x_i], z'[y/x_i] \in \mathcal{T}_\Delta(y)$ and, that $v = z[v_i/x_i]$ and $v' = z'[v_i'/x_i]$ for some $v_i, v_i' \in \mathcal{T}_\Delta$. If no such decomposition exists, we have $\mu(q, q') \leq \bot$.

  Therefore, assume that such a decomposition exists.

  If $z[y/x_i], z'[y/x_i] \in \hat{\mathcal{T}}_\Delta(y)$, then $\mu(q, q') \leq \langle v_i, v_i' \rangle$.

  If $z[y/x_i] \in \hat{\mathcal{T}}_\Delta(y)$ and $z' \in \mathcal{T}_\Delta$, then $\mu(q, q') \leq \langle v_i, \top \rangle$.

  If $z \in \mathcal{T}_\Delta$ and $z'[y/x_i] \in \hat{\mathcal{T}}_\Delta(y)$, then $\mu(q, q') \leq \langle \top, v_i' \rangle$.
- Now assume that $\mu(q_0, q_0') = \langle v, \top \rangle$ for a ground term v. If a variable $x_j \neq x_i$ occurs in z, then $\mu(q, q') \leq \bot$. Assume that $z[y/x_i] \in \mathcal{T}_\Delta(y)$ and $v = z[v_i/x_i]$ for some $v_i \in \mathcal{T}_\Delta$.

  If no such decomposition exists, we have $\mu(q, q') \leq \bot$.

  If such a decomposition exists and $z[y/x_i] \in \hat{\mathcal{T}}_\Delta(y)$, then we have $\mu(q, q') \leq \langle v_i, \top \rangle$.
- The case where $\mu(q_0, q_0')$ equals $\langle \top, v \rangle$ for a ground term $v \in \mathcal{T}_\Delta$, is analogous.

Each constraint induced by a pair of final outputs $F(q)$, $F(q')$ as well as each constraint induced by a matching pair of transitions

$$\mathsf{a}(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_m) \to q_0(z) \text{ and}$$
$$\mathsf{a}(q_1, \ldots, q_{i-1}, q', q_{i+1}, \ldots, q_m) \to q_0'(z')$$

is monotonic w.r.t. the ordering $\leq$ and distributes over pairwise greatest lower bounds.

By induction on the length of contexts $c \in \hat{\mathcal{T}}_\Delta(y)$, we verify that for each pair of states q, q' it holds:

- If $\tau_q(c)$ and $\tau_{q'}(c)$ are defined, then $\mu(q, q') \leq \mathsf{mgu}(\tau_q(c), \tau_{q'}(c))$.
- If either $\tau_q(c)$ or $\tau_{q'}(c)$ is defined and not the other, then $\mu(q, q') = \bot$.

It follows that $\mu(q, q') \leq \mathsf{mgu}(q, q')$ for all states q, q'. Since mgu is a solution of the constraint system, thus, it is the greatest solution.

The complete lattice $(\mathbb{D}, \leq)$ of candidate unifiers (cf. Page 20) has finite height, i.e., each strictly ascending chain $d_1 \lneq d_2 \lneq \ldots \lneq d_k$ has at most length $k = 4$. Using the compacted representations of trees where isomorphic subterms are shared, this construction of the most-general unifiers works in polynomial time. □

Assume now that we are given all most-general unifiers of an earliest DBTT T. Then we can construct a unified earliest transducer T', which is equivalent to T. The idea of the proof is similar as for partitions, cf. Theorem 5.10. Realizable ground terms (and parts of it) are stored in new inessential states and with that are postponed. We have:

Theorem 6.6. *For each earliest DBTT T, a DBTT T' can be constructed in polynomial time with the following properties:*

- *T' is equivalent to T and*
- *T' is unified earliest.*

Proof. Let $T = (Q, \Sigma, \Delta, R, F)$ be an earliest DBTT. By Theorem 6.5, we may assume that we are given all most-general unifiers $\mathsf{mgu}(q, q')$ for states $q, q' \in Q$. Then we construct the unified earliest transducer $T' = (Q', \Sigma, \Delta, R', F')$ in two steps.

First, we introduce new states and get a transducer T_1. Whenever an output v of an input u at state q is produced by T, which will contribute to a ground unifier of q, then the computation on u is redirected to a new state $\langle q, v\rangle$, which memorizes v and does produce $*$ only. Instead, the output v is delayed to the images of the contexts. This implies that the new state $\langle q, v\rangle$ is inessential. Furthermore, for states q' used to evaluate subtrees of u whose outputs v' may contribute to v, further states $\langle q', v'\rangle$ should be introduced.

Formally, the DBTT $T_1 = (Q_1, \Sigma, \Delta, R_1, F_1)$ is defined as follows. We denote by $V \subseteq \mathcal{T}_\Delta \cup \{\bot\}$ the set of subterms of terms occurring as ground unifiers of states or $\bot$. Let Q_1 denote the set of pairs

$$Q_1 = \{\langle q, v\rangle \mid q \in Q, v \in V\} .$$

Let R_1 denote the least set of transitions, which contains for each transition $\mathtt{a}(q_1, \ldots, q_m) \to q(z)$ of R the following transitions. Assume $v_1, \ldots, v_m \in V$ and let $v = z[v_1, \ldots, v_m]$.

- If $v \in V$, then $\mathtt{a}(\langle q_1, v_1\rangle, \ldots, \langle q_m, v_m\rangle) \to \langle q, v\rangle(*) \in R_1$.
- If $v \notin V$, then $\mathtt{a}(\langle q_1, v_1\rangle, \ldots, \langle q_m, v_m\rangle) \to \langle q, \bot\rangle(z[v'_1, \ldots, v'_m]) \in R_1$ where $v'_i = x_i$ if $v_i = \bot$, and $v'_i = v_i$ otherwise.

Let $\langle q, v\rangle \in Q_1$. The final function F_1 is defined for $\langle q, v\rangle$ iff F is defined for q where $F_1(\langle q, \bot\rangle) = F(q)$ and $F_1(\langle q, v\rangle) = F(q) \cdot v$ if $v \neq \bot$. Some of the new states $\langle q, v\rangle$ of T_1 for $v \neq \bot$ may be unreachable. The unified earliest transducer $T' = (Q', \Sigma, \Delta, R', F')$, therefore, is defined as the trim DBTT equivalent to T_1, obtained according to Proposition 4.1.

By induction on the length of contexts c and depth of input trees u, we obtain:

- $\forall \langle q, v\rangle \in Q' :\ c \in \mathcal{C}^T(q)$ iff $c \in \mathcal{C}^{T'}(\langle q, v\rangle)$;
- $\forall \langle q, \bot\rangle \in Q', c \in \mathcal{C}^T(q) :\ \tau^T_q(c) = \tau^{T'}_{\langle q,\bot\rangle}(c)$;
- $\forall \langle q, v\rangle \in Q', v \neq \bot, c \in \mathcal{C}^T(q) :\ \tau^T_q(c) \cdot v = \tau^{T'}_{\langle q,v\rangle}(c)$;
- $\forall u \in \mathcal{T}_\Sigma$: $[\![u]\!]^T = q(v)$ iff $[\![u]\!]^{T'} = \begin{cases} \langle q, v\rangle(*) & v \in V \\ \langle q, \bot\rangle(v) & v \notin V \end{cases}$

The number of outputs produced at a state q of T, which is not produced by $\langle q, \bot\rangle$ in T', is finite (it is bounded by the number of subtrees of ground unifiers of T). Thus, if q is an essential state of T, $\langle q, \bot\rangle$ is an essential state of T' and each state $\langle q, v\rangle$ with $v \neq \bot$ is inessential. Thus, the transducer T' is *proper*. Since the images of contexts of state q w.r.t. T equal the images of contexts of $\langle q, \bot\rangle$ w.r.t. T', and the new states $\langle q, v\rangle$ with $v \neq \bot$ are inessential, the transducer T' is still *earliest*. Let $v, v' \in \mathcal{T}_\Delta$. The most-general unifier of T' is given by

- $\mathsf{mgu}^{T'}(\langle q', \bot\rangle, \langle q, \bot\rangle) = \mathsf{mgu}^{T}(q', q)$,
- $\mathsf{mgu}^{T'}(\langle q', \bot\rangle, \langle q, v\rangle) = \begin{cases} \langle v', \top\rangle & \mathsf{mgu}^{T}(q', q) = \langle v', v\rangle \\ \langle v, \top\rangle & \mathsf{mgu}^{T}(q', q) = \langle y, y\rangle \\ \bot & \text{otherwise} \end{cases}$
- $\mathsf{mgu}^{T'}(\langle q', v'\rangle, \langle q, v\rangle) = \begin{cases} \langle \top, \top\rangle & \mathsf{mgu}^{T}(q', q) = \langle v', v\rangle \\ \bot & \text{otherwise} \end{cases}$

The new bottom-up tree transducer T' is *unified earliest*: Assume there is a unifier $\mathsf{mgu}(\langle q_1, v_1\rangle, \langle q_2, v_2\rangle) = \langle v_1', v_2'\rangle$ in T' with ground $v_1' \in \mathcal{T}_\Delta$. We show that v_1' is not realizable in $\langle q_1, v_1\rangle$. Since $v_1' \neq \top$, the state $\langle q_1, v_1\rangle$ is essential and with that, $v_1 = \bot$.

- If $v_2 = \bot$, then $\langle v_1', v_2'\rangle$ was a most-general unifier in T of q_1, q_2. Consequently, $\langle q_1, v_1'\rangle$ is a state in Q'.
- If $v_2 \neq \bot$ and $v_2' = \top$ and $\mathsf{mgu}^T(q_1, q_2) \neq \langle y, y\rangle$, then $\langle v_1', v_2\rangle$ was a most-general unifier in T for q_1, q_2. Thus, $\langle q_1, v_1'\rangle \in Q'$.
- If $v_2 \neq \bot$ and $v_2' = \top$ and $\mathsf{mgu}^T(q_1, q_2) \neq \langle y, y\rangle$, then $v_2 = v_1' \in V$ and $\langle q_1, v_1'\rangle$ is a state in Q'.

Now we show that T and T' are *equivalent*: Consider a tree $u = c \cdot u'$ with $[\![u']\!]^T = q(v)$. If $v \neq \bot$ and $\langle q, v\rangle \in Q'$ we have:

$$\tau^T(u) = \tau_q^T(c) \cdot v = \tau^{T'}_{\langle q,v\rangle}(c) = \tau^{T'}_{\langle q,v\rangle}(c) \cdot * = \tau^{T'}(u)$$

Otherwise, we have $[\![u']\!]^{T'} = \langle q, \bot\rangle(v)$ and:

$$\tau^T(u) = \tau_q^T(c) \cdot v = \tau^{T'}_{\langle q,v\rangle}(c) \cdot v = \tau^{T'}(u)\ . \qquad \square$$

Example 6.5. Consider again the transducer $T_1 = (Q, \Sigma, \Delta, R_1, F)$ of Example 6.4. The most-general unifiers are

$$\mathsf{mgu}(q_2, q_2') = \langle \mathtt{B}(\mathtt{E}), \mathtt{E}\rangle$$

and $\mathsf{mgu}(q, q') = \bot$, otherwise. We get the set $V = \{\mathtt{B}(\mathtt{E}), \mathtt{E}, \bot\}$ of subterms of terms occurring as ground unifiers of states or $\bot$. All states of $Q \times V$ are possible new states. Except from $\langle q_3, \mathtt{B}(\mathtt{E})\rangle$ and $\langle q_3, \mathtt{E}\rangle$, all are reachable. Starting with

The unified earliest transducer $T_1' = (Q', \Sigma, \Delta, R_1', F')$ has the set of states

$$Q' = \{\langle q_1, \mathtt{E}\rangle, \langle q_1, \mathtt{B(E)}\rangle, \langle q_1, \bot\rangle, \langle q_1', \mathtt{E}\rangle, \langle q_1', \mathtt{B(E)}\rangle, \langle q_1', \bot\rangle,$$
$$\langle q_2, \mathtt{E}\rangle, \langle q_2, \mathtt{B(E)}\rangle, \langle q_2, \bot\rangle, \langle q_2', \mathtt{E}\rangle, \langle q_2', \mathtt{B(E)}\rangle, \langle q_2', \bot\rangle, \langle q_3, \bot\rangle\} .$$

The transition function R_1' is given by

$$\begin{aligned}
\mathtt{a} &\to \langle q_1', \mathtt{E}\rangle(*) & \mathtt{b}(\langle q_1', \mathtt{E}\rangle) &\to \langle q_1', \mathtt{B(E)}\rangle(*)\\
\mathtt{e} &\to \langle q_1, \mathtt{E}\rangle(*) & \mathtt{b}(\langle q_1, \mathtt{E}\rangle) &\to \langle q_1, \mathtt{B(E)}\rangle(*)\\
\mathtt{b}(\langle q_1', \mathtt{B(E)}\rangle) &\to \langle q_1', \bot\rangle(\mathtt{B(B(E))}) & \mathtt{b}(\langle q_1', \bot\rangle) &\to \langle q_1', \bot\rangle(\mathtt{B}(x_1))\\
\mathtt{b}(\langle q_1, \mathtt{B(E)}\rangle) &\to \langle q_1, \bot\rangle(\mathtt{B(B(E))}) & \mathtt{b}(\langle q_1, \bot\rangle) &\to \langle q_1, \bot\rangle(\mathtt{B}(x_1))\\
\mathtt{d}(\langle q_1, \mathtt{E}\rangle) &\to \langle q_2, \mathtt{E}\rangle(*) & \mathtt{g}(\langle q_2, \mathtt{E}\rangle) &\to \langle q_3, \bot\rangle(\mathtt{G(D,E)})\\
\mathtt{d}'(\langle q_1, \mathtt{E}\rangle) &\to \langle q_2', \mathtt{E}\rangle(*) & \mathtt{g}(\langle q_2', \mathtt{E}\rangle) &\to \langle q_3, \bot\rangle(\mathtt{G(D,B(E))})\\
\mathtt{d}(\langle q_1, \mathtt{B(E)}\rangle) &\to \langle q_2, \mathtt{B(E)}\rangle(*) & \mathtt{g}(\langle q_2, \mathtt{B(E)}\rangle) &\to \langle q_3, \bot\rangle(\mathtt{G(D,B(E))})\\
\mathtt{d}'(\langle q_1, \mathtt{B(E)}\rangle) &\to \langle q_2', \mathtt{B(E)}\rangle(*) & \mathtt{g}(\langle q_2', \mathtt{B(E)}\rangle) &\to \langle q_3, \bot\rangle(\mathtt{G(D,B(B(E)))})\\
\mathtt{d}(\langle q_1, \bot\rangle) &\to \langle q_2, \bot\rangle(x_1) & \mathtt{g}(\langle q_2, \bot\rangle) &\to \langle q_3, \bot\rangle(\mathtt{G(D},x_1))\\
\mathtt{d}'(\langle q_1, \bot\rangle) &\to \langle q_2', \bot\rangle(x_1) & \mathtt{g}(\langle q_2', \bot\rangle) &\to \langle q_3, \bot\rangle(\mathtt{G(D,B}(x_1)))\\
\mathtt{f}(\langle q_1', \mathtt{E}\rangle) &\to \langle q_2', \mathtt{E}\rangle(*) & \mathtt{f}(\langle q_1', \mathtt{B(E)}\rangle) &\to \langle q_2', \mathtt{E}\rangle(*)\\
\mathtt{f}(\langle q_1', \bot\rangle) &\to \langle q_2', \mathtt{E}\rangle(*)
\end{aligned}$$

and the final function is F' defined as follows:

$$\begin{aligned}
F'(\langle q_1', \mathtt{E}\rangle) &= \mathtt{E} & F'(\langle q_1', \mathtt{B(E)}\rangle) &= \mathtt{B(E)}\\
F'(\langle q_2, \mathtt{E}\rangle) &= \mathtt{A(E,E)} & F'(\langle q_2, \mathtt{B(E)}\rangle) &= \mathtt{A(B(E),E)}\\
F'(\langle q_2', \mathtt{E}\rangle) &= \mathtt{A(B(E),E)} & F'(\langle q_2', \mathtt{B(E)}\rangle) &= \mathtt{A(B(B(E)),B(E))}\\
F'(\langle q_2, \bot\rangle) &= \mathtt{A}(y, \mathtt{E}) & F'(\langle q_2', \bot\rangle) &= \mathtt{A(B}(y), y)\\
F'(\langle q_1', \bot\rangle) &= y & F'(\langle q_3, \bot\rangle) &= y
\end{aligned}$$

Figure 6.7: The unified earliest DBTT T_1' of Example 6.5.

left-hand side $\mathtt{a}$, we get the new transition $\mathtt{a} \to \langle q_1', \mathtt{E}\rangle(*)$ because $\mathtt{E} \in V$. Furthermore, for the transition $\mathtt{b}(q_1') \to q_1'(\mathtt{B}(x_1))$, we get the transition

$$\mathtt{b}(\langle q_1', \mathtt{E}\rangle) \to \langle q_1', \mathtt{B}(\mathtt{E})\rangle(*)$$

because $\mathtt{B}(x_1)[\mathtt{E}/x_1] = \mathtt{B}(\mathtt{E}) \in V$. Now consider the left-hand side $\mathtt{b}(\langle q_1', \mathtt{B}(\mathtt{E})\rangle)$. The potential output $\mathtt{B}(x_1)[\mathtt{B}(\mathtt{E})/x_1] = \mathtt{B}(\mathtt{B}(\mathtt{E}))$ is not in V. Thus, the right-hand side should be

$$R_1'(\mathtt{b}, \langle q_1', \mathtt{B}(\mathtt{E})\rangle) = \langle q_1', \bot\rangle(\mathtt{B}(\mathtt{B}(\mathtt{E}))) \; .$$

In addition, for the left-hand side $\mathtt{b}(\langle q_1', \bot\rangle)$, we get the transition

$$\mathtt{b}(\langle q_1', \bot\rangle) \to \langle q_1', \bot\rangle(\mathtt{B}(x_1)) \; .$$

Now for all states $\langle q_1', v\rangle$ for $v \in V$, we get the transition

$$\mathtt{f}(\langle q_1', v\rangle) \to \langle q_2', \mathtt{E}\rangle(*) \; .$$

The whole unified earliest transducer T_1' is given in Figure 6.7.

On the other hand, if we start with DBTT T_2, we get a unified earliest transducer T_2' with the transition

$$\mathtt{f}(\langle q_1', v\rangle) \to \langle q_2, \mathtt{B}(\mathtt{E})\rangle(*)$$

for all $v \in \{\mathtt{B}(\mathtt{E}), \mathtt{E}, \bot\}$. However, both $\langle q_2', \mathtt{E}\rangle$ in T_1' and $\langle q_2, \mathtt{B}(\mathtt{E})\rangle$ in T_2' are inessential. Since they define the same context function, they are equivalent. Altogether, the resulting DBTTs T_1' and T_2' are isomorphic up to renaming. ◁

6.1.4 Minimal Transducers

In the last example, we get two equivalent tree transducers, which are unified earliest. But the construction of these transducers yields in a number of redundant states. Consider for example the states $\langle q_1', \mathtt{E}\rangle$, $\langle q_1', \mathtt{B}(\mathtt{E})\rangle$, and $\langle q_1', \bot\rangle$ of transducer T_1' in Figure 6.7. If we merge them, the transducer remains unified earliest. Moreover, the resulting DBTT is smaller.

Thus, as a last step of the normalization of deterministic bottom-up tree transducers, we merge equivalent states by preserving the properties of a unified earliest DBTT. Therefore, we consider a congruence relation on states. It is similarly defined as the congruence relation $\sim_r$ on subtrees in Section 5.2.6. Let $T = (Q, \Sigma, \Delta, R, F)$ be a DBTT and let $\sim_T'$ denote the smallest equivalence relation with the following properties:

- If $\mathsf{mgu}(q, q') = \langle y, y\rangle$ or $\mathsf{mgu}(q, q') = \langle \top, \top\rangle$ then $q \sim_T' q'$;
- Assume that $\mathsf{mgu}(q, q_1) = \langle \top, s_1\rangle$ for some ground term $s_1 \in \mathcal{T}_\Delta$. If for all states q_2 with $\mathsf{mgu}(q, q_2) = \langle \top, s_2\rangle$ for some $s_2 \neq \top$, $\mathsf{mgu}(q_1, q_2) = \langle y, y\rangle$ holds then $q \sim_T' q_1$.

Then, the relation $\sim_T$ is the greatest equivalence relation, which is a refinement of $\sim'_T$ such that, $q_1 \sim_T q_2$, whenever for every symbol $\mathtt{a} \in \Sigma$ of rank m, every $i \in [m]$, and all states $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_m \in Q$, the following holds. There is a transition $\mathtt{a}(p_1, \ldots, p_{i-1}, \boldsymbol{q_1}, p_{i+1}, \ldots, p_m) \to q'_1(z_1)$ in R iff there is a transition $\mathtt{a}(p_1, \ldots, p_{i-1}, \boldsymbol{q_2}, p_{i+1}, \ldots, p_m) \to q'_2(z_2)$ in R. If such two transitions exist then $q'_1 \sim_T q'_2$. Two states q_1, q_2 are called *similar* if $q_1 \sim_T q_2$.

Definition 6.4 (Minimal)**.** A unified earliest transducer $T = (Q, \Sigma, \Delta, R, F)$ is said to be *minimal* iff all distinct states $q_1, q_2 \in Q$ are not similar, i.e., $q_1 \not\sim_T q_2$.

Theorem 6.7. *For each unified earliest DBTT T, a unified earliest DBTT T' can be constructed in polynomial time with the following properties:*

- *T' is equivalent to T*
- *T' is minimal*
- *$|T'| \leq |T|$.*

Proof. Let $T = (Q, \Sigma, \Delta, R, F)$ be a unified earliest DBTT. By fixpoint iteration, we compute the equivalence relation $\sim_T$ on Q. Now we build a transducer T' with the equivalence classes of $\sim_T$ as states. Let $[q] = \{q' \mid q \sim_T q'\}$ the equivalence class of q. We call $[q]$ inessential if all states in $[q]$ are inessential. Otherwise, it is called essential. For each class $[q]$, we mark a representative state $p_q \in Q$, which is essential iff $[q]$ is essential.

Formally, we get $T' = (Q', \Sigma, \Delta, R', F')$ with $Q' = \{[q] \mid q \in Q\}$. The function F' is given by $F'([q]) = F(p_q)$. For R', assume that $q_1, \ldots, q_m \in Q$ are representatives of their classes and that $\mathtt{a}(q_1, \ldots, q_m) \to q(z) \in R$.

- If q is essential, then $\mathtt{a}([q_1], \ldots, [q_m]) \to [q](z) \in R'$.
- If p_q is inessential, then $\mathtt{a}([q_1], \ldots, [q_m]) \to [q](z) \in R'$.
- Otherwise, if $\mathsf{mgu}(q, p_q) = \langle \top, s \rangle$, then $\mathtt{a}([q_1], \ldots, [q_m]) \to [q](s) \in R'$.

By induction on the depth of input trees u, we obtain for all $u \in \mathcal{T}_\Sigma$:

$$[\![u]\!]^{T'} = [q](z) \quad \text{iff} \quad \exists q' \in [q] \text{ with } [\![u]\!]^T = \begin{cases} q'(z) & \text{if } \mathsf{mgu}(q', p_q) = \langle y, y \rangle \\ q'(z) & \text{if } \mathsf{mgu}(q', p_q) = \langle \top, \top \rangle \\ q'(*) & \text{if } \mathsf{mgu}(q', p_q) = \langle \top, z \rangle \end{cases}$$

In addition, by induction on the length of context c, we get for all $c \in \hat{\mathcal{T}}_\Sigma(y)$:

$$\tau^{T'}_{[q]}(c) = z \quad \text{iff} \quad \tau^{T}_{p_q}(c) = z$$

It follows that $\tau^T = \tau^{T'}$ and that T' is trim, proper, and earliest.

To prove that T' is unified earliest, assume that $\mathsf{mgu}(p_1, p_2) = \langle v_1, v_2 \rangle$ for states $p_i \in Q'$ and $v_1 \in \mathcal{T}_\Delta$ is realizable in p_1. If v_2 is ground, it follows that there are states $q_1 \in p_1$ and $q_2 \in p_2$ of T with $\mathsf{mgu}(q_1, q_2) = \langle v_1, v_2 \rangle$ and v_1 is realizable in q_1. If $v_2 = \top$, it follows that $q_1 \sim_T q_2$. □

In the following, we show that the minimal transducer is unique. Before that, let us consider the construction of the minimal DBTT in an example.

Example 6.6. Consider again the unified earliest DBTT of Example 6.5, i.e., $T_1' = (Q', \Sigma, \Delta, R_1', F')$. The complete unified earliest transducer is presented on Page 99. Let $\sim_T'$ and $\sim_T$ denote the equivalence relations $\sim_{T_1'}'$ and $\sim_{T_1'}$, respectively. We compute the following most-general unifiers different to $\bot$:

$$\mathsf{mgu}(\langle q_2, \bot\rangle, \langle q_2', \bot\rangle) = \langle \mathtt{B}(\mathtt{E}), \mathtt{E}\rangle \tag{6.1}$$

$$\mathsf{mgu}(\langle q_2, \mathtt{B}(\mathtt{E})\rangle, \langle q_2', \mathtt{E}\rangle) = \langle \top, \top\rangle \tag{6.2}$$

$$\mathsf{mgu}(\langle q_2, \mathtt{B}(\mathtt{E})\rangle, \langle q_2', \bot\rangle) = \langle \top, \mathtt{E}\rangle \tag{6.3}$$

$$\mathsf{mgu}(\langle q_2, \bot\rangle, \langle q_2', \mathtt{E}\rangle) = \langle \mathtt{B}(\mathtt{E}), \top\rangle \tag{6.4}$$

Additionally, for every pair $(\langle q, \bot\rangle, \langle q, v\rangle)$ of states with $q \in \{q_1, q_1', q_2, q_2'\}$ and $v \in \{\mathtt{E}, \mathtt{B}(\mathtt{E})\}$, the most-general unifier is

$$\mathsf{mgu}(\langle q, \bot\rangle, \langle q, v\rangle) = \langle v, \top\rangle\ . \tag{6.5}$$

Every state q of the form $\langle q_2, v\rangle$ or $\langle q_2', v\rangle$ for $v \in \{\mathtt{E}, \mathtt{B}(\mathtt{E}), \bot\}$ only occurs on the left-hand side $\mathsf{g}(q)$ with $\langle q_3, \bot\rangle$ on the corresponding right-hand side. Thus, two states q, q' of this form are similar if $q \sim_T' q'$. With Equation 6.2, this holds for the pair $\langle q_2, \mathtt{B}(\mathtt{E})\rangle, \langle q_2', \mathtt{E}\rangle$, i.e., $\langle q_2, \mathtt{B}(\mathtt{E})\rangle \sim_T \langle q_2', \mathtt{E}\rangle$. Also, we get with Equation 6.5:

$$\langle q_2, \mathtt{E}\rangle \sim_T \langle q_2, \bot\rangle \quad \text{and} \quad \langle q_2', \mathtt{B}(\mathtt{E})\rangle \sim_T \langle q_2', \bot\rangle$$

Now consider $\langle q_2, \mathtt{B}(\mathtt{E})\rangle$. There are two essential states, which have a most-general unifier of the form $\langle \top, v\rangle$ with this inessential state: $\langle q_2', \bot\rangle$ and $\langle q_2, \bot\rangle$ by Equations 6.3 and 6.5, respectively. Thus, $\langle q_2, \mathtt{B}(\mathtt{E})\rangle$ is not similar to any essential state. The same follows for $\langle q_2', \mathtt{E}\rangle$.

On the other hand, consider the states with q_1 as first component. The left-hand side $\mathtt{d}(\langle q_1, \top\rangle)$ has $\langle q_2, \top\rangle$ on the right-hand side, whereas for $\langle q_1, \mathtt{B}(\mathtt{E})\rangle$ it is $R(\mathtt{d}, \langle q_1, \mathtt{B}(\mathtt{E})\rangle) = \langle q_2, \mathtt{B}(\mathtt{E})\rangle(*)$. Since $\langle q_2, \top\rangle$ and $\langle q_2, \mathtt{B}(\mathtt{E})\rangle$ are not similar, also

$$\langle q_1, \top\rangle \not\sim_T \langle q_1, \mathtt{B}(\mathtt{E})\rangle$$

follows. Analogously, we get with left-hand side $\mathtt{d}'(\langle q_1, \top\rangle)$ and $\mathtt{d}'(\langle q_1, \mathtt{E}\rangle)$ that these states are not similar. Additionally, for both labels $\mathtt{d}$ and $\mathtt{d}'$ follows

$$\langle q_1, \mathtt{E}\rangle \not\sim_T \langle q_1, \mathtt{B}(\mathtt{E})\rangle\ .$$

Last, we have the states, with first component q_1'. There, we get with Equation 6.5 that $\langle q_1', \mathtt{E}\rangle \sim_T' \langle q_1', \bot\rangle$ and $\langle q_1', \bot\rangle \sim_T' \langle q_1', \mathtt{B}(\mathtt{E})\rangle$. Since $\sim_T'$ is the smallest equivalence relation such that this holds, this implies

$$\langle q_1', \mathtt{E}\rangle \sim_T' \langle q_1', \mathtt{B}(\mathtt{E})\rangle\ .$$

The minimal transducer $T_{min} = (Q_{min}, \Sigma, \Delta, R_{min}, F_{min})$ has the set of states

$$Q_{min} = \{[\langle q_1, \mathtt{E}\rangle],\ [\langle q_1, \mathtt{B}(\mathtt{E})\rangle],\ [\langle q_1, \bot\rangle],\ [\langle q_1', \bot\rangle],\\ [\langle q_2, \bot\rangle],\ [\langle q_2', \mathtt{E}\rangle],\ [\langle q_2', \bot\rangle],\ [\langle q_3, \bot\rangle]\ \}\ .$$

The transition function R_{min} is given by

$$\begin{array}{rlrl}
\mathtt{a} &\to [\langle q_1', \bot\rangle](\mathtt{E}) & \mathtt{b}([\langle q_1', \bot\rangle]) &\to [\langle q_1', \bot\rangle](\mathtt{B}(x_1))\\
\mathtt{e} &\to [\langle q_1, \mathtt{E}\rangle](*) & \mathtt{b}([\langle q_1, \mathtt{E}\rangle]) &\to [\langle q_1, \mathtt{B}(\mathtt{E})\rangle](*)\\
\mathtt{b}([\langle q_1, \mathtt{B}(\mathtt{E})\rangle]) &\to [\langle q_1, \bot\rangle](\mathtt{B}(\mathtt{B}(\mathtt{E}))) & \mathtt{b}([\langle q_1, \bot\rangle]) &\to [\langle q_1, \bot\rangle](\mathtt{B}(x_1))\\
\mathtt{d}([\langle q_1, \mathtt{E}\rangle]) &\to [\langle q_2, \bot\rangle](\mathtt{E}) & &\\
\mathtt{d}'([\langle q_1, \mathtt{E}\rangle]) &\to [\langle q_2', \mathtt{E}\rangle](*) & \mathtt{g}([\langle q_2', \mathtt{E}\rangle]) &\to [\langle q_3, \bot\rangle](\mathtt{G}(\mathtt{D}, \mathtt{B}(\mathtt{E})))\\
\mathtt{d}'([\langle q_1, \mathtt{B}(\mathtt{E})\rangle]) &\to [\langle q_2', \bot\rangle](\mathtt{B}(\mathtt{E})) & &\\
\mathtt{d}([\langle q_1, \mathtt{B}(\mathtt{E})\rangle]) &\to [\langle q_2', \mathtt{E}\rangle](*) & &\\
\mathtt{d}([\langle q_1, \bot\rangle]) &\to [\langle q_2, \bot\rangle](x_1) & \mathtt{g}([\langle q_2, \bot\rangle]) &\to [\langle q_3, \bot\rangle](\mathtt{G}(\mathtt{D}, x_1))\\
\mathtt{d}'([\langle q_1, \bot\rangle]) &\to [\langle q_2', \bot\rangle](x_1) & \mathtt{g}([\langle q_2', \bot\rangle]) &\to [\langle q_3, \bot\rangle](\mathtt{G}(\mathtt{D}, \mathtt{B}(x_1)))\\
\mathtt{f}([\langle q_1', \bot\rangle]) &\to [\langle q_2', \mathtt{E}\rangle](*) & &
\end{array}$$

and the final function is F_{min} defined as follows:

$$\begin{array}{ll}
F_{min}([\langle q_2', \mathtt{E}\rangle]) = \mathtt{A}(\mathtt{B}(\mathtt{E}), \mathtt{E}) & \\
F_{min}([\langle q_2, \bot\rangle]) = \mathtt{A}(y, \mathtt{E}) & F_{min}([\langle q_2', \bot\rangle]) = \mathtt{A}(\mathtt{B}(y), y)\\
F_{min}([\langle q_1', \bot\rangle]) = y & F_{min}([\langle q_3, \bot\rangle]) = y
\end{array}$$

Figure 6.8: The minimal DBTT T_{min} of Example 6.6.

For each of these states we have the right-hand sides

$$R'_1(\mathtt{f}, \langle q'_1, v\rangle) = \langle q'_2, \mathtt{E}\rangle(*)$$
$$R'_1(\mathtt{b}, \langle q'_1, v\rangle) = \langle q'_1, v'\rangle(z)$$

for $v \in \{\mathtt{E}, \mathtt{B}(\mathtt{E}), \bot\}$ and suitable $v' \in \{\mathtt{B}(\mathtt{E}), \bot\}$ and $z \in \{*, \mathtt{B}(x_1)\}$. Thus, we get that $\langle q'_1, \mathtt{E}\rangle, \langle q'_1, \bot\rangle$, and $\langle q'_1, \mathtt{B}(\mathtt{E})\rangle$ are equivalent under $\sim_T$. Summarizing, we get the following equivalence classes of $\sim_T$ with more than one element:

$$[\langle q_2, \bot\rangle] = \{\langle q_2, \bot\rangle, \langle q_2, \mathtt{E}\rangle\}$$
$$[\langle q'_2, \mathtt{E}\rangle] = \{\langle q_2, \mathtt{B}(\mathtt{E})\rangle, \langle q'_2, \mathtt{E}\rangle\}$$
$$[\langle q'_2, \bot\rangle] = \{\langle q'_2, \bot\rangle, \langle q'_2, \mathtt{B}(\mathtt{E})\rangle\}$$
$$[\langle q'_1, \bot\rangle] = \{\langle q'_1, \bot\rangle, \langle q'_1, \mathtt{B}(\mathtt{E})\rangle, \langle q'_1, \mathtt{E}\rangle\}$$

To get the correct transitions, let us consider, e.g., the transition $\mathtt{a} \to \langle q'_1, \mathtt{E}\rangle(*)$ of T'_1. The state $\langle q'_1, \mathtt{E}\rangle$ is inessential, whereas its equivalence class $[\langle q'_1, \bot\rangle]$ is essential. The most-general unifier of the two states is (cf. Equation 6.5)

$$\mathsf{mgu}(\langle q'_1, \mathtt{E}\rangle, \langle q'_1, \bot\rangle) = \langle \top, \mathtt{E}\rangle \ .$$

Thus, we get the new transition $\mathtt{a} \to [\langle q'_1, \mathtt{E}\rangle](\mathtt{E})$ in T_{min}.

For sake of completeness, the whole minimal deterministic bottom-up tree transducer $T_{min} = (Q_{min}, \Sigma, \Delta, R_{min}, F_{min})$ for the tree function τ_{mgu} is presented in Figure 6.8. ◁

In the following, we show that the equivalent minimal DBTT for a given earliest unified DBTT is unique. Let the deterministic bottom-up tree transducers $T_1 = (Q_1, \Sigma, \Delta, R_1, F_1)$ and $T_2 = (Q_2, \Sigma, \Delta, R_2, F_2)$ be two equivalent minimal DBTTs, i.e., $\tau^{T_1} = \tau^{T_2}$.

For each state $q \in Q_1$, a state $p_q \in Q_2$ is said to be *related* to q if both are reached by at least one same input tree, i.e., $\exists u \in \mathcal{L}^{T_1}(q) \cap \mathcal{L}^{T_2}(p_q)$. Since q is reachable, there exists $u \in \mathcal{L}^{T_1}(q)$ and since q is meaningful, there should also exist a state $p_q \in Q_2$ with $u \in \mathcal{L}^{T_2}(p_q)$. Thus, for each state $q \in Q_1$ exists at least one related state p_q in T_2. We will show that there exists exactly one related state in T_2 for each state $q \in Q_1$. That will give us a mapping from T_1 to T_2.

Lemma 6.8. *Assume T_1 and T_2 are two minimal DBTTs, which are equivalent. Then for each state q of T_1, there exists exactly one related state p_q in T_2 and the following holds:*

- *Every context c of q is a context of p_q and $\tau_q^{T_1}(c) = \tau_{p_q}^{T_2}(c)$.*
- *$\mathcal{L}^{T_1}(q) = \mathcal{L}^{T_2}(p_q)$ and for each input tree u holds*

$$[\![u]\!]^{T_1} = q(s) \quad \textit{iff} \quad [\![u]\!]^{T_2} = p_q(s) \ .$$

Proof. If the states q and p_q are related then their sets of contexts are equal, i.e., $\mathcal{C}^{T_1}(q) = \mathcal{C}^{T_2}(p_q)$, because the transducers define the same transformation.

First assume that q is inessential. Then for every context $c \in \mathcal{C}^{T_1}(q)$, the image is ground, i.e., $\tau_q^{T_1}(c) \in \mathcal{T}_\Delta$. Assume two different states $p_1 \neq p_2$ of T_2 are related to q in T_1. For both $i \in \{1, 2\}$, let tree $u_i \in \mathcal{L}^{T_1}(q) \cap \mathcal{L}^{T_2}(p_i)$ with $[\![u_i]\!]^{T_2} = p_i(s_i)$. Since the transducers T_1 and T_2 are equivalent, for every context $c \in \mathcal{C}^{T_1}(q)$ it holds

$$\tau_q^{T_1}(c) = \tau^{T_1}(c \cdot u_i) = \tau^{T_2}(c \cdot u_i) = \tau_{p_i}^{T_2}(c) \cdot s_i \ .$$

Thus, for all context $c \in \mathcal{C}^{T_1}(q)$ it holds $\tau_{p_1}^{T_2}(c) \cdot s_1 = \tau_{p_2}^{T_2}(c) \cdot s_2$. Consequently, we get for the most-general unifier:

$$\mathsf{mgu}(p_1, p_2) \in \{\langle s_1, s_2\rangle, \langle s_1, \top\rangle, \langle \top, s_2\rangle, \langle \top, \top\rangle, \langle y, y\rangle\}$$

Since T_2 is unified earliest and s_i is an output of p_i, it is not part of the most-general unifier, i.e., $\mathsf{mgu}(p_1, p_2) \notin \{\langle s_1, s_2\rangle, \langle s_1, \top\rangle, \langle \top, s_2\rangle\}$. Moreover, since T_2 is minimal, $\mathsf{mgu}(p_1, p_2) \notin \{\langle \top, \top\rangle, \langle y, y\rangle\}$. – Contradiction.

Now assume that q is essential. Then, there are infinitely many outputs of q. Since T_2 is finite, there should be a related state p in T_2 with infinitely many common input trees for which q produces infinitely many different outputs, i.e., the range $\mathsf{out}^{T_1}(\mathcal{L}^{T_1}(q) \cap \mathcal{L}^{T_2}(p))$ is infinite. Consider a context $c \in \mathcal{C}^{T_1}(q)$ with $\tau_q^{T_1}(c) \in \hat{\mathcal{T}}_\Delta(y)$. We get for every $u \in \mathcal{L}^{T_1}(q) \cap \mathcal{L}^{T_2}(p)$:

$$\tau_q^{T_1}(c) \cdot \mathsf{out}^{T_1}(u) = \tau_p^{T_2}(c) \cdot \mathsf{out}^{T_2}(u) \ .$$

With Proposition 2.1, we know that the image $\tau_q^{T_1}(c)$ in T_1 is a prefix of the image $\tau_p^{T_2}(c)$ in T_2 or vice versa. W.l.o.g., assume that $\tau_q^{T_1}(c) \cdot s = \tau_p^{T_2}(c)$ for some $s \in \hat{\mathcal{T}}_\Delta(y)$. Then for infinitely many input trees u, $\mathsf{out}^{T_1}(u) = s \cdot \mathsf{out}^{T_2}(u)$. Therefore, also for any other context c' of q,

$$\tau_q^{T_1}(c') \cdot s = \tau_p^{T_2}(c') \ .$$

Since T_2 is earliest, the context s must equal y, and therefore, $\tau_q^{T_1}(c) = \tau_p^{T_2}(c)$ for every context c of q. Furthermore, it follows $\mathsf{out}^{T_1}(u) = \mathsf{out}^{T_2}(u)$ for each $u \in \mathcal{L}^{T_1}(q) \cap \mathcal{L}^{T_2}(p)$. Thus, if essential states q and p are related, they produce the same output for common input trees and induce the same image for their contexts.

Now assume there exists an input tree u_1 of q with $[\![u_1]\!]^{T_2} = p_1(s')$ and $p_1 \neq p$. Let $[\![u_1]\!]^{T_1} = q(s)$. If p_1 is essential, there is a related state q_1 of p_1 with $\tau_{q_1}^{T_1}(c) = \tau_{p_1}^{T_2}(c)$ for every context c of p_1. Then we have for every context c of p_1 (and thus also of q and q_1):

$$\tau_{q_1}^{T_1}(c) \cdot s' = \tau_{p_1}^{T_2}(c) \cdot s' = \tau^{T_2}(c \cdot u_1) = \tau^{T_1}(c \cdot u_1) = \tau_q^{T_1}(c) \cdot s$$

If $q_1 \neq q$, then $\langle s', s\rangle$ is a unifier of q_1 and q. The most-general unifier of q_1 and q cannot equal $\langle y, y\rangle$ or $\langle \top, \top\rangle$, since T_1 is minimal. Also, $\langle s', s\rangle$ or $\langle \top, s\rangle$ cannot

equal the most-general unifier, since s is realizable at q. Finally, $\langle s', \top \rangle$ is also no possible most-general unifier of q_1 and q because q is essential. Consequently, $q_1 = q$, and, therefore, also $p = p_1$.

Now assume that p_1 is inessential. In the following, we prove that $p_1 \sim_{T_2} p$ and therefore, $p_1 = p$ since T_2 is minimal. First, we note that for every context $c \in \mathcal{C}^{T_2}(p_1)$ holds

$$\tau^{T_2}_{p_1}(c) = \tau^{T_2}(c \cdot u_1) = \tau^{T_1}(c \cdot u_1) = \tau^{T_1}_{q}(c) \cdot s = \tau^{T_2}_{p}(c) \cdot s \; .$$

For a contradiction assume that $p_1 \neq p$. Then the most-general unifier of the states p_1 and p is $\langle \top, s \rangle$.

Assume that the reason for $p_1 \not\sim_{T_2} p$ is another essential state p_2 of T_2 such that $\mathsf{mgu}(p_1, p_2) = \langle \top, s_2 \rangle$. Then the mgu of the two essential states p and p_2 is given by $\langle s, s_2 \rangle$. Let q_2 be the essential state of T_1, which is related to p_2. Then q_2 and p_2 also have the same sets of contexts where for all $c \in \mathcal{C}^{T_1}(q_2)$, $\tau^{T_1}_{q_2}(c) = \tau^{T_2}_{p_2}(c)$. It follows that $\mathsf{mgu}(q, q_2) = \mathsf{mgu}(p, p_2) = \langle s, s_2 \rangle$. Since s is realizable in q, this is a contradiction.

If $p_1 \not\sim_{T_2} p$ and there is no other essential state p_2 of T_2 with most-general unifier $\mathsf{mgu}(p_1, p_2) = \langle \top, s_2 \rangle$ for some s_2, then there exists a context $c' \in \mathcal{C}_\Sigma$ (not necessarily in $\mathcal{C}^{T_1}(q)$, $\mathcal{C}^{T_2}(p)$, or $\mathcal{C}^{T_2}(p_1)$) together with states q' of T_1 and distinct states p', p'_1, and p'_2 of T_2 together with output trees z and z_1 with the properties:

- $[\![c']\!]^{T_1}_q = q'(z)$, $[\![c']\!]^{T_2}_p = p'(z)$, and $[\![c']\!]^{T_2}_{p_1} = p'_1(z_1)$;
- p'_1 inessential as well, and $\mathsf{mgu}(p', p'_1) = \langle z \cdot s, \top \rangle$;
- there exists another essential state p'_2 of T_2 such that $\mathsf{mgu}(p'_1, p'_2) = \langle \top, s_2 \rangle$.

With the same argument as above, this implies that there exists an essential state q'_2 of T_1 such that $\mathsf{mgu}(q', q'_2) = \langle z \cdot s, s_2 \rangle$ where $z \cdot s$ is realizable at q' — which is a contradiction.

We conclude that the related state of an essential state is also unique (and essential). It remains to prove for inessential states that the related state produces the same output (i.e., is also inessential) and induces the same image for a context. For a contradiction, assume that the related state p_q of an inessential state q is essential. Then p_q again is related to a unique essential state q' that must be different from q — which is not possible. Consequently, the related state p_q of an inessential state q must be inessential as well. Thus, the output for each input tree $u \in \mathcal{L}^{T_1}(q)$ is $*$ in T_1 and T_2, i.e., $[\![u]\!]^{T_1} = q(*)$ and $[\![u]\!]^{T_2} = p_q(*)$. In addition, for each context $c \in \mathcal{C}^{T_1}(q)$, we have for every input tree u of q:

$$\tau^{T_1}_{q}(c) = \tau^{T_1}(c \cdot u) = \tau^{T_2}(c \cdot u) = \tau^{T_2}_{p_q}(c) \qquad \square$$

Subsequently, if we map each state of the transducer T_1 to its related state of the second minimal transducer T_2, we get an isomorphism. The previous Lemma 6.8 yields that minimal transducers are unique.

Theorem 6.9. *The minimal transducer T for a transformation τ is unique.*

Proof. Assume T_1 and T_2 are minimal transducers with $\tau^{T_1} = \tau^{T_2}$. We define a mapping $\varphi : Q_1 \to Q_2$ by $\varphi(q) = p_q$ where p_q is the related state of q. By the previous lemma, this mapping is well-defined and a bijective. In particular, q is the related state of p_q, too. It remains to show that φ is an isomorphism w.r.t. the transition and final functions, i.e.,

1. $F_1(q)$ is defined iff $F_2(\varphi(q))$ is defined, and if they are defined, then they are equal, i.e., $F_1(q) = F_2(\varphi(q))$, and
2. $\mathtt{a}(q_1, \ldots, q_m) \to q_0(z_0) \in R_1 \iff \mathtt{a}(\varphi(q_1), \ldots, \varphi(q_m)) \to \varphi(q_0)(z_0) \in R_2$.

Both follow from Lemma 6.8:

1. Because $\varphi(q)$ is the related state of q and vice versa, every context c is a context of q iff it is a context of p_q, and their images are the same. In particular for $c = y$,
$$F_1(q) = \tau_q^{T_1}(y) = \tau_{\varphi(q)}^{T_2}(y) = F_2(\varphi(q)) \; .$$
2. For each $i \in [m]$, consider an input tree u_i of q_i. If $\mathtt{a}(q_1, \ldots, q_m) \to q_0(z)$ in R_1, then the tree $u_0 = \mathtt{a}(u_1, \ldots, u_m)$ is an input tree of q_0, and also of the related state $\varphi(q_0)$. Therefore, there are states $p_1, \ldots, p_m$ of T_2 such that u_i are input trees of p_i, and there is a transition of the form $\mathtt{a}(p_1, \ldots, p_m) \to \varphi(q_0)(z')$ in R_2. Let $p_0 = \varphi(q_0)$. Since u_i are input trees of p_i (for all $0 \leq i \leq m$), the states q_i and p_i are related and hence, by Lemma 6.8, we get $p_i = \varphi(q_i)$. It remains to show that $z = z'$. By Lemma 6.8, $\mathsf{out}^{T_1}(u_i) = \mathsf{out}^{T_2}(u_i)$ holds for all $i = 0, \ldots, m$. In particular, this means that $z = *$ iff $z' = *$.

   Now assume that z is ground but different from $*$. Then there exists a context c of q_0 (and p_0) such that $s = \tau_{q_0}^{T_1}(c) = \tau_{p_0}^{T_2}(c)$ contains y. If z' contains an occurrence of a variable, then there are two distinct output trees z_1, z_2 at p_0 such that $s \cdot z = s \cdot z_1 = s \cdot z_2$ — which is impossible. Hence, z' must be ground as well and equal to z. It remains to consider the case where z contains occurrences of variables $x_{j_1}, \ldots, x_{j_r}$ and where all states q_{j_i}, p_{j_i} are essential. Then $z = z'$ since for all i, the images for all contexts of q_{j_i} and p_{j_i} must agree. □

Summarizing, we obtain from Propositions 4.1, and 6.2 and Theorems 6.4, 6.6, 6.7, and 6.9:

Theorem 6.10. *For each DBTT T an equivalent minimal transducer can be constructed, which is unique up to renaming of states. If the DBTT T is already proper, the construction can be performed in polynomial time.* □

This proves the first part of the Myhill-Nerode theorem for DBTTs (cf. Page 83), i.e., (1.) implies (2.).

6.2 From Minimal Transducers to Partitions

In Section 5.1, we defined the partition (out^T, r^T) of a DBTT T and we observe that the partition of any trim DBTT T

- is a partition of the tree function τ^T of the transducer (Corollary 5.3),
- is path-finite (Theorem 5.6), and
- its equivalence relation $\equiv_{r^T}$ has finite index (Lemma 5.4).

Now let $T = (Q, \Sigma, \Delta, R, F)$ be a minimal DBTT, which describes the tree function τ. We show that the partition of T is the bottom-up partition of its tree function τ, i.e., $(\mathsf{out}^T, r^T) = (\sigma_\tau, r_\tau)$. This leads to the next part of Theorem 6.1. It proves that (2.) implies (3.).

First, we show that for all trees $u, u' \in \mathrm{dom}(\mathsf{out}^T)$ it holds $u \equiv_{r^T} u'$ iff u and u' reach the same state in T, i.e., $\exists q \in Q$ with $u, u' \in \mathcal{L}^T(q)$. Thereto, we define the equivalence relation $\equiv_T$ on the set of all input trees of states of T, i.e., $\bigcup\{\mathcal{L}^T(q) \mid q \in Q\}$, as follows:

$$u \equiv_T u' \quad \text{if } \exists\, q \in Q :\ u, u' \in \mathcal{L}^T(q)$$

It is the same equivalence relation as $\equiv_{r^T}$:

Lemma 6.11. *Assume $T = (Q, \Sigma, \Delta, R, F)$ be a minimal deterministic bottom-up tree transducer. Then $\equiv_{r^T}$ equals $\equiv_T$.*

Proof. If $u \equiv_T u'$, there is a state q such that $u, u' \in \mathcal{L}^T(q)$. For their residuals, we know $r^T(u) = \tau^T_q = r^T(u')$. Thus, $u \equiv_{r^T} u'$.

Now assume $u \equiv_{r^T} u'$, but $u \not\equiv_T u'$. There are two states $q \neq q'$ with $u \in \mathcal{L}^T(q)$ and $u' \in \mathcal{L}^T(q')$. Since $u \equiv_{r^T} u'$, we get

$$\tau^T_q = r^T(u) = r^T(u') = \tau^T_{q'} .$$

The most-general unifier of q and q' is $\mathsf{mgu}(q, q') \in \{\langle y, y\rangle, \langle \top, \top\rangle\}$. If q and q' are not similar (cf. Page 101), i.e., $q \not\sim q'$, there must be a context c such that $[\![c]\!]^T_q = q_1(z_1)$ and $[\![c]\!]^T_{q'} = q'_1(z'_1)$ with $\mathsf{mgu}(q_1, q'_1) \in \{\langle v_1, \top\rangle, \langle \top, v_2\rangle\}$ for some $v_1, v_2 \in \mathcal{T}_\Delta$. W.l.o.g. assume $\mathsf{mgu}(q_1, q'_1) = \langle v_1, \top\rangle$. For each context $c_1 \in \mathcal{C}^T(q_1)$ of q_1 we know that c_1 is also a context of q'_1. Let $\tau^T_{q_1}(c_1) = z_2$ and $\tau^T_{q'_1}(c_1) = z'_2$. Since $\mathsf{mgu}(q, q') \in \{\langle y, y\rangle, \langle \top, \top\rangle\}$, we get

$$z_2 \cdot z_1 = \tau^T_q(c_1 \cdot c) = \tau^T_{q'}(c_1 \cdot c) = z'_2 \cdot z'_1 .$$

On the other hand, since $\mathsf{mgu}(q_1, q'_1) = \langle v_1, \top\rangle$,

$$z_2 \cdot v_1 = z'_2 .$$

If $v_1 = z_1$, then v_1 is realizable in q_1, which is a contradiction to the assumption that T is unified earliest. Otherwise, Proposition 2.1 provides $v_1 \in \hat{\mathcal{T}}_\Delta(y)$ or $z_2 = z'_2 \in \mathcal{T}_\Delta$. Both contradict the definition of the most-general unifier. It follows: $u \equiv_{r^T} u' \implies u \equiv_T u'$. □

The equivalence relation of the partition is the same as the equivalence relation defined by the states of the DBTT. Now we show that the partition of the minimal DBTT fulfills similar properties as the transducer, i.e., the partition is also trim, proper, earliest, unified earliest, and minimal. Furthermore, the equivalence relation is already a congruence. Consequently, the partition is a bottom-up partition. Since the bottom-up partition of a transformation is unique, it is the unique bottom-up partition of the transformation defined by the transducer.

Theorem 6.12. *Assume T is a minimal DBTT. Then its partition (out^T, r^T) is the bottom-up partition of τ^T.*

Proof. Let $T = (Q, \Sigma, \Delta, R, F)$. By Corollary 5.3, Lemma 5.4, and Theorem 5.6, we know that (out^T, r^T) is a path-finite partition of τ^T with finite index. We will show that the properties trim, proper, earliest, and unified earliest are carried over from T to the partition. We will furthermore prove that $\equiv_T$ is a congruence, which implies that $\equiv_{r^T}$ is a congruence (Lemma 6.11) and the partition is minimal. For each $u \in \mathrm{dom}(\mathsf{out}^T)$ we denote q_u as the state reached by u in T, i.e., $u \in \mathcal{L}^T(q_u)$. Let $u \in \mathrm{dom}(\mathsf{out}^T)$. With Lemma 6.11 we get

$$[u]_{r^T} = \mathcal{L}^T(q_u)\ . \tag{6.6}$$

With that, we prove that (out^T, r^T) is the bottom-up partition of τ^T:

- It is trim: Since T is trim, we get the following equivalence:

$$\begin{aligned} r^T(u)(\mathcal{C}_\Sigma(y)) \subseteq \mathcal{T}_\Delta &\Leftrightarrow\ \tau^T_{q_u}(\mathcal{C}_\Sigma(y)) \subseteq \mathcal{T}_\Delta \\ &\Leftrightarrow\ [\![u]\!]^T = q_u(*) && T \text{ is trim} \\ &\Leftrightarrow\ \mathsf{out}^T(u) = * \end{aligned}$$

- The partition is proper: With Equation 6.6, we get

$$\mathsf{out}^T([u]_{r^T}) = \mathsf{out}^T(\mathcal{L}^T(q_u))\ .$$

  Thus, q_u is essential iff $r^T(u)$ is essential. Since T is proper, it follows that $\mathsf{out}^T([u]_{r^T})$ is either infinite or only contains $*$.

- The partition (out^T, r^T) is earliest: Assume $r^T(u)$ is essential. Then we know that $\mathsf{gcs}(q_u) = \mathsf{gcs}(u)$. Since T is earliest, the greatest common suffix equals y. On the other hand, if $r^T(u)$ is inessential, the greatest common suffix is defined as $\bot$.

- It is unified earliest: Assume there is a tree $u' \in \mathrm{dom}(\mathsf{out}^T)$ such that the most-general unifier of $r^T(u)$ and $r^T(u')$ equals $\langle v, z\rangle$ with v is a realizable ground term. Assume $\mathsf{out}^T(u) = v$. Then $\langle v, z\rangle$ is also the most-general unifier of q_u and $q_{u'}$ (by definition of the most-general unifier of states). Furthermore, $\mathsf{out}^T(u) = v$ and $u \in \mathcal{L}^T(q_u)$ imply, v is realizable in q_u. – Contradiction.

- The relation $\equiv_{r^T}$ is a congruence: It suffices (with Lemma 6.11) to prove that $\equiv_T$ is a congruence relation. If $u \equiv_T u'$ then both trees u and u' reach the same state q_u. Thus, for every context $c \in \mathcal{C}_\Sigma$, the tree $c \cdot u$ in $\mathrm{dom}(\mathsf{out}^T)$ reachs the state $q_{c \cdot u}$ iff the tree $c \cdot u'$ reachs the same state. It follows that $c \cdot u \equiv_T c \cdot u'$.

- Finally, the partition is minimal: Let $u, u' \in \mathrm{dom}(\mathsf{out}^T)$. By definition of the most-general unifier of states, it holds:

$$\mathsf{mgu}(u, u') = \mathsf{mgu}(q_u, q_{u'})$$

It follows that

$$u \sim'_{r^T} u' \quad \text{iff} \quad q_u \sim'_T q_{u'} \,. \tag{6.7}$$

Furthermore, we get for all trees $u_1, u_2 \in \mathrm{dom}(\mathsf{out}^T)$ the following:

$$\begin{aligned} u_1 \equiv_{r^T} u_2 &\Leftrightarrow u_1 \equiv_T u_2 && \text{Lemma 6.11} \\ &\Leftrightarrow q_{u_1} = q_{u_2} && \text{Definition of } \equiv_T \\ &\Leftrightarrow q_{u_1} \sim_T q_{u_2} && \text{Definition of } \sim_T \qquad (6.8) \\ &\Rightarrow q_{u_1} \sim'_T q_{u_2} && \text{Definition of } \sim_T \qquad (6.9) \end{aligned}$$

Now we know that $\equiv_{r^T}$ is a congruence and that it is a refinement of $\sim'_{r^T}$ (Equations 6.7 and 6.9). Assume that $\equiv_{r^T}$ is not the greatest congruence, which is a refinement of $\sim'_{r^T}$. Then there are trees $u, u' \in \mathrm{dom}(\mathsf{out}^T)$ with $u \sim_{r^T} u'$ but $u \not\equiv_{r^T} u'$. Since $\sim_{r^T}$ is a congruence, it holds for every context c with $c \cdot u \in \mathrm{dom}(\mathsf{out}^T)$ that $c \cdot u' \in \mathrm{dom}(\mathsf{out}^T)$ and $c \cdot u \sim_{r^T} c \cdot u'$. Thus, we get for all such contexts $c \cdot u \sim'_T c \cdot u'$ by Equation 6.7. Consequently, $u \sim_T u'$ and with Equation 6.8 also $u \equiv_{r^T} u'$. – Contradiction. We get that $\equiv_{r^T}$ is the greatest congruence $\sim_{r^T}$, which is a refinement of $\sim'_{r^T}$. □

Finally, we get the result that for every transformation, which is definable by a deterministic bottom-up tree transducer, the bottom-up partition exists and is given by the unique minimal DBTT, which can be constructed (given an arbitrary DBTT of the transformation). The following example illustrates this on the tree function τ_{mgu} of Example 5.8.

Example 6.7. Both in Section 5.2.6 and Section 6.1.4 we considered in the examples the tree function τ_{mgu}. In Example 5.12 we defined the unique minimal partition (σ', r') of τ_{mgu}, i.e. the bottom-up partition of τ_{mgu} (Figure 5.7). In addition, the unique minimal DBTT T_{min} of τ_{mgu} (cf. Figure 6.8) is constructed in Example 6.6.

If we deduce the partition $(\mathsf{out}^{T_{min}}, r^{T_{min}})$ of the DBTT, we get exactly the bottom-up partition (σ', r'). For instance, the state $[\langle q_2, \bot \rangle]$ is reached by the tree $\mathtt{d}(\mathtt{e})$. The transitions

$$\mathtt{e} \to [\langle q_1, \mathtt{E} \rangle](*) \qquad \text{and} \qquad \mathtt{d}([\langle q_1, \mathtt{E} \rangle]) \to [\langle q_2, \bot \rangle](\mathtt{E})$$

give us $\mathsf{out}^{T_{min}}(\mathtt{d}(\mathtt{e})) = \mathtt{E}$. Furthermore, since $F_{min}([\langle q_2, \bot\rangle]) = \mathtt{A}(y, \mathtt{E})$, we get that $r^{T_{min}}(\mathtt{d}(\mathtt{e}))(y) = \mathtt{A}(y, \mathtt{E})$. Additionally, there is the transition:

$$\mathtt{g}([\langle q_2, \bot\rangle]) \to [\langle q_3, \bot\rangle](\mathtt{G}(\mathtt{D}, x_1))$$

Since $F_{min}([\langle q_3, \bot\rangle]) = y$, it follows

$$r^{T_{min}}(\mathtt{d}(\mathtt{e}))(\mathtt{g}(y)) = y \cdot \mathtt{G}(\mathtt{D}, y) = \mathtt{G}(\mathtt{D}, y)\ .$$

This corresponds to the residual $r'(\mathtt{d}(\mathtt{e}))$. ◁

6.3 From Partitions to Minimal Transducers

In this section, we prove the last part of the Theorem 6.1. We have already shown that for every DBTT there is a minimal DBTT and that the partition of a minimal DBTT is the bottom-up partition of its tree function, i.e., we know (1.) ⇒ (2.) ⇒ (3.). Since every minimal DBTT is a deterministic bottom-up tree transducer, we also have (2.) ⇒ (1.). It remains to prove that for every bottom-up partition a DBTT exists, which describes the same tree function. Here, we construct such a transducer and, more specific, we will show that it is already the minimal DBTT.

In the following, let (σ_τ, r_τ) be the bottom-up partition of a tree function $\tau : \mathcal{T}_\Sigma \dashrightarrow \mathcal{T}_\Delta$. Our goal is to define a DBTT T_τ, which describes τ. Thereto, we have to build transitions of the form

$$\mathtt{a}(q_1, \ldots, q_k) \to q(z)$$

where z is a tree with variables $x_1, \ldots, x_k$. Assume $\mathtt{a}(u_1, \ldots, u_k)$ is a subtree and this transition would be applied to the root, i.e., for $i = 1, \ldots, k$ the subtree u_i reaches the state q_i. We build the right-hand side by replacing dedicated subtrees of the form $\sigma_\tau(u_i)$ by variables x_i, such that

$$z[\sigma_\tau(u_1), \ldots, \sigma_\tau(u_k)] = \sigma_\tau(\mathtt{a}(u_1, \ldots, u_k))$$

holds. The correct trees $z_i \in \mathcal{T}_\Delta(y)$ such that $z_i[\sigma_\tau(u_i)] = \sigma_\tau(\mathtt{a}(u_1, \ldots, u_k))$ holds, are given by the dedicated images (cf. Definition 5.8):

$$z_i = \mathsf{di}(u_i)(\mathtt{a}(u_1, \ldots, u_{i-1}, y, u_{i+1}, \ldots, u_k))$$

Subsequently, we define z as the greatest common prefix of $z_1, \ldots, z_k$.

Definition 6.5 (Greatest Common Prefix)**.** Let $z_1, \ldots, z_k \in \mathcal{T}_\Sigma(y)$. The *greatest common prefix* of $z_1, \ldots, z_k$ is

$$\mathsf{gcp}(z_1, \ldots, z_k) = \begin{cases} \mathtt{a}(z'_1, \ldots, z'_n) & \forall\, j \le k :\ z_j = \mathtt{a}(z_{j,1}, \ldots, z_{j,n}) \wedge\ \forall\, i \le n : \\ & \qquad z'_i = \mathsf{gcp}(z_{1,i}, \ldots, z_{k,i}) \text{ are defined} \\ x_i & z_i = y \wedge\ \forall\, j, j' \in [k] \setminus \{i\} : \\ & \qquad z_j = z_{j'} \in \mathcal{T}_\Sigma(y) \setminus \{y\} \\ \text{undefined} & \text{otherwise} \end{cases}$$

The greatest common prefix is a tree containing variables $x_1, \ldots, x_k$. Let p be a y-path of z_i, i.e., $p \in \mathtt{Paths}_y(z_i)$. The trees $z_j[p]$, to which p refers in the other trees z_j (for $j \neq i$), are the same if the greatest common prefix is defined. Furthermore, p refers to the variable x_i in the greatest common prefix. Vice versa, if a path p in the greatest common prefix refers to a variable x_i then p is a y-path in the corresponding tree z_i. With the definition of the greatest common prefix, we now define the bottom-up tree transducer of τ using its bottom-up partition (σ_τ, r_τ):

Definition 6.6. The bottom-up tree transducer T_τ is defined by

$$T_\tau = (\mathrm{dom}(\tau)/_{\equiv_{r_\tau}}, \Sigma, \Delta, R_\tau, F_\tau)$$

where the right-hand sides of R_τ are defined as follows:

- For all $\mathtt{b} \in \Sigma^{(0)}$ with $\mathtt{b} \in \mathrm{dom}(\sigma_\tau)$: $R_\tau(\mathtt{b}) = [\mathtt{b}]_{r_\tau}(\sigma_\tau(\mathtt{b}))$.
- For all $k > 0$, $\mathtt{a} \in \Sigma^{(k)}$, $u_1, \ldots, u_k \in \mathrm{dom}(r_\tau)$: If $\mathtt{a}(u_1, \ldots, u_k) \in \mathrm{dom}(\sigma_\tau)$ then

  $$R_\tau(\mathtt{a}, [u_1]_{r_\tau} \ldots [u_k]_{r_\tau}) = [\mathtt{a}(u_1, \ldots, u_k)]_{r_\tau}(\mathsf{gcp}(z_1, \ldots, z_k))$$

  where for all $1 \leq i \leq k$

  $$z_i = \mathsf{di}(u_i)(\mathtt{a}(u_1, \ldots, u_{i-1}, y, u_{i+1}, \ldots, u_k)) \; .$$

The function F_τ is defined for all $u \in \mathrm{dom}(\sigma_\tau)$ with $y \in \mathrm{dom}(r_\tau(u))$ by

$$F_\tau([u]_{r_\tau}) = r_\tau(u)(y) \; .$$

In the following, we prove that this DBTT is well-defined and correct (Theorems 6.19 and 6.21). First, we show that the greatest common prefix of the dedicated images exists. Thereto, consider a tree $\mathtt{a}(u_1, \ldots, u_k) \in \mathrm{dom}(\sigma_\tau)$ and the corresponding dedicated images $z_1, \ldots, z_k$ as defined in the previous definition. The following two lemmas show that y-paths of different trees z_i and z_j are no prefixes of each other. This result is summarized in Corollary 6.15. With this property and the fact that for each $i \in [k]$ holds

$$z_i \cdot \sigma_\tau(u_i) = \sigma_\tau(\mathtt{a}(u_1, \ldots, u_k)) \; ,$$

we get that the greatest common prefix of $z_1, \ldots, z_k$ is defined (Lemma 6.16).

After that, we prove that the choice of the right-hand side is independent of the representatives of the states on the left-hand side (Lemma 6.18). Roughly, this follows because trees are in the same equivalent class if they have the same residual and, subsequently, they have the same dedicated images. Finally, we show that the DBTT describes τ (Theorem 6.21). Additionally, we prove that the transducer T_τ is already the minimal DBTT for this tree function (Theorem 6.23).

First, we show that y-paths in different images of related contexts w.r.t. subtrees of the same node in an input tree of τ, are no prefixes of each other. Note that we use the enhanced definition (cf. Page 20) of the partial order $\sqsupseteq$ on trees in $\mathcal{T}_\Sigma(y)$. For two ground trees $t, t' \in \mathcal{T}_\Sigma$, $t \sqsupseteq t'$ means that t occurs as a subtree in t'.

Lemma 6.13. *Assume $c \cdot \mathtt{a}(u_1, \ldots, u_k) \in dom(\tau)$. For all $i \in [k]$ let*

$$r_\tau(u_i)(c \cdot \mathtt{a}(u_1, \ldots, u_{i-1}, y, u_{i+1}, \ldots, u_k) = z_i \ .$$

Then, it holds for all y-paths $p_{i_1} \in \mathtt{Paths}_y(z_{i_1})$, $p_{i_2} \in \mathtt{Paths}_y(z_{i_2})$, for some $i_1, i_2 \in [k]$ that p_{i_2} is no proper prefix of p_{i_1}.

Proof. Let $i_1, i_2 \in [k]$ and $p_{i_1} \in \mathtt{Paths}_y(z_{i_1})$, $p_{i_2} \in \mathtt{Paths}_y(z_{i_2})$. Assume $i_1 = i_2$. Since y-nodes have rank 0, p_{i_2} is no proper prefix of p_{i_1}. Thus, assume $i_1 \neq i_2$. Let $c_\mathtt{a}$ be the tree $c \cdot \mathtt{a}(u_1, \ldots, u_k)$ where u_{i_1} and u_{i_2} are replaced by x_1 and x_2, respectively, i.e.:

$$c_\mathtt{a} = \begin{cases} c \cdot \mathtt{a}(u_1, \ldots, u_{i_1-1}, x_1, u_{i_1+1}, \ldots, u_{i_2-1}, x_2, u_{i_2+1}, \ldots, u_k) & i_1 < i_2 \\ c \cdot \mathtt{a}(u_1, \ldots, u_{i_2-1}, x_2, u_{i_2+1}, \ldots, u_{i_1-1}, x_1, u_{i_1+1}, \ldots, u_k) & i_1 > i_2 \end{cases}$$

Under the assumption that p_{i_2} is a proper prefix of p_{i_1}, we show that σ_τ is not path-finite. Thereto, we prove that the residual $r_\tau(u_{i_2})$ maps an infinite chain of input contexts $c_2^{(1)}$, $c_2^{(2)}$, ... to images $z_2^{(1)}$, $z_2^{(2)}$, ..., respectively, such that the input contexts have all the same y-path p. But there are y-paths $p^{(i)}$ of $z_2^{(i)}$ such that $p^{(1)}$ is a proper prefix of $p^{(2)}$, which is a proper prefix of $p^{(3)}$ and so on. First, we show that there is a chain of pairs

$$(u_1^{(1)}, u_2^{(1)}), (u_1^{(2)}, u_2^{(2)}), \ldots$$

such that $\sigma_\tau(u_1^{(1)})$ is a proper subtree of $\sigma_\tau(u_2^{(1)})$, which is a proper subtree of $\sigma_\tau(u_1^{(2)})$ and so forth. Furthermore, every second element $u_2^{(i)}$ in this chain is r_τ-equivalent to u_{i_2} and the other elements $u_1^{(i)}$ are r_τ-equivalent to u_{i_1}.

We inductively construct such a chain of pairs $(u_1^{(i)}, u_2^{(i)})$. Thereto, let $P(u_1^{(i)}, u_2^{(i)})$ define the property of $u_1^{(i)}$ and $u_2^{(i)}$ as informally described above. Formally, we define for trees $u_1, u_2 \in \mathrm{dom}(\sigma_\tau)$:

$$\begin{aligned} P(u_1, u_2) = \forall j \in [2] : \ & u_j \equiv_{r_\tau} u_{i_j} && (P_1) \\ \wedge \ & c_\mathtt{a}[u_1, u_2] \in \mathrm{dom}(\tau) && (P_2) \\ \wedge \ & r_\tau(u_1)(c_\mathtt{a}[y, u_2]) = z_1 \in \hat{\mathcal{T}}_\Delta(y) && (P_3) \\ \wedge \ & r_\tau(u_2)(c_\mathtt{a}[u_1, y]) = z_2 \in \hat{\mathcal{T}}_\Delta(y) && (P_4) \\ \wedge \ & \exists v \in \hat{\mathcal{T}}_\Delta(y) \setminus \{y\} : \ \sigma_\tau(u_2) = v \cdot \sigma_\tau(u_1) && (P_5) \\ \wedge \ & \exists p \in \mathtt{Paths}_y(z_2) : \ z_1[p] = v && (P_6) \end{aligned}$$

Assume there is a set $\{(u_1^{(i)}, u_2^{(i)}) \mid i \in \mathbb{N}\}$ such that for all $i \in \mathbb{N}$ holds $P(u_1^{(i)}, u_2^{(i)})$. We denote by $p^{(i)}$ the path $p \in \mathtt{Paths}_y(r_\tau(u_2^{(i)})(c_\mathtt{a}[u_1^{(i)}, y]))$, which

exists according to P_6 for $(u_1^{(i)}, u_2^{(i)})$. In addition, assume that for all $i \in \mathbb{N}$ holds: $p^{(i)}$ is a proper prefix of $p^{(i+1)}$. Then, we get an infinite set of equivalent trees $u_2^{(i)} \equiv_{r_\tau} u_{i_2}$. All these trees have the same residual, hence, for all $i \in \mathbb{N}$ holds $c_{\mathtt{a}}[u_1^{(i)}, y] \in \mathrm{dom}(r_\tau(u_{i_2}))$ (it follows by P_4 for $(u_1^{(i)}, u_2^{(i)})$). The path p_{in} referring to the node labeled x_2 in $c_{\mathtt{a}}$ is a y-path in each input context $c_{\mathtt{a}}[u_1^{(i)}, y]$. However, the set of y-paths in the output (as considered in the definition of path-finiteness) contains all $p^{(i)}$:

$$\{p^{(i)} \mid i \in \mathbb{N}\} \subseteq \bigcup \{\mathtt{Paths}_y(r_\tau(u_2)(c)) \mid c \in \mathrm{dom}(r_\tau(u_2)) \wedge p \in \mathtt{Paths}_y(c)\}$$

Thus, this set is infinite and r_τ is not path-finite. — Contradiction. It follows that p_{i_2} is no proper prefix of p_{i_1}.

Now we prove by induction that such a set $\{(u_1^{(i)}, u_2^{(i)}) \mid i \in \mathbb{N}\}$ exists if p_{i_2} is a proper prefix of p_{i_1}. More precisely, we prove the following:

> If p_{i_2} is a proper prefix of p_{i_1}, then there is a set $\{(u_1^{(i)}, u_2^{(i)}) \mid i \in \mathbb{N}\}$ such that for all $i \in \mathbb{N}$ holds $P(u_1^{(i)}, u_2^{(i)})$ and $p^{(i)}$ is a proper prefix of $p^{(i+1)}$.

Again, we refer by $p^{(i)}$ to the path $p \in \mathtt{Paths}_y(r_\tau(u_2^{(i)})(c_{\mathtt{a}}[u_1^{(i)}, y]))$, which fulfills Property P_6 for the i^{th} pair $(u_1^{(i)}, u_2^{(i)})$. Assume the path p_{i_2} is a proper prefix of path p_{i_1}.

Basis case: First, we show that u_{i_1} and u_{i_2} fulfill $P(u_{i_1}, u_{i_2})$. With $z_j = z_{i_j}$ for both $j \in [2]$ follow immediately P_1 and P_2. The terms z_{i_1}, z_{i_2} are in $\hat{\mathcal{T}}_\Delta(y)$ because p_{i_1} and p_{i_2} are y-paths of them, respectively. P_3 and P_4 follow. If p_{i_2} is a proper prefix of p_{i_1}, then $z_{i_2}[p_{i_2}] = y$, whereas there is a tree $v \in \hat{\mathcal{T}}_\Delta(y)$ different to y with $z_{i_1}[p_{i_2}] = v$. Besides P_6, we also get P_5:

$$\begin{aligned}
\sigma_\tau(u_{i_2}) &= (z_{i_2} \cdot \sigma_\tau(u_{i_2}))[p_{i_2}] \\
&= \tau(c_{\mathtt{a}}[u_{i_1}, u_{i_2}])[p_{i_2}] \\
&= (z_{i_1} \cdot \sigma_\tau(u_{i_1}))[p_{i_2}] \\
&= v \cdot \sigma_\tau(u_{i_1})
\end{aligned}$$

Property P_5 holds with $p = p_{i_2}$. Let u_{i_1} and u_{i_2} be $u_1^{(1)}$ and $u_2^{(1)}$, respectively. In addition, we have $p^{(1)} = p_{i_2}$.

Inductive step: Now assume we have trees $u_1^{(i)}$ and $u_2^{(i)}$, which fulfill the property $P(u_1^{(i)}, u_2^{(i)})$. Let the trees z_j in P_3 and P_4 for $j \in [2]$, v in P_5, and the path p in P_6 of $P(u_1^{(i)}, u_2^{(i)})$ be denoted by $z_j^{(i)}$, $v^{(i)}$, and $p^{(i)}$, respectively. Furthermore, we denote $\tau(c_{\mathtt{a}}[u_1^{(i)}, u_2^{(i)}])$ by $z^{(i)}$. It follows by the definition and the consistency of (σ_τ, r_τ):

$$z_1^{(i)} \cdot \sigma_\tau(u_1^{(i)}) = z^{(i)} = z_2^{(i)} \cdot \sigma_\tau(u_2^{(i)}) \,. \tag{6.10}$$

With Properties P_3 and P_4 it follows that the residuals of $u_1^{(i)}$ and $u_2^{(i)}$ are essential. Since the bottom-up partition is proper, there are infinitely many

trees $\bar{u}_j \equiv_{r_\tau} u_j^{(i)}$ for both $j \in [2]$. Because of this, we get the following equations for each of these pairs $(\bar{u}_1, \bar{u}_2)$:

- $r_\tau(\bar{u}_1)(c_a[y, u_2^{(i)}]) = z_1^{(i)}$ and
- $r_\tau(\bar{u}_2)(c_a[u_1^{(i)}, y]) = z_2^{(i)}$.

Consequently: $\tau(c_a[\bar{u}_1, u_2^{(i)}]) = z_1^{(i)} \cdot \sigma_\tau(\bar{u}_1)$ and $\tau(c_a[u_1^{(i)}, \bar{u}_2]) = z_2^{(i)} \cdot \sigma_\tau(\bar{u}_2)$. Thus, there exist $\bar{z}_1$ and $\bar{z}_2$ with

- $r_\tau(u_1^{(i)})(c_a[y, \bar{u}_2]) = \bar{z}_1$ and
- $r_\tau(u_2^{(i)})(c_a[\bar{u}_1, y]) = \bar{z}_2$.

Note that $\bar{z}_1$ depends on $\bar{u}_2$ and $\bar{z}_2$ on $\bar{u}_1$. Then, we have

$$\bar{z}_1 \cdot \sigma_\tau(u_1^{(i)}) = \tau(c_a[u_1^{(i)}, \bar{u}_2]) = z_2^{(i)} \cdot \sigma_\tau(\bar{u}_2) \tag{6.11}$$

$$z_1^{(i)} \cdot \sigma_\tau(\bar{u}_1) = \tau(c_a[\bar{u}_1, u_2^{(i)}]) = \bar{z}_2 \cdot \sigma_\tau(u_2^{(i)}) \; . \tag{6.12}$$

Because of the equivalence of $u_j^{(i)}$ and $\bar{u}_j$ (for both $j \in [2]$), also

- $r_\tau(\bar{u}_1)(c_a[y, \bar{u}_2]) = \bar{z}_1$ and
- $r_\tau(\bar{u}_2)(c_a[\bar{u}_1, y]) = \bar{z}_2$.

Consequently:

$$\bar{z}_1 \cdot \sigma_\tau(\bar{u}_1) = \tau(c[\bar{u}_1, \bar{u}_2]) = \bar{z}_2 \cdot \sigma_\tau(\bar{u}_2) \; . \tag{6.13}$$

For each pair $(\bar{u}_1, \bar{u}_2)$ hold the Properties P_1 and P_2.

Since the partition is proper, the set of output trees $\{\sigma_\tau(\bar{u}_j) \mid \bar{u}_j \equiv_{r_\tau} u_j^{(i)}\}$ is infinite (for both $j \in [2]$), whereas the number of subtrees of $\sigma_\tau(u_1^{(i)})$ and $\sigma_\tau(u_2^{(i)})$ is finite. Furthermore, also for every tree $\bar{v}_1$, also $v^{(i)} \cdot \bar{v}_1$ and $\bar{v}_1$ have only finitely many subtrees. Thus, there are infinitely many different pairs $(\bar{v}_1, \bar{v}_2)$ satisfying the conditions:

1. $\bar{v}_j = \sigma_\tau(\bar{u}_j)$ for some $\bar{u}_j \equiv_{r_\tau} u_j^{(i)}$ (for both $j \in [2]$)
2. $\bar{v}_2 \neq v^{(i)} \cdot \bar{v}_1$
3. $\bar{v}_j \not\sqsupseteq \sigma_\tau(u_{j'}^{(i)})$ for $j, j' \in [2]$ (and with it $\sigma_\tau(u_{j'}^{(i)}) \neq \bar{v}_j$)
4. $\bar{v}_2 \not\sqsupseteq \bar{v}_1$

Additionally, we restrict the set of pairs $(\bar{v}_1, \bar{v}_2)$, which fulfill these conditions such that the following two conditions hold, too:

5. $r_\tau(u_1^{(i)})(c_{\mathtt{a}}[y, \bar{u}_2]) \in \hat{T}_\Delta(y)$
6. $r_\tau(u_2^{(i)})(c_{\mathtt{a}}[\bar{u}_1, y])[p^{(i)}] \in \hat{T}_\Delta(y)$

To show that such a pair exists, assume there is no tree $\bar{v}_2 = \sigma_\tau(\bar{u}_2)$, which fulfills Conditions 1.-5., i.e., there are at least two different trees $\bar{v}_2 = \sigma_\tau(\bar{u}_2)$ and $\bar{v}'_2 = \sigma_\tau(\bar{u}'_2)$ with

- $\bar{z}_1 = r(u_1^{(i)})(c_{\mathtt{a}}[y, \bar{u}_2]) \in \mathcal{T}_\Delta$ and
- $\bar{z}'_1 = r(u_1^{(i)})(c_{\mathtt{a}}[y, \bar{u}'_2]) \in \mathcal{T}_\Delta$.

With Equations 6.11 and 6.13 for both $\bar{u}_2$ and $\bar{u}'_2$ and an arbitrary $\bar{u}_1$, we get:

$$z_2^{(i)} \cdot \bar{v}_2 = \bar{z}_1 \cdot \sigma_\tau(u_1^{(i)}) = \bar{z}_1 = \bar{z}_1 \cdot \bar{v}_1 = \bar{z}_2 \cdot \bar{v}_2$$
$$z_2^{(i)} \cdot \bar{v}'_2 = \bar{z}'_1 \cdot \sigma_\tau(u_1^{(i)}) = \bar{z}'_1 = \bar{z}'_1 \cdot \bar{v}_1 = \bar{z}_2 \cdot \bar{v}'_2$$

By Proposition 2.1 it follows that it exists $s \in \hat{\mathcal{T}}_\Delta(y)$ with $z_2^{(i)} = \bar{z}_2 \cdot s$ or $z_2^{(i)} \cdot s = \bar{z}_2$ (because $z_2^{(i)} \notin \mathcal{T}_\Delta$ according to Property P_4). It follows (in both cases) that $s \cdot \bar{v}_2 = \bar{v}_2$ and in order that, s equals y and $z_2^{(i)} = \bar{z}_2$. With Equations 6.10 and 6.12, we get:

$$\begin{aligned} z_1^{(i)} \cdot \sigma_\tau(u_1^{(i)}) &= z_2^{(i)} \cdot \sigma_\tau(u_2^{(i)}) && \text{(Equation 6.10)} \\ &= \bar{z}_2 \cdot \sigma_\tau(u_2^{(i)}) \\ &= z_1^{(i)} \cdot \bar{v}_1 && \text{(Equation 6.12)} \end{aligned}$$

Since $\sigma_\tau(u_1^{(i)}) \neq \bar{v}_1$ (Condition 3), $z_1^{(i)}$ must be a ground tree in $\mathcal{T}_\Delta$, which is a contradiction to Property P_3 for $(u_1^{(i)}, u_2^{(i)})$. It follows that there are infinitely many pairs fulfilling Conditions 1.-5.

Now assume there is no tree $\bar{v}_1 = \sigma_\tau(\bar{u}_1)$, which fulfills Conditions 1.-6., i.e., there are at least two different trees $\bar{v}_1 = \sigma_\tau(\bar{u}_1)$ and $\bar{v}'_1 = \sigma_\tau(\bar{u}'_1)$ with

- $\bar{z}_2 = r_\tau(u_2^{(i)})(c_{\mathtt{a}}[\bar{u}_1, y])$ and $\bar{z}_2[p^{(i)}] \in \mathcal{T}_\Delta$ and
- $\bar{z}'_2 = r_\tau(u_2^{(i)})(c_{\mathtt{a}}[\bar{u}'_1, y])$ and $\bar{z}'_2[p^{(i)}] \in \mathcal{T}_\Delta$.

With Equations 6.12 for both $\bar{u}_1$ and $\bar{u}'_1$ and an arbitrary $\bar{u}_2$, we get:

$$\bar{z}_2[p^{(i)}] = (\bar{z}_2 \cdot \sigma_\tau(u_2^{(i)}))[p^{(i)}] = (z_1^{(i)} \cdot \bar{v}_1)[p^{(i)}] = v^{(i)} \cdot \bar{v}_1$$
$$\bar{z}'_2[p^{(i)}] = (\bar{z}'_2 \cdot \sigma_\tau(u_2^{(i)}))[p^{(i)}] = (z_1^{(i)} \cdot \bar{v}'_1)[p^{(i)}] = v^{(i)} \cdot \bar{v}'_1$$

Since $\bar{v}_1 \neq \bar{v}'_1$, it follows that $z_1^{(i)}[p^{(i)}] = v^{(i)}$. But then we get

$$\begin{aligned} \bar{v}_2 &= (z_2^{(i)} \cdot \bar{v}_2)[p^{(i)}] && P_6 \text{ for } (u_1^{(i)}, u_2^{(i)}) \\ &= (\bar{z}_1 \cdot \sigma_\tau(u_1^{(i)}))[p^{(i)}] && \text{Equation 6.11} \\ &= v^{(i)} \cdot \sigma_\tau(u_1^{(i)}) \\ &= \sigma_\tau(u_2^{(i)}) && P_5 \text{ for } (u_1^{(i)}, u_2^{(i)})\ . \end{aligned}$$

This contradicts the third condition, i.e., $\sigma_\tau(u_2^{(i)}) \neq \bar{v}_2$ (Page 115). Thus, there must be a pair $(u_1^{(i+1)}, u_2^{(i+1)})$ such that the Conditions 1.-6. hold for trees $v_j^{(i+1)} = \sigma_\tau(u_j^{(i+1)})$, for $j \in [2]$, with

- $z_1^{(i+1)} = r_\tau(u_1^{(i)})(c_{\mathtt{a}}[y, u_2^{(i+1)}]) \in \hat{\mathcal{T}}_\Delta(y)$ and
- $z_2^{(i+1)} = r_\tau(u_2^{(i)})(c_{\mathtt{a}}[u_1^{(i+1)}, y])$ with $z_2^{(i+1)}[p^{(i)}] \in \hat{\mathcal{T}}_\Delta(y)$.

Consequently, the pair $(u_1^{(i+1)}, u_2^{(i+1)})$ fulfills Properties P_1 - P_4. Now assume that Property P_5 is not true for this pair, i.e., $v_1^{(i+1)} \not\sqsupseteq v_2^{(i+1)}$. Equation 6.12 and Property P_6 for pair $(u_1^{(i)}, u_2^{(i)})$ state

$$v^{(i)} \cdot v_1^{(i+1)} = (z_1^{(i)} \cdot v_1^{(i+1)})[p^{(i)}] = (z_2^{(i+1)} \cdot \sigma_\tau(u_2^{(i)}))[p^{(i)}] \,.$$

Since $v_1^{(i+1)} \not\sqsupseteq \sigma_\tau(u_2^{(i)})$ and $\sigma_\tau(u_2^{(i)}) = v^{(i)} \cdot \sigma_\tau(u_1^{(i)})$, there are the following possibilities:

1. $v_1^{(i+1)} = v \cdot \sigma_\tau(u_2^{(i)})$ and $z_2^{(i+1)}[p^{(i)}] = v^{(i)} \cdot v \in \hat{\mathcal{T}}_\Delta(y)$ or
2. $z_2^{(i+1)}[p^{(i)}] = v^{(i)} \cdot v_1^{(i+1)}$.

Assume the first possibility: Equation 6.13 gives us

$$(z_1^{(i+1)} \cdot v_1^{(i+1)})[p^{(i)}] = (z_2^{(i+1)} \cdot v_2^{(i+1)})[p^{(i)}] = v^{(i)} \cdot v \cdot v_2^{(i+1)} \,.$$

We know that $v_1^{(i+1)}$ is no subtree of $v_2^{(i+1)}$ and vice versa. Furthermore, it holds $v_1^{(i+1)} = v \cdot v^{(i)} \cdot \sigma_\tau(u_1^{(i)})$ and, thus, $v_1^{(i+1)}$ is no subtree of v and $v^{(i)}$. It follows that

$$z_1^{(i+1)}[p^{(i)}] = v^{(i)} \cdot v \cdot v_2^{(i+1)} \,.$$

With Equation 6.11 and P_6 for $(u_1^{(i)}, u_2^{(i)})$, it follows

$$v^{(i)} \cdot v \cdot v_2^{(i+1)} = z_1^{(i+1)} \cdot \sigma_\tau(u_1^{(i)})[p^{(i)}] = (z_2^{(i)} \cdot v_2^{(i+1)})[p^{(i)}] = v_2^{(i+1)} \,.$$

Thus, $v^{(i)} \cdot v = y$, which is a contradiction to Property P_5 for $(u_1^{(i)}, u_2^{(i)})$.

Now assume the second possibility, i.e., $z_2^{(i+1)}[p^{(i)}] = v^{(i)} \cdot v_1^{(i+1)}$: With Equation 6.13, we get

$$(z_1^{(i+1)} \cdot v_1^{(i+1)})[p^{(i)}] = (z_2^{(i+1)} \cdot v_2^{(i+1)})[p^{(i)}] = v^{(i)} \cdot v_1^{(i+1)}$$

and with Equation 6.11 and Property P_6 for $(u_1^{(i)}, u_2^{(i)})$:

$$(z_1^{(i+1)} \cdot \sigma_\tau(u_1^{(i)}))[p^{(i)}] = (z_2^{(i)} \cdot v_2^{(i+1)})[p^{(i)}] = v_2^{(i+1)} \,.$$

Let us consider $z_1^{(i+1)}[p^{(i)}]$:

- If $z_1^{(i+1)}[p^{(i)}] = v^{(i)} \cdot v_1^{(i+1)}$, it equals $v_2^{(i+1)}$. However, this contradicts the assumption that $v_1^{(i+1)}$ is no subtree of $v_2^{(i+1)}$.

- If $z_1^{(i+1)}[p^{(i)}] = v^{(i)}$, we get $v_2^{(i+1)} = v^{(i)} \cdot \sigma_\tau(u_1^{(i)}) = \sigma_\tau(u_2^{(i)})$. However, this contradicts the choice of $v_2^{(i+1)}$.
- If $z_1^{(i+1)}[p^{(i)}] = v'$ for some $v' \neq v^{(i)}$, we get $v' \cdot \sigma_\tau(u_1^{(i)}) = v_2^{(i+1)}$. Then either $v_1^{(i+1)}$ is a subtree of v' and with that $v_1^{(i+1)} \sqsupseteq v_2^{(i+1)}$ (which is a contradiction to the assumption), or it is a subtree of $v^{(i)}$ and with that $v_1^{(i+1)} \sqsupseteq v^{(i)} \cdot v_1^{(i)} = v_2^{(i)}$ (which contradicts Condition 3).

Thus, we get Property P_5 for $(u_1^{(i+1)}, u_2^{(i+1)})$:

$$\exists v^{(i+1)} \in \hat{\mathcal{T}}_\Delta(y) \setminus \{y\} : \; v_2^{(i+1)} = v^{(i+1)} \cdot v_1^{(i+1)} \; .$$

We define $v^{(i+1)}$ such that $v_1^{(i+1)}$ is no subtree of it.

Now we will find the path $p^{(i+1)}$: Thereto, consider the path $p^{(i)}$. We know by Property P_6 for $(u_1^{(i)}, u_2^{(i)})$, that $z_1^{(i)}[p^{(i)}] = v^{(i)}$ and $z_2^{(i)}[p^{(i)}] = y$. Equations 6.11 and 6.12 provide:

$$\begin{aligned}(z_1^{(i+1)} \cdot \sigma_\tau(u_1^{(i)}))[p^{(i)}] &= v_2^{(i+1)} \\ (z_2^{(i+1)} \cdot \sigma_\tau(u_2^{(i)}))[p^{(i)}] &= v^{(i)} \cdot v_1^{(i+1)}\end{aligned}$$

With Condition 6, we know that $p^{(i)}$ is a prefix of a y-path of $z_2^{(i+1)}$. More precisely,

$$z_2^{(i+1)}[p^{(i)}] = v^{(i)} \cdot v' \tag{6.14}$$

with $v' \cdot \sigma_\tau(u_2^{(i)}) = v_1^{(i+1)}$ and $v' \in \hat{\mathcal{T}}_\Delta(y)$. It must be $v^{(i)} \cdot v'$ instead of only a part of $v^{(i)}$. Assume the opposite, i.e., $z_2^{(i+1)}[p^{(i)}] = v'$ with $v^{(i)}$ is no prefix of v'. Then we get with Equation 6.12 that

$$v' \cdot \sigma_\tau(u_2^{(i)}) = v^{(i)} \cdot v_1^{(i+1)} \; .$$

Because $v_1^{(i+1)}$ is no subtree of $\sigma_\tau(u_2^{(i)})$ (Condition 3), there must be a path p such that $v^{(i)}[p] = \sigma_\tau(u_2^{(i)})$. Then Property P_5 for the pair $(u_1^{(i)}, u_2^{(i)})$ implies that $\sigma_\tau(u_2^{(i)})$ is a subtree of itself — Contradiction. Thus, Equation 6.14 is true.

With this property and Equation 6.13,

$$(z_1^{(i+1)} \cdot v_1^{(i+1)})[p^{(i)}] = v^{(i)} \cdot v' \cdot v_2^{(i+1)} = v^{(i)} \cdot v' \cdot v^{(i+1)} \cdot v_1^{(i+1)} \; .$$

There are two possibilities: $z_1^{(i+1)}[p^{(i)}] \in \mathcal{T}_\Delta$ or $z_1^{(i+1)}[p^{(i)}] = v^{(i)} \cdot v' \cdot v^{(i+1)}$. The other cases are not possible: $v_1^{(i+1)}$ is no subtree of $v^{(i)}$ or v' because $v_1^{(i+1)}$ is no subtree of $\sigma_\tau(u_2^{(i)})$ and no proper subtree of itself. In addition, by definition of $v^{(i+1)}$, the tree $v_1^{(i+1)}$ is no subtree of $v^{(i+1)}$.

Assume the first possibility, i.e., $z_1^{(i+1)}[p^{(i)}] = v^{(i)} \cdot v' \cdot v_2^{(i+1)}$. Then we get with Equation 6.11

$$v^{(i)} \cdot v' \cdot v_2^{(i+1)} = (z_1^{(i+1)} \cdot \sigma_\tau(u_1^{(i)}))[p^{(i)}] = v_2^{(i+1)} \; .$$

Since $v^{(i)} \neq y$, this is a contradiction. Thus, the last possibility, must be true. We have:

$$z_1^{(i+1)}[p^{(i)}] = v^{(i)} \cdot v' \cdot v^{(i+1)} \qquad \text{and} \qquad z_2^{(i+1)}[p^{(i)}] = v^{(i)} \cdot v'$$

Let p be a y-path of $v^{(i)} \cdot v'$. Then we define

$$p^{(i+1)} = p^{(i)} \cdot p \; .$$

This path fulfills Property P_6 for $(u_1^{(i+1)}, u_2^{(i+1)})$ and $p^{(i)}$ is a proper prefix of this path $p^{(i+1)}$. □

The y-paths are not only no proper prefixes of each other. They are also different:

Lemma 6.14. *Assume $c \cdot \mathtt{a}(u_1, \ldots, u_k) \in dom(\tau)$. For all $i \in [k]$ let*

$$r_\tau(u_i)(c \cdot \mathtt{a}(u_1, \ldots, u_{i-1}, y, u_{i+1}, \ldots, u_k) = z_i \; .$$

Then, it holds for all y-paths $p_{i_1} \in \mathtt{Paths}_y(z_{i_1})$, $p_{i_2} \in \mathtt{Paths}_y(z_{i_2})$, for some $i_1, i_2 \in [k]$ that $i_1 \neq i_2$ implies $p_{i_2} \neq p_{i_1}$.

Proof. Let $1 \leq i_1 < i_2 \leq k$, $p_{i_1} \in \mathtt{Paths}_y(z_{i_1})$, and $p_{i_2} \in \mathtt{Paths}_y(z_{i_2})$. Under the assumption that p_{i_2} equals p_{i_1}, we show that there is an image $\bar{z}_2$ of a context of u_{i_2} such that there is a y-path p in $\bar{z}_2$ and p_{i_1} is a proper prefix of p. Then we get by Lemma 6.13 a contradiction.

Assume $p_{i_2} = p_{i_1}$. Let $c_\mathtt{a}$ be the tree $c \cdot \mathtt{a}(u_1, \ldots, u_k)$ where u_{i_1} and u_{i_2} are replaced by x_1 and x_2 respectively:

$$c_\mathtt{a} = c \cdot \mathtt{a}(u_1, \ldots, u_{i_1-1}, \boldsymbol{x_1}, u_{i_1+1}, \ldots, u_{i_2-1}, \boldsymbol{x_2}, u_{i_2+1}, \ldots, u_k) \; .$$

Analog to the inductive step of Lemma 6.13, there are infinitely many trees $\bar{u}_j \equiv_{r_\tau} u_{i_j}$ for both $j \in [2]$, such that $\sigma_\tau(\bar{u}_j) \not\supseteq \sigma_\tau(u_{i_{j'}})$ for $j, j' \in \{1, 2\}$, and $\sigma_\tau(\bar{u}_1) \neq \sigma_\tau(\bar{u}_2)$. Furthermore, there exist $\bar{z}_1, \bar{z}_2$ with the following properties:

$$\begin{aligned}
r_\tau(\bar{u}_1)(c_\mathtt{a}[y, u_{i_2}]) &= z_{i_1} = r_\tau(u_{i_1})(c_\mathtt{a}[y, u_{i_2}]) \\
r_\tau(\bar{u}_2)(c_\mathtt{a}[u_{i_1}, y]) &= z_{i_2} = r_\tau(u_{i_2})(c_\mathtt{a}[u_{i_1}, y]) \\
r_\tau(\bar{u}_1)(c_\mathtt{a}[y, \bar{u}_2]) &= \bar{z}_1 = r_\tau(u_{i_1})(c_\mathtt{a}[y, \bar{u}_2]) \\
r_\tau(\bar{u}_2)(c_\mathtt{a}[\bar{u}_1, y]) &= \bar{z}_2 = r_\tau(u_{i_2})(c_\mathtt{a}[\bar{u}_1, y])
\end{aligned}$$

$$z_{i_1} \cdot \sigma_\tau(u_{i_1}) = \tau(c_a[u_{i_1}, u_{i_2}]) = z_{i_2} \cdot \sigma_\tau(u_{i_2}) \tag{6.15}$$

$$\bar{z}_1 \cdot \sigma_\tau(u_{i_1}) = \tau(c_a[u_{i_1}, \bar{u}_2]) = z_{i_2} \cdot \sigma_\tau(\bar{u}_2) \tag{6.16}$$

$$z_{i_1} \cdot \sigma_\tau(\bar{u}_1) = \tau(c_a[\bar{u}_1, u_{i_2}]) = \bar{z}_2 \cdot \sigma_\tau(u_{i_2}) \tag{6.17}$$

$$\bar{z}_1 \cdot \sigma_\tau(\bar{u}_1) = \tau(c[\bar{u}_1, \bar{u}_2]) = \bar{z}_2 \cdot \sigma_\tau(\bar{u}_2) \; . \tag{6.18}$$

Now consider the path p_{i_1}. We know that $z_{i_1}[p_{i_1}] = y = z_{i_2}[p_{i_1}]$. Equation 6.15 provides:

$$\sigma_\tau(u_{i_1}) = (z_{i_1} \cdot \sigma_\tau(u_{i_1}))[p_{i_1}] = (z_{i_2} \cdot \sigma_\tau(u_{i_2}))[p_{i_1}] = \sigma_\tau(u_{i_2})$$

By Equations 6.16 and 6.17, we get

$$(\bar{z}_1 \cdot \sigma_\tau(u_{i_1}))[p_{i_1}] = \sigma_\tau(\bar{u}_2)$$
$$(\bar{z}_2 \cdot \sigma_\tau(u_{i_2}))[p_{i_1}] = \sigma_\tau(\bar{u}_1)$$

Assume that p_{i_1} is no prefix of a y-path of $\bar{z}_2$. Then $\bar{z}_2[p_{i_1}] = \sigma_\tau(\bar{u}_1)$ and (with Equation 6.18)

$$(\bar{z}_1 \cdot \sigma_\tau(\bar{u}_1))[p_{i_1}] = \sigma_\tau(\bar{u}_1) \ .$$

There are two possibilities: $\bar{z}_1[p_{i_1}] \in \mathcal{T}_\Delta$ or $\bar{z}_1[p_{i_1}] = y$. We get

$$\sigma_\tau(\bar{u}_1) = (\bar{z}_1 \cdot \sigma_\tau(u_{i_1}))[p_{i_1}] = \sigma_\tau(\bar{u}_2)$$
$$\text{or} \quad \sigma_\tau(u_{i_1}) = (\bar{z}_1 \cdot \sigma_\tau(u_{i_1}))[p_{i_1}] = \sigma_\tau(\bar{u}_2) \ ,$$

respectively. Both contradict the choice of $\sigma_\tau(\bar{u}_1) \neq \sigma_\tau(\bar{u}_2) \neq \sigma_\tau(u_{i_1})$. Thus, path p_{i_1} must be a prefix of a y-path of $\bar{z}_2$. More precisely,

$$\bar{z}_2[p_{i_1}] = v'$$

with $v' \cdot \sigma_\tau(u_{i_2}) = \sigma_\tau(\bar{u}_1)$ and $v' \in \hat{\mathcal{T}}_\Delta(y) \setminus \{y\}$ (because $\sigma_\tau(\bar{u}_1) \neq \sigma_\tau(u_{i_2})$). Let p' be a y-path of v'. Then $p_{i_1} \cdot p' = p$ is a y-path of $\bar{z}_2$ and we have:

- $c \cdot \mathsf{a}(u'_1, \ldots, u'_k) \in \mathrm{dom}(\tau)$ with $u'_i = u_i$ for $i \neq i_1$ and $u'_{i_1} = \bar{u}_1$.
- for all $i \in [k]$: $r_\tau(u'_i)(c \cdot \mathsf{a}(u'_1, \ldots, u'_{i-1}, y, u'_{i+1}, \ldots, u'_k)) = z'_i$ with $z'_{i_2} = \bar{z}_2$ and $z'_{i_1} = z_{i_1}$.
- paths $p_{i_1} \in \mathtt{Paths}_y(z'_{i_1})$, $p \in \mathtt{Paths}_y(z'_{i_2})$ holds p_{i_1} is a proper prefix of p.

This contradicts Lemma 6.13. □

By these two lemmas, a corresponding property for paths of dedicated images follows immediately:

Corollary 6.15. *Assume* $\mathsf{a}(u_1, \ldots, u_k) \in dom(\sigma_\tau)$. *For all* $i \in [k]$ *let*

$$\mathsf{di}(u_i)(\mathsf{a}(u_1, \ldots, u_{i-1}, y, u_{i+1}, \ldots, u_k) = z_i \ .$$

Then, it holds for all y-paths $p_{i_1} \in \mathtt{Paths}_y(z_{i_1})$, $p_{i_2} \in \mathtt{Paths}_y(z_{i_2})$, *for some* $i_1, i_2 \in [k]$ *that* $i_1 \neq i_2$ *implies* p_{i_2} *is no prefix of* p_{i_1}.

By the following lemma, we get that the greatest common prefixes, as used in Definition 6.6, are defined. In the definition, the trees $\mathsf{gcp}(z_1, \ldots, z_k)$ with dedicated images z_i are part of the right-hand sides. The conditions of the following lemma are fulfilled for these images z_i (with $v_i = \sigma_\tau(u_i)$).

Lemma 6.16. *Assume* $z_1, \ldots, z_k \in \mathcal{T}_\Sigma(y)$ *and there are* $v_1, \ldots, v_k \in \mathcal{T}_\Sigma$, *such that for all* $i, j \in [k]$ *holds*

$$z_i \cdot v_i = z_j \cdot v_j \ .$$

If for all $i, j \in [k]$ *with* $i \neq j$ *and* $p_i \in \mathtt{Paths}_y(z_i)$, $p_j \in \mathtt{Paths}_y(z_j)$ *holds that* p_i *is no prefix of* p_j, *then* $\mathsf{gcp}(z_1, \ldots, z_k)$ *is defined.*

Proof. First, assume $z_1, \dots, z_k \in \mathcal{T}_\Sigma$. Then,

$$z_i = z_i \cdot v_i = z_j \cdot v_j = z_j$$

for all $i, j \in \{1, \dots, k\}$. Consequently, $\mathsf{gcp}(z_1, \dots, z_k) = z_1$ is defined.

Now assume, w.l.o.g., that $z_1 \in \hat{\mathcal{T}}_\Sigma(y)$ and p is a longest y-path of z_1, i.e., for all $i \in [k]$ and all $p' \in \mathtt{Paths}_y(z_i)$ holds $|p| \geq |p'|$. We proof that the greatest common prefix is defined, by induction over the length of p. For $p = \epsilon$, it follows that $z_1 = y$ and $z_2, \dots, z_k \in \mathcal{T}_\Sigma$. Otherwise, p would be a prefix of another y-path. Thus, $z_i = z_i \cdot v_i = z_j \cdot v_j = z_j$ for all $i, j \in \{2, \dots, k\}$. We get $\mathsf{gcp}(z_1, \dots, z_k) = x_1$ is defined.

If $p = (\mathtt{a}, d).p'$ for a path p', $\mathtt{a} \in \Sigma^{(n)}$, and $d \leq n$, then $z_1 = \mathtt{a}(z_{1,1}, \dots, z_{1,n})$ and $p' \in \mathtt{Paths}_y(z_{1,d})$. If $z_i = y$ for some $i \neq 1$, then ϵ is a y-path and a prefix of p, which is a contradiction. Thus, $z_2, \dots, z_k$ do not equal y. Furthermore, for $2 \leq i \leq k$, the equation $z_i \cdot v_i = z_1 \cdot \sigma(u_1)$ implies that $z_i = \mathtt{a}(z_{i,1}, \dots, z_{i,n})$ with $z_{i,j} \cdot v_i = z_{1,j} \cdot \sigma(u_1)$ for $1 \leq j \leq n$. By induction, we get for all $1 \leq j \leq n$ that $z'_j = \mathsf{gcp}(z_{1,j}, \dots, z_{k,j})$ is defined. Consequently, $\mathsf{gcp}(z_1, \dots, z_k)$ is defined and equals $\mathtt{a}(z'_1, \dots, z'_n)$. □

The definition of a right-hand side in Definition 6.6 depends on the representatives of the states in the left-hand side. However, also if we choose different representatives, we get the same right-hand side. To prove that, we first need the following lemma:

Lemma 6.17. *Assume $z_1, \dots, z_k \in \mathcal{T}_\Sigma(y)$ and there are $v_1, \dots, v_k \in \mathcal{T}_\Sigma$, such that for all $i, j \in [k]$ holds*

$$z_i \cdot v_i = z_j \cdot v_j \ .$$

If $z = \mathsf{gcp}(z_1, \dots, z_k)$ is defined, then for all $1 \leq i \leq k$

$$z[v_1, \dots, v_{i-1}, y, v_{i+1}, \dots, v_k] = z_i \ .$$

Proof. W.l.o.g., we proof the equation for $i = 1$, i.e., $z[y, v_2, \dots, v_k] = z_1$, by induction over z: First, assume $z = x_1$. By definition of gcp it follows

$$z_1 = y \qquad \text{and} \qquad \forall\, j \in \{2, \dots, k\} :\ z_j = z_2 \in \mathcal{T}_\Sigma(y) \setminus \{y\}$$

Thus, $z[y, v_2, \dots, v_k] = y = z_1$. Now assume $z = x_i$ for some $1 \neq i \leq k$. By definition of gcp it follows

$$z_i = y \qquad \text{and} \qquad \forall\, j \in \{1, \dots, k\} \setminus \{i\} :\ z_j = z_1 \in \mathcal{T}_\Sigma$$

Otherwise, if $z_1 \in \hat{\mathcal{T}}_\Sigma(y)$, $\epsilon \in \mathtt{Paths}_y(z_i)$ is a prefix of each y-path in z_1. Thus,

$$z[y, v_2, \dots, v_k] = v_i = z_i \cdot v_i = z_1 \cdot v_1 = z_1 \ .$$

Last, assume $z = \mathtt{a}(z'_1, \dots, z'_n)$ for some $\mathtt{a} \in \Sigma^{(n)}$. By definition of gcp it follows

$$\forall\, j \leq k :\ z_j = \mathtt{a}(z_{j,1}, \dots, z_{j,n}) \qquad \text{and} \qquad \forall\, i \leq n :\ z'_i = \mathsf{gcp}(z_{1,i}, \dots, z_{k,i})$$

For all $j \in [k]$ holds $v_{\mathsf{a}} = z_j \cdot v_j$. Since $z_j = \mathsf{a}(z_{j,1}, \ldots, z_{j,n})$, we get $v_{\mathsf{a}} = \mathsf{a}(v_{\mathsf{a},1}, \ldots, v_{\mathsf{a},n})$. In particular, $z_{j,i} \cdot v_i = v_{\mathsf{a},i}$ for all $i \in [n]$. By induction, we get $z'_i[y, v_2, \ldots, v_k] = z_{1,i}$ and this provides:

$$z_1 = \mathsf{a}(z_{1,1}, \ldots, z_{1,n}) = \mathsf{a}(z'_1, \ldots, z'_n)[y, v_2, \ldots, v_k] = z[y, v_2, \ldots, v_k] \qquad \square$$

In particular, the previous lemmas hold for the dedicated images z_i in the Definition 6.6. Thus, we get that the greatest common prefix of trees with different, but r_τ-equivalent trees are the same:

Lemma 6.18. *Assume* $\mathsf{a}(u_1, \ldots, u_k), \mathsf{a}(u'_1, \ldots, u'_k) \in dom(\sigma_\tau)$ *with* $u_i \equiv_{r_\tau} u'_i$ *for all* $i \in [k]$ *and for all* $i \in [k]$ *let*

$$\mathsf{di}(u_i)(\mathsf{a}(u_1, \ldots, u_{i-1}, y, u_{i+1}, \ldots, u_k) = z_i$$
$$\mathsf{di}(u'_i)(\mathsf{a}(u'_1, \ldots, u'_{i-1}, y, u'_{i+1}, \ldots, u'_k)) = z_i \ .$$

Then $\mathsf{gcp}(z_1, \ldots, z_k) = \mathsf{gcp}(z'_1, \ldots, z'_k)$.

Proof. Consider the greatest common prefix $\mathsf{gcp}(z_1, \ldots, z_k)$ with respect to the tree $\mathsf{a}(u_1, \ldots, u_k)$. If we swap only for one index $j \in [k]$ from u_j to u'_j, then all z_i possibly change (without z_j). Thus, for each sequence $\bar{u}_1, \ldots, \bar{u}_k$ where $\bar{u}_i \in \{u_i, u'_i\}$, we get different images $\bar{z}_i$. For a set $I \subseteq [k]$ of indices, we define gcp^I as the greatest common prefix of the sequence $z^I_1, \ldots, z^I_k$ corresponding to the sequence $u^I_1, \ldots, u^I_k$ where $u^I_i = u'_i$ if $i \in I$ and $u^I_i = u_i$ otherwise. We prove that

$$\mathsf{gcp}^I[v^I_1, \ldots, v^I_{j-1}, x_j, \ldots, x_k] = \mathsf{gcp}^{I \cup \{i\}}[v^I_1, \ldots, v^I_{j-1}, x_j, \ldots, x_k]$$

implies

$$\mathsf{gcp}^I[v^I_1, \ldots, v^I_j, x_{j+1}, \ldots, x_k] = \mathsf{gcp}^{I \cup \{i\}}[v^I_1, \ldots, v^I_j, x_{j+1}, \ldots, x_k]$$

for every $i, j \in [k]$.

Let $u_{\mathsf{a}} = \mathsf{a}(u_1, \ldots, u_k)$, $v_{\mathsf{a}} = \sigma_\tau(u_{\mathsf{a}})$, $\sigma_\tau(u_i) = v_i$, and $\sigma_\tau(u'_i) = v'_i$, for all $i \in [k]$. For a set $I \subseteq [k]$, we define:

$$u^I_i = \begin{cases} u_i & i \notin I \\ u'_i & i \in I \end{cases} \qquad\qquad v^I_i = \begin{cases} v_i & i \notin I \\ v'_i & i \in I \end{cases}$$

Furthermore, we get z^I_i with:

$$\begin{aligned} z^I_i &= \mathsf{di}(u_i)(\mathsf{a}(u^I_1, \ldots, u^I_{i-1}, y, u^I_{i+1}, \ldots, u^I_k)) \\ &= \mathsf{di}(u'_i)(\mathsf{a}(u^I_1, \ldots, u^I_{i-1}, y, u^I_{i+1}, \ldots, u^I_k)) \end{aligned}$$

The equality of the two dedicated images follows since $u_i \equiv_{r_\tau} u'_i$. It follows that for a set I and index $i \notin I$ holds:

$$z^I_i = z^{I \cup \{i\}}_i \tag{6.19}$$

We abbreviate greatest common prefixes of $z_1^I, \ldots, z_k^I$ by $\mathsf{gcp}^I = \mathsf{gcp}(z_1^I, \ldots, z_k^I)$. By Lemma 6.16, we know that for all I the greatest common prefix gcp^I is defined. For readability, we write for a set $J \subseteq [k]$: $\mathsf{gcp}^I_J = \mathsf{gcp}^I[\bar{v}_1, \ldots, \bar{v}_k]$ where $\bar{v}_i = x_i$ if $i \in J$, $\bar{v}_i = v_i^I$ otherwise. In particular, $\mathsf{gcp}^I_{[k]} = \mathsf{gcp}^I$.

Thus, we want to prove that

$$\mathsf{gcp}^{\emptyset}_{[k]} = \mathsf{gcp}^{[k]}_{[k]} \,. \tag{6.20}$$

Let $I \subseteq [k]$ and $i \notin I$. By Lemma 6.17, we get $\mathsf{gcp}^I_{\{i\}} = z_i^I$ and $\mathsf{gcp}^{I\cup\{i\}}_{\{i\}} = z_i^{I\cup\{i\}}$. With Equation 6.19 it follows:

$$\mathsf{gcp}^I_{\{i\}} = \mathsf{gcp}^{I\cup\{i\}}_{\{i\}} \tag{6.21}$$

In the end of this proof, we prove the following two implications. Thereto, let $I, J \subseteq [k]$ be arbitrary sets and indexes $i \notin I$ and $j \notin J$ with $i \neq j$. It holds

$$v_i = v_i' \implies \mathsf{gcp}^I_{[k]} = \mathsf{gcp}^{I\cup\{i\}}_{[k]} \tag{6.22}$$

$$\mathsf{gcp}^I_J = \mathsf{gcp}^{I\cup\{i\}}_J \implies \mathsf{gcp}^I_{J\cup\{j\}} = \mathsf{gcp}^{I\cup\{i\}}_{J\cup\{j\}} \,. \tag{6.23}$$

By Implication 6.22 follows, for arbitrary set $I \subseteq [k]$ and index $i \notin I$:

$$v_i = v_i' \implies \mathsf{gcp}^I_J = \mathsf{gcp}^{I\cup\{i\}}_J \text{ for all } J \subseteq [k]$$

By the second implication, it follows immediately, for $I, J \subseteq [k]$ and $i \in J \setminus I$:

$$\mathsf{gcp}^I_J = \mathsf{gcp}^{I\cup\{i\}}_J \implies \mathsf{gcp}^I_{[k]} = \mathsf{gcp}^{I\cup\{i\}}_{[k]} \tag{6.24}$$

Then, we get by Equation 6.21 for every $0 \leq i \leq k-2$ the following:

$$\mathsf{gcp}^{[i]}_{\{i+1\}} = \mathsf{gcp}^{[i+1]}_{\{i+1\}} \qquad \text{and} \qquad \mathsf{gcp}^{[i+1]}_{\{i+2\}} = \mathsf{gcp}^{[i+2]}_{\{i+2\}}$$

In addition, by Implication 6.23 and its conclusive Equation 6.24, we get

$$\mathsf{gcp}^{[i]}_{[k]} = \mathsf{gcp}^{[i+1]}_{[k]} = \mathsf{gcp}^{[i+2]}_{[k]} \,.$$

If $k = 1$, Equation 6.21 already provides: $\mathsf{gcp}^{\emptyset}_{[1]} = \mathsf{gcp}^{[1]}_{[1]}$. This results in Equation 6.20: $\mathsf{gcp}^{\emptyset}_{[k]} = \mathsf{gcp}^{[k]}_{[k]}$.

It remains to prove the two implications. First, we show Implication 6.22, i.e., for $1 \leq i \leq k$ and $I \subseteq [k]$ with $i \notin I$, it holds

$$v_i = v_i' \implies \mathsf{gcp}^I_{[k]} = \mathsf{gcp}^{I\cup\{i\}}_{[k]} \,.$$

Assume $v_i = v_i'$. If for all $j \in [k]$ holds $v_j = v_j'$, then follows $\mathsf{gcp}^I_{[k]} = \mathsf{gcp}^{I\cup\{i\}}_{[k]}$ immediately. Because of the equivalence of u_j and u_j', $z_j^I = z_j^{I\cup\{i\}}$ holds for

all $j \in [k]$. Thus, consider $j \in [k]$ with $v_j \neq v'_j$. Assume $j \notin I$. We get the following equations:

$$\begin{aligned}
\mathsf{gcp}^{I}_{\{i\}}[v_i/x_i] &= \mathsf{gcp}^{I}_{\{j\}}[v_j/x_j] \\
\mathsf{gcp}^{I\cup\{i\}}_{\{i\}}[v'_i/x_i] &= \mathsf{gcp}^{I\cup\{i\}}_{\{j\}}[v_j/x_j] \\
\mathsf{gcp}^{I\cup\{j\}}_{\{i\}}[v_i/x_i] &= \mathsf{gcp}^{I\cup\{j\}}_{\{j\}}[v'_j/x_j] \\
\mathsf{gcp}^{I\cup\{i,j\}}_{\{i\}}[v'_i/x_i] &= \mathsf{gcp}^{I\cup\{i,j\}}_{\{j\}}[v'_j/x_j]
\end{aligned}$$

By Equation 6.21, we could replace $\mathsf{gcp}^{I\cup\{i,j\}}_{\{i\}}$ by $\mathsf{gcp}^{I\cup\{j\}}_{\{i\}}$ and so on. Furthermore, we have the assumption that $v_i = v'_i$. We get

$$\begin{aligned}
\mathsf{gcp}^{I}_{\{j\}}[v_j/x_j] =& \mathsf{gcp}^{I}_{\{i\}}[v_i/x_i] = \mathsf{gcp}^{I\cup\{i\}}_{\{j\}}[v_j/x_j] \\
\mathsf{gcp}^{I}_{\{j\}}[v'_j/x_j] =& \mathsf{gcp}^{I\cup\{j\}}_{\{i\}}[v_i/x_i] = \mathsf{gcp}^{I\cup\{i\}}_{\{j\}}[v'_j/x_j]
\end{aligned}$$

It follows by Proposition 2.1 that $z^{I}_{\{j\}} = \mathsf{gcp}^{I}_{\{j\}} = \mathsf{gcp}^{I\cup\{i\}}_{\{j\}} = z^{I\cup\{i\}}_{\{j\}}$. Now assume $j \in I$. By the same argument for $I' = I \setminus \{j\}$ and Equation 6.21, we get

$$z^{I}_{\{j\}} = \mathsf{gcp}^{I}_{\{j\}} = \mathsf{gcp}^{I'}_{\{j\}} = \mathsf{gcp}^{I'\cup\{i\}}_{\{j\}} = \mathsf{gcp}^{I\cup\{i\}}_{\{j\}} = z^{I\cup\{i\}}_{\{j\}} \ .$$

Since this holds for every $j \in [k]$, we get $\mathsf{gcp}^{I}_{[k]} = \mathsf{gcp}^{I\cup\{i\}}_{[k]}$ (which are the gcp of the same trees).

Last, we prove Implication 6.23, i.e., for arbitrary sets $I, J \subseteq [k]$ and indices $i \notin I$ and $j \notin J$ with $i \neq j$. It holds

$$\mathsf{gcp}^{I}_{J} = \mathsf{gcp}^{I\cup\{i\}}_{J} \implies \mathsf{gcp}^{I}_{J\cup\{j\}} = \mathsf{gcp}^{I\cup\{i\}}_{J\cup\{j\}} \ .$$

First, assume $v_i = v'_i$. By Implication 6.22, it follows:

$$\mathsf{gcp}^{I}_{J} = \mathsf{gcp}^{I\cup\{i\}}_{J} \qquad \text{and} \qquad \mathsf{gcp}^{I}_{J\cup\{j\}} = \mathsf{gcp}^{I\cup\{i\}}_{J\cup\{j\}}$$

Thus, assume $v_i \neq v'_i$. Let $I, J \subseteq [k]$ be arbitrary sets with indices $i \notin I$, $j \notin J$. Assume

$$\mathsf{gcp}^{I}_{J} = \mathsf{gcp}^{I\cup\{i\}}_{J} \qquad \text{and} \qquad \mathsf{gcp}^{I}_{J\cup\{j\}} \neq \mathsf{gcp}^{I\cup\{i\}}_{J\cup\{j\}} \ .$$

It follows

$$\mathsf{gcp}^{I}_{J\cup\{j\}}[v^{I}_{j}/x_j] = \mathsf{gcp}^{I}_{J} = \mathsf{gcp}^{I\cup\{i\}}_{J} = \mathsf{gcp}^{I\cup\{i\}}_{J\cup\{j\}}[v^{I\cup\{i\}}_{j}/x_j] \ .$$

There is a path p_j such that (w.l.o.g.)

$$\mathsf{gcp}^{I}_{J\cup\{j\}}[p_j] = x_j \qquad \text{and} \qquad \mathsf{gcp}^{I\cup\{i\}}_{J\cup\{j\}}[p_j] = v \neq x_j \ .$$

This implies that $z_j^I[p_j] = y$ and $z_j^{I\cup\{i\}}[p_j] = v' \neq y$. W.l.o.g., assume $j \notin I$. We have:

$$z_j^I \cdot v_j = z_i^I \cdot v_i \tag{6.25}$$
$$z_j^{I\cup\{i\}} \cdot v_j = z_i^{I\cup\{i\}} \cdot v_i' \tag{6.26}$$
$$z_j^{I\cup\{j\}} \cdot v_j' = z_i^{I\cup\{j\}} \cdot v_i \tag{6.27}$$
$$z_j^{I\cup\{i,j\}} \cdot v_j' = z_i^{I\cup\{i,j\}} \cdot v_i' \tag{6.28}$$

If $z_j^I[p_j] = y$, then also $z_j^{I\cup\{j\}}[p_j] = y$ (Equation 6.19). With it, we get $\mathsf{gcp}^{I\cup\{j\}}[p_j] = x_j$. By Corollary 6.15, we know that there is no y-path in $z_i^{I\cup\{j\}}$ which is a prefix of p_j or vice versa. Thus, $z_i^{I\cup\{j\}}[p_j] \in \mathcal{T}_\Delta$. It follows:

$$\begin{aligned}
v_j' &= (z_j^{I\cup\{j\}} \cdot v_j')[p_j] = (z_i^{I\cup\{j\}} \cdot v_i)[p_j] && \text{Equation 6.27}\\
&= z_i^{I\cup\{j\}}[p_j] && \text{Lemmas 6.13 and 6.14}\\
&= z_i^{I\cup\{i,j\}}[p_j] && \text{Equation 6.19}\\
&= (z_i^{I\cup\{i,j\}} \cdot v_i')[p_j] && \text{Lemmas 6.13 and 6.14}\\
&= (z_j^{I\cup\{i,j\}} \cdot v_j')[p_j] && \text{Equation 6.28}\\
&= (z_j^{I\cup\{i\}} \cdot v_j')[p_j] && \text{Equation 6.19}\\
&= v' \cdot v_j'
\end{aligned}$$

This is a contradiction if $v' \in \hat{\mathcal{T}}_\Delta(y)$. Otherwise, $v' = v_j'$ and we get:

$$\begin{aligned}
v_j' &= v' = z_j^{I\cup\{i\}}[p_j]\\
&= (z_j^{I\cup\{i\}} \cdot v_j)[p_j] && v' \in \mathcal{T}_\Delta\\
&= (z_i^{I\cup\{i\}} \cdot v_i')[p_j] && \text{Equation 6.26}\\
&= z_i^{I\cup\{i\}}[p_j] && \text{Lemmas 6.13 and 6.14: } z_i^{I\cup\{i\}}[p_j] \in \mathcal{T}_\Delta\\
&= z_i^I[p_j] && \text{Equation 6.19}\\
&= (z_i^I \cdot v_i)[p_j] && \text{Lemmas 6.13 and 6.14}\\
&= (z_j^I \cdot v_j)[p_j] && \text{Equation 6.25}\\
&= v_j
\end{aligned}$$

This contradicts the assumption. Thus, Implication 6.23 holds and with that Equation 6.20: $\mathsf{gcp}_{[k]}^{\emptyset} = \mathsf{gcp}_{[k]}^{[k]}$. □

Summarizing, we get with Corollary 6.15, Lemmas 6.16 and 6.18 that, given a bottom-up partition for a transformation τ, the transducer T_τ of Definition 6.6 is well-defined and correct:

Theorem 6.19. *The transducer T_τ is a well-defined deterministic bottom-up tree transducer.*

Proof. Since $\equiv_{r_\tau}$ has finite index, T_τ has finitely many states. We have to show that the definitions of the transition function and the final function are independent of the choice of representatives of involved states.

The partition (σ_τ, r_τ) is the bottom-up partition of τ. Subsequently, the equivalence relation $\equiv_{r_\tau}$ is a congruence relation. Let $k \geq 0$, $\mathtt{a} \in \Sigma^{(k)}$ and for all $i \in [k]$ let $u_i, u_i' \in \mathrm{dom}(\sigma_\tau)$ with $u_i \equiv_{r_\tau} u_i'$. Thus, we get

$$\begin{aligned}\mathtt{a}(u_1, \ldots, u_k) &\equiv_{r_\tau} \mathtt{a}(u_1', u_2, \ldots, u_k)\\ &\ldots\\ &\equiv_{r_\tau} \mathtt{a}(u_1', \ldots, u_i', u_{i+1} \ldots, u_k)\\ &\ldots\\ &\equiv_{r_\tau} \mathtt{a}(u_1', \ldots, u_k')\end{aligned}$$

It follows that for a transition, the state on the right-hand side is well-defined and independent of the choice of representatives for the states on the left-hand side. By Lemma 6.18, the same holds for the output $\mathsf{gcp}(z_1, \ldots, z_k)$ on the right-hand side.

Similarly, for the final function we get: Let $u, u' \in \mathrm{dom}(\sigma_\tau)$ with $u \equiv_{r_\tau} u'$. Since $r_\tau(u) = r_\tau(u')$, the context y is in $\mathrm{dom}(r_\tau(u))$ iff it is in $\mathrm{dom}(r_\tau(u'))$. Furthermore,

$$F_\tau([u]_{r_\tau}) = r_\tau(u)(y) = r_\tau(u')(y) = F_\tau([u']_{r_\tau}) \,. \qquad \square$$

To prove the correctness of T_τ, i.e., that $\tau^{T_\tau} = \tau$, we first show the following equivalence:

Lemma 6.20. *For every tree $u \in \mathcal{T}_\Sigma$ holds*

$$\sigma_\tau(u) = v \wedge q = [u]_{r_\tau} \iff [\![u]\!]^{T_\tau} = q(v) \,.$$

Proof. Assume $u \in \mathrm{dom}(\sigma_\tau)$ and $c \in \mathrm{dom}(r_\tau(u))$. We prove the equivalence by induction. If $u = \mathtt{b} \in \Sigma^{(0)}$, then there is a transition $\mathtt{b} \to [\mathtt{b}](\sigma_\tau(u))$. This implies $[\![\mathtt{b}]\!]^{T_\tau} = [\mathtt{b}]_{r_\tau}(\sigma_\tau(\mathtt{b}))$. If $u = \mathtt{a}(u_1, \ldots, u_k)$ for $k > 0$. For all $i \in [k]$, $\sigma_\tau(u_i)$ is defined because σ_τ is a subtree function. Then, there is a transition

$$\mathtt{a}([u_1]_{r_\tau}, \ldots, [u_k]_{r_\tau}) \to [u]_{r_\tau}(\mathsf{gcp}(z_1, \ldots, z_k)) \in R_\tau \,.$$

Let $\sigma_\tau(u_i) = v_i$ for $i \in [k]$. We know by induction that for all $i \in [k]$ holds: $[\![u_i]\!]^{T_\tau} = [u_i]_{r_\tau}(v_i)$. By definition, $[\![u]\!]^{T_\tau} = [u]_{r_\tau}(\mathsf{gcp}(z_1, \ldots, z_k)[v_1, \ldots, v_k])$. In addition, Lemma 6.17 directs that

$$\mathsf{gcp}(z_1, \ldots, z_k)[v_1, \ldots, v_k] = v \,.$$

On the other hand, assume $[\![u]\!]^{T_\tau} = q(v)$. If $u = \mathtt{b} \in \Sigma^{(0)}$, then there is a transition $\mathtt{b} \to q(v)$. By construction, this implies that $v = \sigma_\tau(\mathtt{b})$ and $q = [\mathtt{b}]_{r_\tau}$.

If the tree $u = \mathtt{a}(u_1, \ldots, u_k)$ for $k > 0$, then there are states $q_1, \ldots, q_k$ with $[\![u_i]\!]^{T_\tau} = q_i(v_i)$ for all $i \in [k]$ and a transition

$$\mathtt{a}(q_1, \ldots, q_k) \to q(z) \in R_\tau \ .$$

We get the result $[\![u]\!]^{T_\tau} = q(z[v_1, \ldots, v_k])$. By induction, we know the output $\sigma_\tau(u_i) = v_i$ and the state $q_i = [u_i]_{r_\tau}$ for all $i \in [k]$. It follows by the construction of T_τ and determinism that $q = [u]_{r_\tau}$. Since the greatest common prefix is unique, $\mathsf{gcp}(z_1, \ldots, z_k)[v_1, \ldots, v_k] = v$. □

It follows that T_τ defines the desired transformation τ:

Theorem 6.21. *The bottom-up tree transducer T_τ describes τ, i.e., $\tau^{T_\tau} = \tau$.*

Proof. With Lemma 6.20, we get for $u \in \mathcal{T}_\Delta$ the following: Assume $\tau(u) = v'$. Then $\sigma_\tau(u) = v$ and $r_\tau(u)(y) = z$ are defined for some v and some z with $v' = z \cdot v$. We get for the transducer that $[\![u]\!]^{T_\tau} = [u]_{r_\tau}(v)$ (Lemma 6.20) and $F_\tau([u]_{r_\tau}) = r_\tau(u)(y) = z$. Thus, $\tau^{T_\tau}(u) = z \cdot v = \tau(u)$.

Now assume $\tau^{T_\tau}(u) = v'$. Then there is a state $q \in Q$, such that $[\![u]\!]^{T_\tau} = q(v)$ and $F_\tau(q) = z$ and $v' = z \cdot v$. Thus, $\sigma_\tau(u) = v$ and $q = [u]_{r_\tau}$ (Lemma 6.20) and $r_\tau(u)(y) = F_\tau([u]_{r_\tau}) = z$, which implies $\tau(u) = z \cdot v = \tau^{T_\tau}(u)$. □

Finally, we get that the DBTT of a bottom-up partition is already the unique minimal transducer of its tree function. Thereto, we need the following invariant:

Lemma 6.22. *For every tree $u \in \mathcal{T}_\Sigma$ and every context $c \in \mathcal{C}_\Sigma(y)$ holds*

$$c \cdot u \in \mathit{dom}(\sigma_\tau) \implies [\![c]\!]^{T_\tau}_{[u]_{r_\tau}} = [c \cdot u]_{r_\tau}(\mathsf{di}(u)(c)) \ .$$

Proof. Assume $c \cdot u \in \mathrm{dom}(\sigma_\tau)$. We prove the implication by induction: First, assume $c = y$, then by definition, we get

$$[\![y]\!]^{T_\tau}_{[u]_{r_\tau}} = [u]_{r_\tau}(y) = [y \cdot u]_{r_\tau}(\mathsf{di}(u)(y)) \ .$$

Now assume that $c = \mathtt{a}(c_1, \ldots, c_k)$ for some label $\mathtt{a} \in \Sigma^{(k)}$. Since c is a context, there is only one occurrence of y in c. Thus, w.l.o.g. only $c_1 \in \mathcal{C}_\Sigma(y)$ and $c_2, \ldots, c_k \in \mathcal{T}_\Sigma$. The result of c starting at $[u]_{r_\tau}$ is defined by

$$\begin{array}{rcl} [\![c]\!]^{T_\tau}_{[u]_{r_\tau}} & = & q'(z[z_1, \ldots, z_k]) \\ & & \text{with } \forall i\ [\![c_i]\!]^{T_\tau}_{[u]_{r_\tau}} = q_i(z_i) \text{ and } R_\tau(\mathtt{a}, q_1 \ldots q_k) = q'(z) \end{array}$$

In addition, we know:

1. For $i > 1$ holds $[\![c_i]\!]^{T_\tau}_{[u]_{r_\tau}} = [\![c_i]\!]^{T_\tau} = [c_i]_{r_\tau}(\sigma_\tau(c_i))$.
2. By induction follows $[\![c_1]\!]^{T_\tau}_{[u]_{r_\tau}} = [c_1 \cdot u]_{r_\tau}(\mathsf{di}(u)(c_1))$.
3. $R_\tau(\mathtt{a}, [c_1 \cdot u]_{r_\tau}.[c_2]_{r_\tau} \ldots [c_k]_{r_\tau}) = [\mathtt{a}(c_1 \cdot u, c_2, \ldots, c_k)]_{r_\tau}(z) = [c \cdot u]_{r_\tau}(z)$

The first equation follows by Lemma 6.20 and the third is given by the construction of T_τ. The output tree z is the greatest common prefix as given in the definition of T_τ. It holds

$$z[y, \sigma_\tau(c_2), \ldots, \sigma_\tau(c_k)] = \mathsf{di}(c_1 \cdot u)(\mathsf{a}(y, c_2, \ldots, c_k)) \; .$$

This follows by Lemma 6.17 and the fact that

$$\begin{aligned} &\mathsf{di}(c_1 \cdot u)(\mathsf{a}(y, c_2, \ldots, c_k)) \cdot \sigma_\tau(c_1 \cdot u) \\ = \; &\mathsf{di}(c_i)(\mathsf{a}(c_1 \cdot u, c_2, \ldots, c_{i-1}, y, c_{i+1}, \ldots, c_k)) \cdot \sigma_\tau(c_i) \end{aligned}$$

holds for every $i > 2$. For the dedicated image of $c_1 \cdot u$, we get for every context $c' \in \mathrm{dom}(r_\tau(c \cdot u))$:

$$\begin{aligned} &r_\tau(c \cdot u)(c') \cdot \mathsf{di}(c_1 \cdot u)(\mathsf{a}(y, c_2, \ldots, c_k)) \cdot \mathsf{di}(u)(c_1) \\ = \; &r_\tau(c_1 \cdot u)(c' \cdot \mathsf{a}(y, c_2, \ldots, c_k)) \cdot \mathsf{di}(u)(c_1) \\ = \; &r_\tau(u)(c' \cdot c) \end{aligned}$$

Since $\mathsf{di}(u)(c)$ is the unique tree for which this equation holds (Definition 5.8), we get

$$\begin{aligned} \mathsf{di}(u)(c) &= \mathsf{di}(c_1 \cdot u)(\mathsf{a}(y, c_2, \ldots, c_k)) \cdot \mathsf{di}(u)(c_1) \\ &= z[\mathsf{di}(u)(c_1), \sigma_\tau(c_2), \ldots, \sigma_\tau(c_2)] \; , \end{aligned}$$

which is the output of c starting at state $[u]_{r_\tau}$:

$$[\![c]\!]^{T_\tau}_{[u]_{r_\tau}} = [c \cdot u]_{r_\tau}(\mathsf{di}(u)(c)) \qquad \square$$

To prove that the unique minimal bottom-up partition induces the unique minimal DBTT, we show that the transducer T_τ is minimal, i.e., it is trim, proper, earliest, unified earliest, and minimal. These properties are induced by the corresponding properties of the bottom-up partition.

Theorem 6.23. *The transducer T_τ is minimal.*

Proof. The transducer is trim, proper, earliest, unified earliest, and minimal because the partition is trim, proper, earliest, unified earliest, minimal, and its equivalence relation is a congruence:

- Every state is represented by a subtree $u \in \mathrm{dom}(\sigma_\tau)$. Thus, the state is reached by this tree. Both follow by Lemma 6.20. Since the partition is trim, there is a context $c \in \mathrm{dom}(r_\tau(u))$ and with Lemma 6.22, this is also a context of $[u]_{r_\tau}$ such that this state is meaningful. If the output of a state q is useless and there is a transition $\mathsf{a}(q_1, \ldots, q_k) \to q(z)$, then there are trees $u_i \in \mathcal{L}^{T_\tau}(q_i)$ (for all $i \in [k]$) and the residual of $\mathsf{a}(u_1, \ldots, u_k)$ is inessential. Then the dedicated images

  $$\mathsf{di}(u_i)(\mathsf{a}(u_1, \ldots, u_{i-1}, y, u_{i+1}, \ldots, u_k)) = *$$

  and thus, the greatest common prefix of these trees equals $*$, too. It follows that $z = *$ and with that, the transducer is trim.

- Let $q = [u]_{r_\tau}$ be an inessential state. Then $\{[\![u']\!]^{T_\tau} \mid u' \in \mathcal{L}^{T_\tau}(q)\}$ is finite. Thus, $\sigma_\tau([u]_{r_\tau})$ is finite, too. It follows that $r_\tau(u)$ is inessential and since the partition (σ_τ, r_τ) is proper, $\sigma_\tau([u]_{r_\tau}) = \{*\}$. This implies that the inessential state q only produces $*$. — T_τ is proper.

- The greatest common suffix of a state q equals the greatest common suffix of a tree $u \in \mathcal{L}^{T_\tau}(q)$. Because the partition is earliest, the greatest common suffices equal y or $\bot$. For essential states, it contains y, thus, it must equal y. Thus, the transducer is earliest.

- The most-general unifier of two essential states q, q' is defined by the most-general unifier of its context functions $\tau_q^{T_\tau}$ and $\tau_{q'}^{T_\tau}$, respectively. Let $c \in \mathrm{dom}(\tau_q^{T_\tau})$ be an arbitrary context with the result $[\![c]\!]_q^{T_\tau} = \bar{q}(z)$. Furthermore, let $F_\tau(\bar{q}) = z'$. By Lemma 6.22, we know that $\bar{q} = [c \cdot u]_{r_\tau}$ for some tree $u \in \mathcal{L}^{T_\tau}(q)$. Then, we get by Lemma 6.22 and the definition of the dedicated image:

$$\tau_q^{T_\tau}(c) = z' \cdot z = r_\tau(c \cdot u)(y) \cdot \mathsf{di}(u)(c) = r_\tau(u)(c)$$

  Analogously, we get for a tree $u' \in \mathcal{L}^{T_\tau}(q')$ that $\tau_{q'}^{T_\tau}$ equals the residual of u', i.e., $\tau_{q'}^{T_\tau} = r_\tau(u')$. We get:

$$\mathsf{mgu}(q, q') = \mathsf{mgu}(r_\tau(u), r_\tau(u'))$$

  Assume there is a ground term v in such a most-general unifier, which is realizable in q. That means there is an input tree $\bar{u} \in \mathcal{L}^{T_\tau}(q)$ such that $\mathsf{out}(\bar{u}) = v$. Then $\sigma_\tau(\bar{u}) = v$ and $\bar{u} \equiv_{r_\tau} u$ (both follow by Lemma 6.22). Hence v is realizable in $r_\tau(u)$, which contradicts that the partition is unified earliest. Subsequently, T_τ is unified earliest.

- Finally, we show that the transducer is minimal. Thereto, assume T_τ is not minimal. Then there are distinct states q_1, q_2 with $q_1 \sim_{T_\tau} q_2$. Let $u_1, u_2 \in \mathrm{dom}(\sigma_\tau)$ with $[u_i]_{r_\tau} = q_i$ for $i \in [2]$. For every context $c \in \mathcal{C}_\Sigma(y)$ follows that $[c \cdot u_1]_{r_\tau} \sim_{T_\tau} [c \cdot u_2]_{r_\tau}$ if the states exist. We know that the most-general unifier of two states equals the most-general unifier of the residuals of any of their representative subtrees, i.e.,

$$\mathsf{mgu}(q, q') = \mathsf{mgu}(r_\tau(u), r_\tau(u')) \text{ if } u \in \mathcal{L}^{T_\tau}(q),\ u' \in \mathcal{L}^{T_\tau}(q')\ .$$

  Thus, it follows that for all contexts $c \in \mathcal{C}_\Sigma(y)$ holds

$$c \cdot u_1 \sim'_{r_\tau} c \cdot u_2\ .$$

  Since $\sim_{r_\tau}$ is the greatest congruence, which is a refinement of $\sim'_{r_\tau}$, we get $u_1 \sim_{r_\tau} u_2$. Because (σ_τ, r_τ) is minimal, u_1 and u_2 are r_τ-equivalent. It follows that $q_1 = [u_1]_{r_\tau} = [u_2]_{r_\tau} = q_2$. — Contradiction.

Thus, the DBTT T_τ of the bottom-up partition (σ_τ, r_τ) is the unique minimal transducer of τ. □

Summarizing, we proved the last part of the Myhill-Nerode Theorem 6.1. More precise, Theorems 6.19 and 6.21 give us that for every bottom-up partition a deterministic bottom-up tree transducer exists, which describes the same tree function. In addition, Theorem 6.23 states that this transducer is already the unique minimal DBTT.

Chapter 7

Conclusion

In this part, we have shown two main results for deterministic bottom-up finite-state tree-to-tree transducers:

1. We provided a Myhill-Nerode style theorem, i.e., we gave a necessary and sufficient condition for a tree function to be definable by a DBTT.
2. We gave a construction to get a unique DBTT, which can be performed in polynomial time for a large class of transducers.

For the Myhill-Nerode style theorem, tree functions were semantically characterized by partitions, which, in addition, provide an equivalence relation on subtrees of the domain of the tree function. It was shown that a tree function is definable by a DBTT if both the equivalence relation has finite index and the partition is path-finite. The latter guarantees that every path occurring in input trees yields only to finitely many different paths in corresponding output trees. We introduced a normal form for path-finite partitions with equivalence relations of finite index. This bottom-up partition is unique. Step by step, the original partition is improved to guarantee different properties. In particular, the equivalence relation of the bottom-up partition is a congruence. Additionally, the corresponding tree transducer with equivalence classes as states is defined. It is shown that the bottom-up partition leads to the unique DBTT of the same tree function.

The second contribution is the construction of this unique transducer, given an arbitrary deterministic bottom-up tree transducer. In case that the DBTT is already proper, i.e., does only produce output at essential states, the construction can be performed in polynomial time — given that we represent right-hand sides compactly. Though similar in spirit as the corresponding construction for top-down deterministic transducers, the given construction for bottom-up transducers is amazingly involved and relies on a long sequence of transformations of the original DBTT to rule out anomalies in the behavior of the transducer. In a last step, the DBTT was minimized.

It is well known that equivalence for deterministic bottom-up transducers is decidable [Zac80]. Now our results provide an polynomial algorithm for the

equivalence problem for proper DBTTs. For both transducers the unique form has to be constructed and checked for equivalence.

7.1 Future Work

We plan to evaluate in how far our novel normal-form of DBTTs and the Myhill-Nerode theorem can be applied in the context of learning tree-to-tree transformations. In a Gold-style learning algorithm [Gol78], the minimal DBTT would be learned by a characteristic sample set of input and output trees. It would be interesting, how such an algorithm can be defined using the subtree-equivalence of the bottom-up partition. Presumably, deducing the bottom-up partition and mainly identifying trees with essential residuals will be an intrinsic part of such an algorithm. As another example, the normal form can be used to decide certain (semantic) subclasses of DBTTs; e.g., we can decide whether a given DBTT is equivalent to a relabeling, using the normal form. This provides an alternative proof of [Gaz06], for the deterministic case.

Also, it remains to future work to evaluate how these ideas can be enhanced to other classes of tree transformations, e.g., macro tree transducers, which are introduced by Engelfriet in [Eng80]. In [LMN10], Lemay et al. present a Myhill-Nerode Theorem and a Gold-style learning algorithm for deterministic top-down tree transducers. A common superclass of deterministic bottom-up and top-down tree transducers are deterministic top-down tree transducers with regular look-ahead [Eng77]. Such a transducer first assigns informations to the nodes of the input tree by processing a bottom-up relabeling. After that, it uses these informations executing a top-down tree transducer. Is it possible to combine the two minimization algorithms for bottom-up and top-down transducers to get a minimal top-down transducer with regular look-ahead?

There are a lot of different applications of tree transducers, e.g., natural language processing and XSLT (see introduction). An ambitious goal is to enhance the transformation models (as tree transducer) and the corresponding algorithms (e.g., to minimize or learn the models) to approach more complex transformations, which are considered in natural language processing or which occur in practical XSLT examples. Such approaches are often restricted by complexity bounds.

Part II

Type Checking Tree Walking Transducers

Chapter 8

Introduction

The extensible markup language XML is the current standard format for exchanging structured data. Its widespread use has initiated lots of work to support processing of XML on many different levels: Specialized query languages for XML, such as XQuery, transformation languages like XSLT, and programming language support either in the form of special purpose languages like XDuce, or of binding facilities for mainstream programming languages like JAXB. A central problem in XML processing is the *(static) type checking problem*: Given an input and output type and a transformation f, can we statically check whether all outputs generated by f on valid inputs conform to the output type? Since XML types are intrinsically more complex than the types found in conventional programming languages, the type checking problem for XML poses new challenges on the design of type checking algorithms. The excellent survey of [MS05] gives an overview of the different approaches to XML type checking.

In Chapter 1 we have already said that the type checking problem for XML transformations in its most general setting is undecidable. Instead of the approximative approaches, here, we will analyze *exact XML type checking*, i.e., restricting the types and transformations in such a way that type checking becomes decidable. For the exact setting, types can be considered as regular or *recognizable* tree languages — thus, capturing the expressive strength of virtually all known type formalisms for XML [MLM00]. Exact type checking is possible for a large class of transformations (cf. Chapter 1), but even for top-down transformations it is exponential time complete and for more complex transformations, the problem is non-elementary. In this part, we present algorithms to type check expressive classes of transformations defined by tree walking transducers (2TTs), macro tree walking transducers (2MTTs), and macro forest walking transducers (2MFTs). We show that these algorithms work in polynomial time for large subclasses of transformations.

Our approach is *forward type inference*: Given an input type I and a transformation τ, we statically infer all outputs generated by τ on valid inputs. Then we determine whether the set $\tau(I)$ of generated outputs conforms to a given output type. Thereto, we check the intersection of $\tau(I)$ with the complement of

the output type for emptiness. The input and output types are given by tree automata and the transformation is given by a 2TT, 2MTT, or 2MFT. To infer the set $\tau(I)$ of produced output trees, we restrict the transformation by enforcing an additional run of the transducer on the input tree. This additional run simulates the tree automata on the input type and checks if the input tree is valid. Next, for this transducer and a tree automaton describing the output type, a transducer is constructed, which produces desired output trees only, i.e., trees of the complement of the output type. Thereto, this transducer realizes the same transformation as before, but simultaneously checks whether the produced output is accepted by the output automaton. If this transducer does not produce any output, we get an affirmative answer to the type checking problem. To check the transducer for emptiness, we construct a bottom-up tree automaton, which describes the output produced by the transducer. With regard to the different processing of the input trees (walking vs. bottom-up), this construction is complex. Roughly, we subsume all visits of the walking transducer at one node to get the state of the automaton at this node. Thus, if the transducer visits each node only a bounded number of times, the bottom-up automaton can be constructed in polynomial time.

This part is organized as follows: In **Chapter 9**, we define tree walking transducers and related notions. Additionally, we present a normal form for 2TTs, which offers better complexity bounds for the subsequent algorithms. The rules of a 2TT in normal form have restricted right-hand sides.

The type checking problem is formulated in **Chapter 10**. Our approach of using forward type inference, is presented. The types are given by finite tree automata. The basic definitions and properties for bottom-up and deterministic top-down tree automata are summarized in this chapter.

In **Chapter 11**, we give the algorithm to type check tree walking transducers. First, we intersect the transducer with the output type given by a tree automaton. The resulting transducer produces only output trees not in the output type and apart from that, works as the given transducer. If the transformation of the new transducer is empty, the origin 2TT only produces valid output with respect to the given output type. Thus, it remains to decide if the new transducer realizes the empty transformation. Thereto, we deduce an alternating tree walking automaton (ATWA), which accepts the domain of the 2TT. For this automaton, an equivalent bottom-up tree automaton is constructed. Since emptiness for a bottom-up tree automaton is decidable in linear time, we get in Section 11.2 an exponential time algorithm to decide emptiness for both 2TTs and alternating tree automata. In the next section, we show that this algorithm is a polynomial time algorithm if the transducer is *b-bounded*, i.e., every subtree is only processed in at most b different ways. With that, we show that type checking a *strictly* b-bounded 2TT, i.e., every node in an input tree is only visited b times, is polynomial in the size of the transducer and the sizes of deterministic bottom-up automata describing the input and output types.

A similar approach is applied to type check macro tree walking transducers in **Chapter 12**. First, we give the definition and discuss their semantics. Here, we additionally have to deal with parameters for which we consider call-by-

value evaluation. For complexity reasons, we present two different algorithms to compute the intersection of a 2MTT with a deterministic bottom-up tree automaton and with the complement of a deterministic top-down tree automaton, respectively. Then, we construct again an ATWA defining the domain of the resulting macro tree walking transducer. With the algorithm to decide emptiness for an ATWA of the previous chapter, type checking for 2MTTs is decidable in exponential time. For input-linear macro tree walking transducers, i.e., strictly 1-bounded 2MTTs, it takes polynomial time only (Section 12.3).

Finally, we consider in **Chapter 13** macro forest walking transducers. Here, the output type is given by a finite forest monoid (FFM). We present an algorithm to compute the intersection of a 2MFT with an FFM representing the complement of the output type. Since the construction of the complement FFM, in general, is exponential, we also give an algorithm intersecting output-linear 2MFTs with bottom-up forest automata. A bottom-up forest automaton is a bottom-up tree automaton accepting the binary tree representations of forests. Then, an ATWA (working on enriched encodings of forests) is constructed for every macro forest walking transducer. The notion of strict b-boundedness is also generalized for 2MFTs.

Chapter 9

Tree Walking Transducers

Nondeterministic tree transducers describe transformations τ from trees to sets of trees over a ranked alphabet Σ, i.e., $\tau : \mathcal{T}_\Sigma \to 2^{\mathcal{T}_\Sigma}$. Consider for example a transformation, which translates documents describing company structures as the one in Figure 1.2 into a collection of all employees, which are listed under a new root node labeled `staff`. Besides a `data` element, these new `employee` elements now contain an element `boss` if the employee is the subordinate of someone. For our example document, the transformation produces:

```
<staff>
  <employee>
    <data> <name> Charles Montgomery Burns </name> ... </data>
  </employee>
  <employee>
    <data> <name> Waylon Smithers </name> ... </data>
    <boss> <name> Charles Montgomery Burns </name> ... </boss>
  </employee>
  <employee> ...  </employee> ...
  <employee> ...  </employee> ...
</staff>
```

The corresponding tree is referred as v_B and its binary encoding (cf. Page 14) as $v_B{}'$. The latter is depicted on the right in Figure 9.1. Remember that the input tree of this example is abbreviated by u_B and its encoding by $u_B{}'$, illustrated in Figures 1.2(b) and 2.1, respectively.

A tree walking transducer starts at the root of the input tree. Depending on the label of the current node, the direction and the state, it produces a tree with leaves, which again may contain state calls for nodes of the input tree. These recursively accessed nodes are determined according to the directives specified in the right-hand side of the applied rule: On directive *up*, the father of the current node is processed, on directive $down_i$, the i-th child and on directive *stay* the current node itself. Tree walking transducers can be considered as generalizations of *top-down* tree transducers. While top-down tree transducers

are only allowed to move downward in the input tree, tree walking transducers may also stay at the current node or move upward in the tree.

Example 9.1. Using our representation of forests by binary trees (Page 14), the transformation of our example is realized by a tree walking transducer M_{staff} with the following rules.

$$
\begin{aligned}
1\quad & q_I(\texttt{department}) && \to \texttt{staff}(q(\mathit{down}_1), \texttt{e})\ ,\\
2\quad & q(\texttt{employee}) && \to \texttt{employee}(\texttt{data}(q_{\mathit{data}}(\mathit{down}_1), q_{\mathit{boss}}(\mathit{stay})),\\
& && \qquad\qquad q_{\mathit{sub}}(\mathit{down}_1))\ ,\\
3\quad & q(\texttt{e}) && \to q_{\mathit{up}}(\mathit{up})\ ,
\end{aligned}
$$

together with a state q_{data} for copying the personal data

$$
4\quad q_{\mathit{data}}(\texttt{data}) \to \mathit{copy}(\mathit{down}_1)\ ,
$$

as well as a state q_{boss} to find the boss

$$
\begin{aligned}
5\quad & q_{\mathit{boss}}(\texttt{employee}) && \to q_{\mathit{boss}}(\mathit{up})\ ,\\
6\quad & q_{\mathit{boss}}(\texttt{department}) && \to \texttt{e}\ ,\\
7\quad & q_{\mathit{boss}}(\texttt{subordinates}) && \to \texttt{boss}(q_{\mathit{data}}(\mathit{up}), \texttt{e})\ ,
\end{aligned}
$$

and a state q_{sub}, which processes the subordinates

$$
\begin{aligned}
8\quad & q_{\mathit{sub}}(\texttt{data}) && \to q_{\mathit{sub}}(\mathit{down}_2)\ ,\\
9\quad & q_{\mathit{sub}}(\texttt{subordinates}) && \to q(\mathit{down}_1)\ ,\\
10\quad & q_{\mathit{sub}}(\texttt{e}) && \to q_{\mathit{next}}(\mathit{up})\ .
\end{aligned}
$$

The state q_{next} searches (in dfs-manner) the next employee

$$
\begin{aligned}
11\quad & q_{\mathit{next}}(\texttt{data}) && \to q_{\mathit{next}}(\mathit{up})\ ,\\
12\quad & q_{\mathit{next}}(\texttt{employee}) && \to q(\mathit{down}_2)\ ,
\end{aligned}
$$

together with a state q_{up} for going to the boss if there is no further subordinate

$$
\begin{aligned}
13\quad & q_{\mathit{up}}(\texttt{employee}) && \to q_{\mathit{up}}(\mathit{up})\ ,\\
14\quad & q_{\mathit{up}}(\texttt{subordinates}) && \to q_{\mathit{next}}(\mathit{up})\ ,\\
15\quad & q_{\mathit{up}}(\texttt{department}) && \to \texttt{e}
\end{aligned}
$$

where state *copy* in Line 4 is meant to copy the content of `data` (i.e., the left child in the binary representation). The initial state is q_I, which means that we start with state q_I at the root of the tree. The output trees of this transformation are binary representations of the lists of all members of staff. The root, which is labeled `staff`, has a right child with label `e`. The left child of `staff` has label `employee` whose left child is a `data`-node (with the personal data and the boss) and whose right child is a chain of `employee`-nodes. Figure 9.1 illustrates this transformation for the binary example tree ${u_B}'$ resulting in the tree ${v_B}'$. ◁

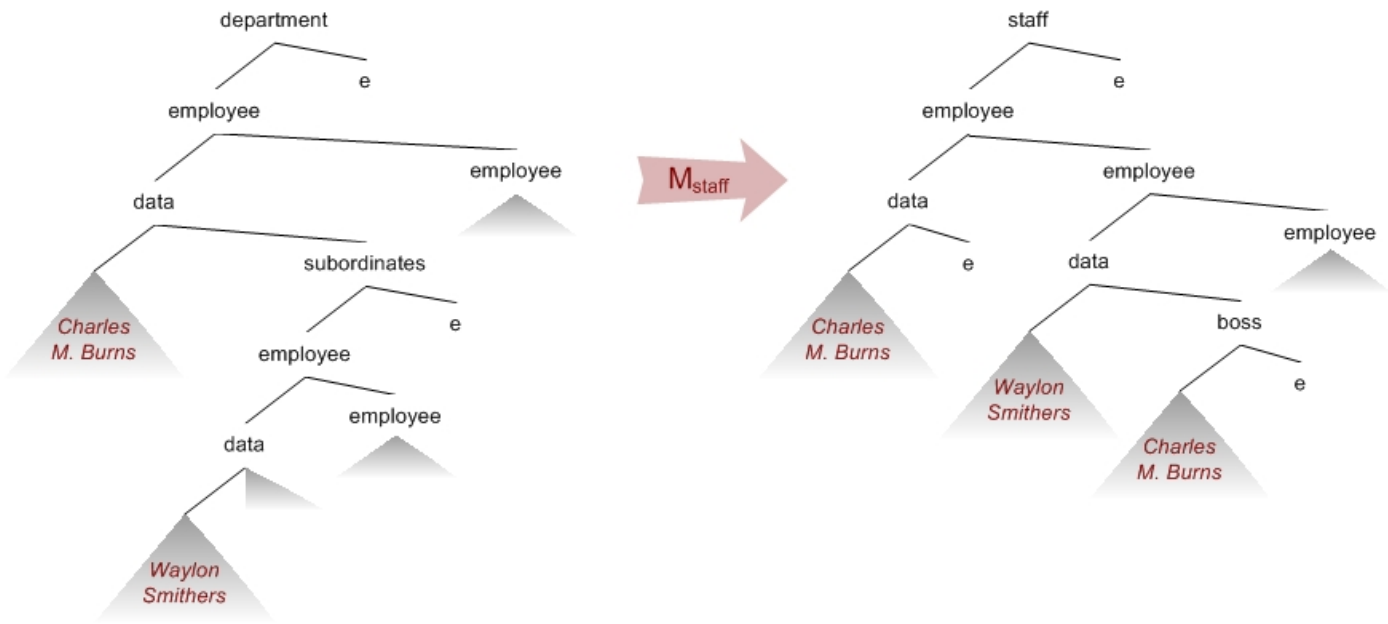


Figure 9.1: The tree $u_B{}'$ and its output tree $v_B{}'$ of the transformation of the 2TT M_{staff}.

The example illustrates that the "first-child next-sibling" encoding of forests implies that the *up*-operation of the tree walking transducer may not necessarily access directly the father in the forest representation but may instead reach the *left* sibling – depending whether or not the current node is a left or right child (i.e., has direction 1 or 2). In the example this was no problem: The state q_{boss} simply proceeds upwards in the tree representation until a node with the right label is reached. A direct construction of forest walking transducers, which provides the operations *up*, *down*, *left*, and *right* will be presented in Chapter 13. For the moment, we restrict ourselves to tree walking transducers on ranked trees (which perhaps are encodings of unranked forests).

Formally, the rules of a tree walking transducer are slightly more general than the ones shown in Example 9.1: Additional to the label of the current node, the left-hand side of a rule also checks the direction of the current node, i.e., whether the current node is the root node (direction is 0), or whether it is the i-th child of its parent node. It is well-known that in the case of tree walking automata (viz. tree walking transducers with output symbols $\{0, 1\}$ of rank 0), such direction tests (or "child number" test) are crucial: Without them, the automaton cannot even realize a depth-first left-to-right traversal over the input tree, i.e., it cannot systematically search through every node of the input. For some translations, however, direction tests are not needed (such as in our Example 9.1). In that example, we must think of every rule as existing in (at most) three incarnations, for direction 0 (root node), direction 1 (left child), and direction 2 (right child). For instance, the q-rule for `employee`-nodes (rule number 2 of the example) is needed in the following two incarnations:

2a $\quad q(\texttt{employee}, 1) \to \texttt{employee}(\texttt{data}(q_{data}(down_1), q_{boss}(stay)), q_{sub}(down_1))$

2b $\quad q(\texttt{employee}, 2) \to \texttt{employee}(\texttt{data}(q_{data}(down_1), q_{boss}(stay)), q_{sub}(down_1))$

Recall from the Preliminaries (Chapter 2) that the maximal rank of symbols in a ranked alphabet Σ is denoted by $mr(\Sigma)$.

Definition 9.1 (2TT)**.** A *tree walking transducer* M (*2TT* for short) is a tuple (Q, Σ, R, Q_0) where Q is a set of states, Σ is a ranked alphabet, $Q_0 \subseteq Q$ is a set of initial states, and R is a finite set of rules. A rule is of the form $q(\mathtt{a}, \eta) \to \zeta$ where $q \in Q$, $\mathtt{a} \in \Sigma^{(m)}$, $m \geq 0$, $\eta \geq 0$ and ζ is a tree generated by the grammar

$$\zeta ::= \mathtt{b}(\underbrace{\zeta, \ldots, \zeta}_{m' \text{ times}}) \mid q'(op) \ ,$$

with $\mathtt{b} \in \Sigma^{(m')}$, $m' \geq 0$, $q' \in Q$, and $op \in \{stay, up\} \cup \{down_i \mid 1 \leq i \leq m\}$.

Tree walking transducers are also called 2-way tree transducers because they generalize to trees the well known concept of 2-way finite state transducer on words (see, e.g., [Gre78]).

Conventionally, tree transducers are defined over two ranked alphabets of input and output symbols. In Definition 9.1 of a 2TT M, we only use one alphabet Σ, which contains input and output symbols. If we want to distinguish

the two, we say that $\mathtt{a} \in \Sigma$ is an *input symbol* if $\mathtt{a}$ appears on the left-hand side of a rule of M; we say that it is an *output symbol* if it appears in the right-hand side of a rule of M. In Example 9.1, `data` is an input and output symbol and `boss` is an output symbol of M_{staff}.

In practice, transducers also have to cope with *unknown* labels in the input such as, e.g., portions of text, which then either are ignored or copied into the output. In order to deal with this, we could simply extend our formalism by an extra symbol $\bullet$ of any given rank which serves as a placeholder for unknown labels of this rank. This idea can be extended to placeholders for unknown elements of different atomic types, for instance `String`, `Number`, or `Date`. Thus, we can describe the so called "Simple Types" of XML Schema (cf. [FW04]).

For a right-hand side ζ, we also write $\zeta = z[q_1(op_1), \ldots, q_c(op_c)]$ to refer to all occurrences of state calls in the right-hand side; there, $z \in \mathcal{T}_\Sigma(X)$ is a tree, which contains exactly one occurrence of the variable x_i for $i = 1, \ldots, c$. Note that z does not contain state calls. For example the right-hand side of the rule in Line 2 in the Example 9.1 can be written as

$$z[q_{data}(down_1), q_{boss}(stay), q_{sub}(down_1)]$$

where $z = \mathtt{employee}(\mathtt{data}(x_1, x_2), x_3)$.

A 2TT is called *deterministic* iff there is at most one state in the set Q_0 and for every triple $(q, \mathtt{a}, \eta)$ of a state, a symbol, and a direction, there is at most one rule with $q(\mathtt{a}, \eta)$ as left-hand side. The example 2TT M_{staff} is deterministic.

Intuitively, the meaning of the expressions of a right-hand side is as follows: The output can either be an element $\mathtt{b}$ whose content is recursively determined, or a recursive call to some state q' on the current input node, on its father or on its i-th subtree. The match patterns in the left-hand side of the rules are restricted to the form "$\mathtt{a}, \eta$", i.e., it is only allowed to check the label of the current input node and its direction. Thus, the transformation of a 2TT M starts at the root node of the input u with one of the initial states. A state q can be applied to an input node ϑ with label $lab(\vartheta) = \mathtt{a}$ and direction $\eta = \eta(\vartheta)$ if there is a rule with left-hand side $q(\mathtt{a}, \eta)$. The evaluation continues on a child ϑi of ϑ for each occurrence of a state call $q'(down_i)$, at ϑ itself for each occurrence of a state call $q'(stay)$, and at the parent of ϑ, for each occurrence of a state call $q'(up)$.

Hence, the *meaning* $[\![q]\!]_u$ of a state q of M with respect to an input tree u can be defined as a function from the nodes (of the input tree) to sets of trees, i.e., $[\![q]\!]_u : \mathtt{Nodes}(u) \to 2^{\mathcal{T}_\Sigma}$. The values $[\![q]\!]_u$ for all q are jointly defined as the *least* functions satisfying:

$$[\![q]\!]_u(\vartheta) \supseteq ([\![\zeta]\!]_u(\vartheta)) \quad \text{for rule} \quad q(\mathtt{a}, \eta) \to \zeta$$

where ϑ is a node of u with $lab(\vartheta) = \mathtt{a}$ and $\eta = \eta(\vartheta)$ with

$$[\![\mathtt{b}(\zeta_1, \ldots, \zeta_m)]\!]_u(\vartheta) = \{\mathtt{b}(u'_1, \ldots, u'_m) \mid u'_i \in [\![\zeta_i]\!]_u(\vartheta)\}$$
$$[\![q'(op)]\!]_u(\vartheta) = [\![q']\!]_u([\![op]\!]_u(\vartheta))$$

where *op* stands for *stay*, *up* or $down_i$ for $1 \leq i \leq rank(\mathtt{a})$, and $[\![op]\!]_u$ is defined by:

$$\begin{aligned} [\![stay]\!]_u(\vartheta) &= \vartheta \\ [\![down_i]\!]_u(\vartheta) &= \vartheta i \\ [\![up]\!]_u(\vartheta i) &= \vartheta \ . \end{aligned}$$

The transformation τ_M realized by the 2TT M on an input tree u and sets U of input trees, respectively, is defined by

$$\tau_M(u) = \bigcup\{[\![q_0]\!]_u(\epsilon) \mid q_0 \in Q_0\} \quad \text{and} \quad \tau_M(U) = \bigcup\{\tau_M(u) \mid u \in U\} \ .$$

For a deterministic tree walking transducer M, the transformation τ_M is a partial function $\tau_M : \mathcal{T}_\Sigma \dashrightarrow \mathcal{T}_\Sigma$. The *domain* of the transducer is the domain of the transformation, i.e., $dom(M) = dom(\tau_M) = \{u \mid \tau_M(u) \neq \emptyset\}$. As usual, the *size* $|M|$ of a 2TT M is the sum of the sizes of all its rules where the size of a rule $q(\mathtt{a}, i) \to \zeta$ is defined as $3 + |\zeta|$. Recall that $|\zeta|$ equals the number of nodes of ζ.

Applying the 2TT M_{staff} from before to ${u_B}'$, we obtain the tree ${v_B}'$. The right-hand sides of rules in a 2TT may be arbitrarily large and contain arbitrarily many state calls. Dealing with such rules increases the complexity of some algorithms on 2TTs. Thus, we give a normal form for 2TTs where the number of state calls in right-hand sides is bounded by the maximal rank of output symbols. In the particular case where we consider binary representations of forests, the number of state calls in right-hand sides can be restricted to 2.

Lemma 9.1. *For every 2TT M, a 2TT M' can be constructed in time $\mathcal{O}(|M|)$ such that*

1. *$\tau_{M'} = \tau_M$ and*
2. *the right-hand side of each rule of M' contains at most k occurrences of states*

where k is the maximal rank of the output symbols of M.

Proof. Let $M = (Q, \Sigma, R, Q_0)$. Intuitively, the idea of the construction is to introduce auxiliary states for all proper subtrees which contain more than one state call. For a symbol $\mathtt{a} \in \Sigma$ and direction η, let $Z_{\mathtt{a},\eta}$ denote the set of all subterms with more than one state call in right-hand sides of rules for $\mathtt{a}, \eta$. For each $\zeta \in Z_{\mathtt{a},\eta}$, we introduce a fresh state $q_{\mathtt{a},\eta,\zeta}$. Assume that $\zeta = \mathtt{b}(\zeta_1, \ldots, \zeta_m)$. Then we introduce the new rule

$$q_{\mathtt{a},\eta,\zeta}(\mathtt{a}, \eta) \to \mathtt{b}(\zeta_1', \ldots, \zeta_m')$$

where $\zeta_j' = \zeta_j$ if ζ_j contains at most one occurrence of a state, and otherwise $\zeta_j' = q_{\mathtt{a},\eta,\zeta_j}(stay)$. We construct $M' = (Q', \Sigma, R', Q_0)$ as follows. The set of rules R' of the new transducer consists of all these newly constructed rules.

Additionally, we add for every rule $q(\mathtt{a},\eta) \to \mathtt{b}(\zeta_1,\ldots,\zeta_m)$ of M a new rule $q(\mathtt{a},\eta) \to \mathtt{b}(\zeta'_1,\ldots,\zeta'_m)$ where for every j, $\zeta'_j = \zeta_j$ if ζ_j contains at most one occurrence of a state, and $\zeta'_j = q_{\mathtt{a},\eta,\zeta_j}(\mathit{stay})$ otherwise. The set of states Q' contains all states of Q and, additionally, the new states $q_{\mathtt{a},\eta,\zeta}$ for every symbol $\mathtt{a} \in \Sigma$, direction η, and every term $\zeta \in Z_{\mathtt{a},\eta}$.

The resulting transducer M' has a new state at most for every non-leaf node of a right-hand side of a rule in M. Thus, in the worst case, we have at most $|M|$ new states. In the new rules, the right-hand side of the original rule of M is split in its subtrees. Thereby, we have $|M'| \in \mathcal{O}(|M|)$. □

In order to describe the behavior of the 2TT $M = (Q, \Sigma, R, Q_0)$ on a fixed input tree u, we are also going to define *runs* of M. A run can itself be described by a ranked tree over the set of rules. Here, the rank of a rule $q(\mathtt{a},\eta) \to \zeta$ is given by the number of occurrences of calls $q'(\mathit{op})$ in ζ to states q' in Q.

Definition 9.2 (Run)**.** Let q denote a state of M and ϑ a node in the input tree u of direction η, which is labeled with $\mathtt{a}$. Assume that $r : q(\mathtt{a},\eta) \to \zeta$ is a rule in R with $\zeta = z[q_1(\mathit{op}_1),\ldots,q_m(\mathit{op}_m)]$. Then the tree

$$\rho = r(\rho_1,\ldots,\rho_m) \in \mathcal{T}_R$$

is a (q,ϑ)-*run* of the 2TT M on the tree u if for every $i \in [m]$, ρ_i is a (q_i,ϑ_i)-run of M on u where ϑ_i is obtained from ϑ by operation op_i. The *output* $\tau(\rho)$ produced by a run ρ is defined by

$$\tau(\rho) = z[\tau(\rho_1),\ldots,\tau(\rho_m)]\,.$$

A (q_0,ϵ)-run for an initial state q_0 is also called *accepting run* of M on u.

If M is deterministic, there exists at most one accepting run on every tree.

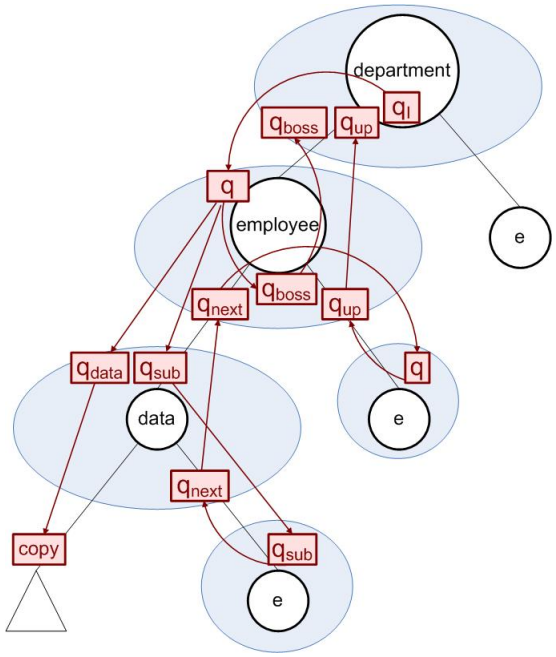
Example 9.2. Figure 9.2(a) shows the behavior of the (deterministic) example 2TT M_{staff} on the tree u_e, which describes a department with one employee:

$$u_e = \mathtt{department}(\mathtt{employee}(\mathtt{data}(\ldots,\mathtt{e}),\mathtt{e}),\mathtt{e})$$

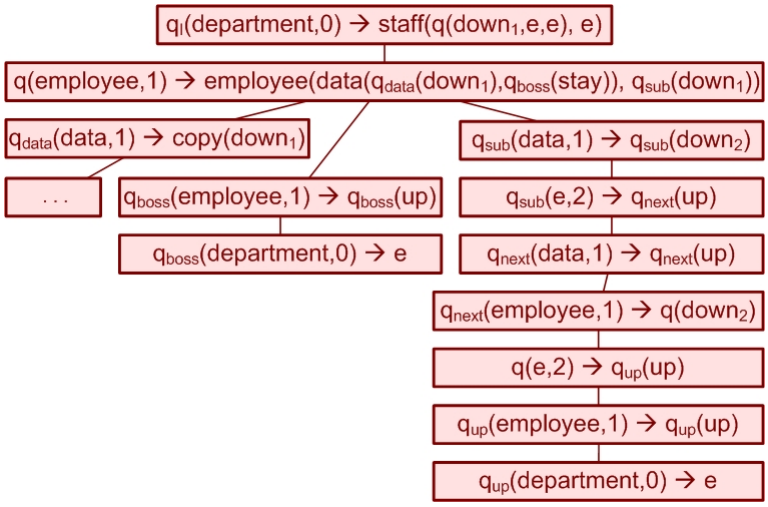
All states in an oval around a node are applied to this node. The picture includes the dependences of the states. For example, consider the employee node $\vartheta = 1$ (with $\mathit{lab}(\vartheta) = \mathtt{employee}$). There, we have the state q. In the 2TT, there is just one rule with left-hand side $q(\mathtt{employee},1)$:

$$_{2\mathrm{a}}\quad q(\mathtt{employee},1) \to \mathtt{employee}(\mathtt{data}(q_{\mathit{data}}(\mathit{down}_1), q_{\mathit{boss}}(\mathit{stay})), q_{\mathit{sub}}(\mathit{down}_1))$$

Thus, we have a $(q,1)$-run $\rho = r_{2a}(\rho_1,\rho_2,\rho_3)$ where ρ_1,ρ_3 are $(q_{\mathit{data}},1.1)$- and $(q_{\mathit{sub}},1.1)$-runs, respectively, and ρ_2 is a $(q_{\mathit{boss}},1)$-run. This is illustrated by the three arrows starting at q at node ϑ. The Figure 9.2(b) shows a (q_I,ϵ)-run $\rho' = r_1(\rho)$. The state *copy* was not detailed in Example 9.1. Accordingly, the $(\mathit{copy},1.1.1)$-run here is not complete. The output $\tau(\rho)$ of this run is the tree $\mathtt{staff}(\mathtt{employee}(\mathtt{data}(\ldots,\mathtt{e}),\mathtt{e}),\mathtt{e})$. ◁



(a) The behavior of 2TT M_{staff} on tree u_e.



(b) The accepting run of M_{staff} on the tree u_e.

Figure 9.2: The 2TT M_{staff} on the tree u_e.

Another approach to define the semantics of a 2TT is via accepting runs. Indeed, this operational semantics of a 2TT coincides with the denotational semantics provided first.

Theorem 9.2. *For a tree u and a 2TT M, the following two statements are equivalent.*

1. *There is an accepting run ρ of M for u with $\tau(\rho) = v$*
2. *$v \in \tau_M(u)$.*

Theorem 9.2 can be proved by fixpoint induction. The denotational view on the semantics of a 2TT allows us to use fixpoint arguments for proving the correctness of constructions, whereas the operational view is better suited for combinatorial arguments.

9.1 Notes and References

Top-down tree transducers were invented by Rounds and Thatcher [Rou70, Tha69]. Top-down tree transducers terminate for every input tree because they process the input tree strictly top-down. While the height increase of a top-down tree transducer is at most linear, the size increase is at most exponential (viz. the translation of a monadic tree with n nodes into a full binary tree of height n). A nondeterministic top-down tree transducer can associate at most double exponentially many output trees to a given input tree; e.g., the transducer with the five rules $q(\mathtt{a}, \eta) \to \mathtt{b}(q(\mathit{down}_1), q(\mathit{down}_1))$, $q(\mathtt{a}, \eta) \to \mathtt{c}(q(\mathit{down}_1), q(\mathit{down}_1))$ for $\eta \in \{0, 1\}$, and $q(\mathtt{e}, 1) \to \mathtt{e}$. Tree walking transducers with output strings were invented in [AU71]; by adding the ability to generate output trees rather than strings, we obtain the tree walking transducer of this thesis. It can be seen as the k-pebble tree transducer of [MSV03], for the case that $k = 0$. In [KS81], it was shown that tree walking transducers without child number test are not useful: They cannot even check whether all leaves of input trees are labeled by some symbol $\mathtt{a}$. As mentioned in [EM03b], in the total deterministic case the tree walking transducer is essentially the same as the attribute grammar [Knu68]. Similar to the fact that circularity of attribute grammars is decidable, it is possible to change any deterministic tree walking transducer in such a way that all runs are terminating [EM03b]. This is not possible for nondeterministic tree walking transducers because they can associate infinitely many output trees to a given input tree (viz. the transducer with the two rules $q(\mathtt{a}, 0) \to \mathtt{b}(q(\mathit{stay}))$ and $q(\mathtt{a}, 0) \to \mathtt{e}$). The normal form of Lemma 9.1 is similar to the one for pebble macro tree transducers given in Theorem 16 of [EM03b]. Attribute grammars with tree output are also called "attributed tree transducers" [Fül81]; for total deterministic such transducers (which coincide with our tree walking transducers when they are total deterministic) it is known that the size-to-height relationship of input tree to output tree is linear, and that the number of different output subtrees in an output tree is linear in the size of the corresponding input tree (see, e.g., [FV98]).

Chapter 10

Type Checking

In this chapter, we present general techniques for certifying that all outputs produced by a transducer M for trees of a given input type are *well-formed*, i.e., comply with some given output type O. This problem is called *type checking* of the transducer M. Here, a type is just a set of trees, i.e., a *tree language*. Clearly, the tractability of type checking heavily depends on the class of languages used as types, and the class of transformations.

The binary tree $v_B{}'$ is an example for the output language of the 2TT M_{staff} (cf. Figure 9.1). Such output trees are binary trees with a root labeled with `staff` and a right-comb of `employee` nodes as left subtree. It is the first-child next-sibling representation of a tree, which has a root labeled with `staff` and arbitrary many `employee` nodes as children. The following DTD (Document Type Definition, [BPS08]) is describing this type (not the binary representation) where `content` stands for further personal data which are not specified here:

```
<!ELEMENT staff (employee)*>
<!ELEMENT employee (data, boss)>
<!ELEMENT data (name, content)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT boss (name, content)>
<!ELEMENT content ... >
```

10.1 Type Checking by Forward Type Inference

Type checking a transducer M means to verify that all trees produced by M for input trees in the given input type I are necessarily contained in the given output type O. If τ is the transformation induced by the transducer M, we want to check whether or not

$$\tau(I) \subseteq O$$

where $\tau(I) = \{\tau(u) \mid u \in I\}$. If this check succeeds, then we say that M *type checks w.r.t.* I *and* O. We solve this problem by forward inference, i.e., we

determine whether

$$\tau(I) \cap \overline{O} = \emptyset$$

where $\overline{O}$ is the complement of the type O. In order to decide emptiness of this intersection, we proceed in two steps. First, we construct from transducer M a transducer $M_{\overline{O}}$ which produces only those outputs of M, which are from $\overline{O}$. For 2TTs, this construction is presented in Section 11.1. Second, we present methods for deciding emptiness of transducers (w.r.t. I).

10.2 Tree Automata

There are several specification formalisms for XML types, such as DTD, XML Schema, or RELAX NG. For our purpose, the particular type formalisms is not essential, as all of these formalisms can be abstracted by recognizable (or regular) tree languages. Thus, each type definition can be translated into a finite tree automaton. XML Schema specifications, e.g., can be considered as simple classes of deterministic top-down automata. We briefly recall crucial definitions of finite tree automata as needed in this chapter. Several kinds of tree automata and basic constructions are presented, e.g., in [Löd11, CDG+07]. Recall that we denote the maximal rank of a ranked alphabet Σ by $mr(\Sigma)$.

Definition 10.1 (BTA)**.** Let Σ be a ranked alphabet. A *bottom-up finite state tree automaton* A (over Σ), *BTA* for short, is a tuple (P, Σ, δ, F) where

- P is a finite set of states,
- $F \subseteq P$ is a set of accepting states, and
- $\delta \subseteq \bigcup_{m=0}^{mr(\Sigma)} (P \times \Sigma^{(m)} \times P^m)$ is a finite set of transitions of the form $(p, \mathtt{a}, p_1 \dots p_m)$ where $\mathtt{a} \in \Sigma^{(m)}$ and $p, p_1, \dots, p_m \in P$.

A transition $(p, \mathtt{a}, p_1 \dots p_m)$ denotes that, if for all $i \in [m]$, BTA A arrives in state p_i after processing some tree u_i, then it can assign state p to the tree $\mathtt{a}(u_1, \dots, u_m)$. Technically, a *p-run* ρ of A on a tree $u = \mathtt{a}(u_1, \dots, u_m) \in \mathcal{T}_\Sigma$ is a tree

$$\rho = r(\rho_1, \dots, \rho_m) \in \mathcal{T}_\delta$$

where r is a transition $(p, \mathtt{a}, p_1 \dots p_m) \in \delta$ and ρ_i is a p_i-run of A for u_i. For some applications, it suffices to represent p-runs by trees from $\mathcal{T}_P$ with root p. The tree language $\mathcal{L}(A)$, accepted by A, consists of the trees $u \in \mathcal{T}_\Sigma$ by which A can reach an accepting state, i.e. it exists a p-run ρ of A for u with $p \in F$; the latter run is called *accepting run of A on u*. A bottom-up tree automaton $A = (P, \Sigma, \delta, F)$ is *deterministic* (DBTA) if for each symbol $\mathtt{a} \in \Sigma^{(m)}$ and every sequence $p_1 \dots p_m$ of states, there is at most one state p with $(p, \mathtt{a}, p_1 \dots p_m) \in \delta$, i.e., δ induces a partial function of type $\Sigma \times P^* \dashrightarrow P$. A BTA is called *complete* if there is at least one rule $(p, \mathtt{a}, p_1 \dots p_m) \in \delta$ for all $m \geq 0$, $\mathtt{a} \in \Sigma^{(m)}$, and $p_1, \dots, p_m \in P$.

We may also interpret the transitions of a BTA in a *top-down* fashion. Then we obtain the known top-down tree automaton (TTA), which starts at the input root node and assigns states to the children of a node, depending on the label of the node and the current state.

Definition 10.2 (DTTA)**.** A BTA is called *deterministic top-down* (*DTTA* for short) if the set of final states is a singleton set, and the transition relation δ induces a partial function $P \times \Sigma \dashrightarrow P^*$, i.e., for each state $p \in P$ and each symbol $\mathtt{a} \in \Sigma^{(m)}$, there is at most one sequence of states $p_1 \dots p_m \in P^m$ with $(p, \mathtt{a}, p_1 \dots p_m) \in \delta$.

As usual, the *size* $|A|$ of a finite state tree automaton A is the sum of sizes of all its transitions and the number of states where a transition $(p, \mathtt{a}, p_1 \dots p_m)$ has size $m+2$. Let BTA, DBTA, and DTTA denote the classes of all languages definable by BTAs, DBTAs, and DTTAs, respectively. It is known that BTA = DBTA equals the class of regular tree languages, and that DTTA is properly contained in this class.

Example 10.1. Coming back to the transformation from Example 9.1, the set of valid output documents should be lists of staff members (cf. the DTD on Page 149). More precisely:

- `staff` should contain a possibly empty sequence of `employee` elements;
- Each `employee` element should contain a `data` element and optionally, a `boss` element.

A bottom-up tree automaton describing (the binary representations of) this set is given by $A_{\text{staff}} = (P, \Sigma, \delta, F)$ where $P = \{r_{\text{staff}}, r_{\text{empl}}, r_{\text{data}}, r_{\text{name}}, r_{\text{boss}}, r_{\mathtt{e}}, \dots\}$ and δ contains (amongst others) the transitions

$$\begin{array}{llll}
(r_{\text{staff}}, & \texttt{staff}, & r_{\text{empl}}\ r_{\mathtt{e}}), & \\
(r_{\text{empl}}, & \texttt{employee}, & r_{\text{data}}\ r_{\text{empl}}), & (r_{\text{empl}}, \quad \mathtt{e}), \\
(r_{\text{data}}, & \texttt{data}, & r_{\text{name}}\ r_{\text{boss}}), & \\
(r_{\text{boss}}, & \texttt{boss}, & r_{\text{name}}\ r_{\mathtt{e}}), & (r_{\text{boss}}, \quad \mathtt{e}), \\
(r_{\text{name}}, & \texttt{name}, & r_{\text{content}}\ r_{\mathtt{e}}), & \\
(r_{\mathtt{e}}, & \mathtt{e}) &&
\end{array}$$

where r_{content} is the state characterizing valid personal data of employees. The set of accepting states is, thus, given by $F = \{r_{\text{staff}}\}$. Note that this BTA is, in fact, deterministic top-down. Note further that the documents adhering to the DTD on the beginning of Chapter 10 are, in general, no ranked trees but forests. ◁

In the previous example, the BTA ran on the first-child next-sibling encoding of forests as binary ranked trees. For convenience, we call such a finite tree automaton also *finite forest automaton* (short: BFA). Again, if it is deterministic or deterministic top-down, we abbreviate it DBFA and DTFA, respectively.

10.3 Basic Properties of BTAs

The approach, which we advocate here, is called *forward type checking* (cf. Section 10.1). Assume that O is the type of all valid output trees. In order to check that the transducer M produces only outputs in O, we construct a transducer, which, for every input u, only produces those output trees of transducer M, which are *not* valid, i.e., which are in $\overline{O}$ (the complement of O). Thus, type correctness for M is reduced to emptiness of the auxiliary transducer $M_{\overline{O}}$. For this idea to work, it is useful to have effective constructions which take the specification of a type and return a specification for its complement. For a *complete* DBTA $A = (P, \Sigma, \delta, F)$, this construction is simple: We need to exchange accepting and non-accepting states, i.e., replace F with $P \backslash F$. Since every regular tree language can be accepted by a complete DBTA, this construction implies that the complement of a regular tree language is a regular tree language, too. The complement of a type described by a deterministic top-down tree automaton is a regular language as well, but not necessarily in DTTA. The obvious technique for constructing an automaton for the complement is, therefore, to transform the deterministic top-down automaton into a complete deterministic bottom-up automaton and then apply the complement construction for complete DBTAs. This first construction, however, possibly incurs an exponential blow-up in the number of states. Therefore, we approve a different approach: Instead of constructing a *deterministic* automaton for the complement, we construct a *non-deterministic* automaton. The latter can be achieved by only moderately increasing the size.

Lemma 10.1. *For a DTTA A over the ranked alphabet Σ, there is a BTA A' over Σ with $\mathcal{L}(A') = \mathcal{T}_\Sigma \backslash \mathcal{L}(A)$ and $|A'| \in \mathcal{O}((|A| + |P| \cdot |\Sigma|) \cdot mr(\Sigma))$ where P is the set of states of A.*

Proof. Intuitively, the automaton A' guesses a path in the input tree to some node where the original automaton A fails. Formally, let $A = (P, \Sigma, \delta, \{p_0\})$ be a DTTA and define BTA $A' = (P', \Sigma, \delta', \{p'_0\})$ with $P' = \{p' \mid p \in P\} \cup \{\bullet\}$ for a new state $\bullet \notin P$. A state p' is meant to generate only trees for which there is *no* p-run of A. The state $\bullet$ describes arbitrary trees, i.e., the language $\mathcal{T}_\Sigma$. The set δ' of transitions of the new BTA is defined as follows:

- For every transition $(p, \mathtt{a}, p_1 \dots p_m) \in \delta$ with $m = rank(\mathtt{a}) \geq 1$, and for every $i \in [m]$ let $(p', \mathtt{a}, \bullet^{i-1} p'_i \bullet^{m-i}) \in \delta'$.
- For every state $p \in P$, $0 \leq m \leq mr(\Sigma)$, and $\mathtt{a} \in \Sigma^{(m)}$, $(p', \mathtt{a}, \bullet^m) \in \delta'$ whenever $\forall p_1, \dots, p_m \in P : (p, \mathtt{a}, p_1 \dots p_m) \notin \delta$.
- For every $\mathtt{a} \in \Sigma^{(m)}$, $(\bullet, \mathtt{a}, \bullet^m) \in \delta'$.

For the correctness of the construction, we claim that for every state p of the DTTA A, and every input tree u, the BTA A' has a p'-run on u iff DTTA A has no p-run on u. This claim can be proven by induction on the height of input trees.

Now let $k = mr(\Sigma)$ be the maximal rank of symbols in Σ. For each transition in δ, we get at most k new transitions in δ' (one for each successor state). Additionally, we possibly require a new rule of length at most $k+2$ for each pair of a state in P' and a symbol in Σ. Thus, the size of the new automaton A' is in $\mathcal{O}((|A| + |P| \cdot |\Sigma|) \cdot mr(\Sigma))$. □

Example 10.2. For the DTTA $A_{\text{staff}} = (P, \Sigma, \delta, \{r_{\text{staff}}\})$ in Example 10.1, the bottom-up tree automaton $A'_{\text{staff}} = (P', \Sigma, \delta', \{r'_{\text{staff}}\})$ for the complement has the following transitions for the label `staff`:

$$(r'_{\text{staff}}, \texttt{staff}, r'_{\text{empl}}\bullet), \quad (r'_{\text{staff}}, \texttt{staff}, \bullet r'_{\texttt{e}}), \quad (r', \texttt{staff}, \bullet\bullet)$$

for all $r' \in P' \setminus \{r'_{\text{staff}}\}$. For the label `boss` it has the transitions:

$$(r'_{\text{boss}}, \texttt{boss}, r'_{\text{name}}\bullet), \quad (r'_{\text{boss}}, \texttt{boss}, \bullet r'_{\texttt{e}}), \quad (r', \texttt{boss}, \bullet\bullet)$$

for all $r' \in P' \setminus \{r'_{\text{boss}}\}$. The set of accepting states is $\{r'_{\text{staff}}\}$. ◁

10.4 Notes and References

XML type definition languages such as DTDs [BPS08], XML Schema [FW04], or RELAX NG [CM01] are closely related to the regular tree languages [MLM00, Nev02] that is, to the class of tree languages recognized by finite tree automata.

Tree automata are a well studied formalism in computer science, dating back to the late 1960s. For surveys on tree automata, please see [GS84, GS97, CDG+07]. Tree automata inherit most of the good properties of finite automata on strings, such as effective closure under Boolean operations and decidability of emptiness. An important property, which will be used later for type checking, is that emptiness of BTAs can be decided in linear time (see, e.g., [Löd11], Theorem 1.7.4 in [CDG+07]).

Theorem 10.2. *Given a BTA A it can be decided in linear time whether or not $\mathcal{L}(A) = \emptyset$.*

Just as in the string case, nondeterministic bottom-up tree automata can be determinized (with a potential and sometimes unavoidable exponential blow up in automaton size). This is not the case for top-down tree automata: The class DTTA of languages accepted by deterministic top-down tree automata is a strict subclass of BTA, which does not even contain all finite languages. A famous example of a language, which is not in DTTA, is the set $U = \{\texttt{f(a,b)}, \texttt{f(b,a)}\}$. Note that for a given BTA, it is decidable if its language is in DTTA. This is due to the fact that DTTA languages can be characterized by the "path-closed" property [Cou78, Vir80]; the latter means that the trees in the languages are exactly obtained by combining all paths of the corresponding path language. The language U for instance is not path-closed. Using a similar example, it is easily shown that DTTA is not closed under complementation (and neither under union).

Chapter 11

Type Checking of Tree Walking Transducers

In this chapter, we present techniques to type check 2TTs against regular tree languages. We use the approach of forward type inference presented in the previous chapter. Thereto, we start with intersecting tree walking transducers with complements of output types where the output types are given by different types of tree automata.

11.1 Intersecting 2TTs with Output Types

For a given tree walking transducer, we build a second 2TT, which produces only output trees in the *complement* of the output type, and otherwise realizes the same transformation as the original transducer. If the output type is described by a complete DBTA $A = (P, \Sigma, \delta, F)$, the complement will be recognized by the complete DBTA $\bar{A} = (P, \Sigma, \delta, P \setminus F)$. Furthermore, for a given DTTA exists a BTA describing the complement (cf. Lemma 10.1). Thus, it is sufficient to construct a 2TT M_A for a 2TT M and a BTA A (which may be the complement automaton of a complete DBTA or of a DTTA) with $\tau_{M_A}(u) = \tau_M(u) \cap \mathcal{L}(A)$ for every tree u.

Theorem 11.1. *For every 2TT M and every BTA A, there is a 2TT M_A with*

$$\tau_{M_A}(u) = \tau_M(u) \cap \mathcal{L}(A)$$

for all $u \in \mathcal{T}_\Sigma$. The size $|M_A|$ of M_A is in $\mathcal{O}(|M| \cdot |A|^{d+1})$ where d is the maximal number of occurrences of states in right-hand sides of M.

Proof. Let $M = (Q, \Sigma, R, Q_0)$ and $A = (P, \Sigma, \delta, F)$. For each state q in Q and all states $p \in P$, we generate new states for M_A of the form $\langle q, p \rangle$. Such a state is meant to generate only trees $u \in \mathcal{T}_\Sigma$ for which there is a run of A starting at

the leaves and reaching the root of u in state p. The rules of the new 2TT M_A are

$$\langle q, p\rangle(\mathtt{a}, \eta) \to \zeta'$$

for every rule $q(\mathtt{a}, \eta) \to \zeta$ of M and $\zeta' \in \tau^p[\zeta]$. The sets $\tau^p[.]$ are inductively defined by:

$$\tau^p[\mathtt{b}(\zeta_1, \ldots, \zeta_m)] = \{\mathtt{b}(\zeta'_1, \ldots, \zeta'_m) \mid (p, \mathtt{b}, p'_1 \ldots p'_m) \in \delta \ \wedge\ \forall i : \zeta'_i \in \tau^{p'_i}[\zeta_i]\}$$
$$\tau^p[q'(op)] = \{\langle q', p\rangle(op)\} .$$

The set of initial states of M_A is $Q'_0 = Q_0 \times F$. By fixpoint induction, we verify for every state q, every input tree $u \in \mathcal{T}_\Sigma$, every node $\vartheta \in \mathtt{Nodes}(u)$, and every state p that

$$[\![\langle q, p\rangle]\!]_u(\vartheta) = [\![q]\!]_u(\vartheta) \cap \{v \in \mathcal{T}_\Sigma \mid \exists \text{ run } \rho \text{ on } v \text{ with } \rho(\epsilon) = p\} .$$

For each state in M we have at most $|A|$ new states in M_A. If we have c occurrences of state calls in the right-hand side of a rule r of M, with the state on the left-hand side, we obtain at most $|A|^{c+1}$ new rules for r in M_A. Therefore, the new 2TT is of size $\mathcal{O}(|M| \cdot |A|^{d+1})$ where d is the maximal number of occurrences of state calls in right-hand sides in M. □

Considering only binary trees, we obtain size $\mathcal{O}(|M| \cdot |A|^3)$ for the intersection 2TT (with Lemma 9.1). The last step is to decide whether $\tau_{M_A} \neq \emptyset$. Thereto, we build a BTA describing the domain of M_A. This will be done after completing the example.

Example 11.1. Let us try to type check the 2TT $M_{\text{staff}} = (Q, \Sigma, R, Q_0)$ via forward type inference. According to Lemma 9.1, we restrict the maximal number of state calls in right-hand sides to 2. In our example 2TT, the rule

$$q(\mathtt{employee}, \eta) \to \mathtt{employee}(\mathtt{data}(q_{data}(down_1), q_{boss}(stay)), q_{sub}(down_2))$$

has three state calls. We obtain the new rules:

$$q(\mathtt{employee}, \eta) \to \mathtt{employee}(q'(stay), q_{sub}(stay))$$
$$q'(\mathtt{employee}, \eta) \to \mathtt{data}(q_{data}(down_1), q_{boss}(stay)) .$$

According to the proof of Lemma 9.1, the new state q' in these rules is the state

$$q_{\mathtt{employee},\mathtt{data}(q_{data}(down_1), q_{boss}(stay))} .$$

As output type, consider again the DTTA $A_{\text{staff}} = (P, \Sigma, \delta, \{r_{\text{staff}}\})$. The complement BTA $A'_{\text{staff}} = (P', \Sigma, \delta', \{r'_{\text{staff}}\})$ is given in Example 10.2. Then, the intersection transducer $(M_{\text{staff}})_{A'_{\text{staff}}}$ is given by $(Q \times P', \Sigma, R', Q_0 \times \{r'_{\text{staff}}\})$. In what follows, we show how the first few rules in R' are constructed from R and δ'.

For $q_I(\mathtt{department}, 0) \to \mathtt{staff}(q(down_1), \mathtt{e})$ and r'_{staff}, we obtain the rule

$$\langle q_I, r'_{\text{staff}}\rangle(\mathtt{department}, 0) \to \mathtt{staff}(\langle q, r'_{\text{empl}}\rangle(down_1), \mathtt{e}) .$$

For the rule $q(\texttt{employee}, \eta) \to \texttt{employee}(q'(stay), q_{sub}(down_2))$ and r'_{empl}, we obtain

$$\langle q, r'_{\text{empl}}\rangle(\texttt{employee}, \eta) \to \texttt{employee}(\langle q', r'_{\text{data}}\rangle(stay), \langle q_{sub}, \bullet\rangle(down_2))$$
$$\langle q, r'_{\text{empl}}\rangle(\texttt{employee}, \eta) \to \texttt{employee}(\langle q', \bullet\rangle(stay), \langle q_{sub}, r'_{\text{empl}}\rangle(down_2)) \ .$$

For $q(\texttt{e}, \eta) \to q_{up}(up)$ and all $r \in P'$, we obtain

$$\langle q, r\rangle(\texttt{e}, \eta) \to \langle q_{up}, r\rangle(up) \ . \qquad \triangleleft$$

11.2 Deciding Emptiness of 2TTs

In order to check the emptiness of a tree walking transducer w.r.t. a given input type, we construct a nondeterministic finite state automaton (cf. Section 10.2) which then is checked for emptiness. First, for a 2TT M and an input type I, we define an *alternating tree walking automaton* M', which ignores the output of the 2TT, but apart from that imitates the behavior of M on trees in I. For M', we then construct a nondeterministic BTA $A_{M'}$ accepting all trees u such that $\tau_M(u) \neq \emptyset$. The right-hand sides of transitions of an alternating tree walking automaton are conjunctions. Whereas the empty conjunction, i.e., $\bigwedge \emptyset$, equals true.

Definition 11.1 (ATWA)**.** An *alternating tree walking automaton* (ATWA for short) is a tuple $M = (Q, \Sigma, \delta_M, Q_0)$ where

- Q is a finite set of states,
- Σ a ranked input alphabet,
- $Q_0 \subseteq Q$ a set of initial states, and
- δ_M a finite set of rules of the form

$$q(\texttt{a}, \eta) \to q_1(op_1) \wedge \ldots \wedge q_c(op_c)$$

with $c \geq 0$, $q, q_1, \ldots, q_c \in Q$, $\texttt{a} \in \Sigma^{(m)}$, $m \geq 0$, $\eta \geq 0$, and for $1 \leq i \leq c$, $op_i \in \{stay, up\} \cup \{down_j \mid j \in [m]\}$.

A transition $q(\texttt{a}, \eta) \to H$ is also called q-rule. An ATWA traverses a tree like a 2TT, but produces no output. The language $\mathcal{L}(M)$ of an ATWA M is defined as the set of all trees for which an accepting run of M exists.

Definition 11.2 (Run)**.** For an ATWA $M = (Q, \Sigma, \delta_M, Q_0)$, assume that the rule $r : q(\texttt{a}, \eta) \to H$ is in δ_M with $H = q_1(op_1) \wedge \ldots \wedge q_c(op_c)$. Then the tree

$$\rho = r(\rho_1, \ldots, \rho_c) \in \mathcal{T}_{\delta_M}$$

is a (q, ϑ)-run of the ATWA M on the tree u if ϑ is a node of u with direction η and label $\texttt{a}$ and for all i, ρ_i is a (q_i, ϑ_i)-run of M on u where ϑ_i is obtained from ϑ by operation op_i. A (q_0, ϵ)-run ρ with $q_0 \in Q_0$ is also called *accepting*.

A subtree $\rho[w]$ of a run ρ on a node w, which is a (q, ϑ)-run for some state q and some node ϑ of u is called (q, ϑ)-*subrun*. The set of rules, which are applied to one node ϑ during a run ρ on u, is the set

$$rules_\rho(\vartheta) = \{r \mid \exists \rho_1, \ldots, \rho_c : r(\rho_1, \ldots, \rho_c) \text{ is a } (q, \vartheta)\text{-subrun of } \rho\} .$$

Lemma 11.2. *Assume that M is a 2TT. Then an ATWA M' can be constructed in linear time such that for every input tree u, M' has an accepting run for u iff M has an accepting run for u.*

Proof. The ATWA M' has the same set of states as M. The rules of M' are obtained from those of M by replacing every right-hand side ζ of M with the conjunction of all $q(op)$ occurring in ζ. Formally, let $M = (Q_M, \Sigma, R, Q_0)$ be a 2TT. Then, the ATWA M' is defined by $M' = (Q_M, \Sigma, \delta', Q_0)$ for a set δ' of transitions $\delta' = \{[r] \mid r \in R\}$ where

$$\begin{aligned} [q(\mathtt{a}, \eta) \to \zeta] &= q(\mathtt{a}, \eta) \to [\zeta] \\ [\zeta] &= q_1(op_1) \wedge \ldots \wedge q_c(op_c) \\ &\qquad \text{for a right-hand side } \zeta = z[q_1(op_1), \ldots, q_c(op_c)] . \end{aligned}$$

In order to prove the correctness of this construction, we first extend the translation [.] from rules to trees of rules. For $\rho = r(\rho_1, \ldots, \rho_c) \in \mathcal{T}_R$, $[\rho]$ is inductively defined as the tree $[\rho] = [r]([\rho_1], \ldots, [\rho_c])$. Then we claim that ρ is a (q, ϑ)-run of M iff $[\rho]$ is a (q, ϑ)-run of M'. The proof is by structural induction on ρ. Since for every (q, ϑ)-run ρ' of M', some $\rho \in \mathcal{T}_R$ exists with $[\rho] = \rho'$, we conclude that $\tau_M(u) \neq \emptyset$ iff $u \in \mathcal{L}(M')$. □

As an example for this translation of tree walking transducers into ATWAs, consider the 2TT M_{staff} (Example 9.1). For the second rule, we get:

$$r_2' : q(\mathtt{employee}, \eta) \to q_{data}(down_1) \wedge q_{boss}(stay) \wedge q_{sub}(down_1) .$$

In order to accept only trees of an input type I, we enlarge an ATWA M. Let I be given by a finite state tree automaton $A = (P, \Sigma, \delta_A, F)$ and ATWA $M = (Q, \Sigma, \delta_M, Q_0)$. For every rule $q_0(\mathtt{a}, \eta) \to H$ of M with $q_0 \in Q_0$ and every $p_f \in F$, we enhance the ATWA with the rule $q_0(\mathtt{a}, \eta) \to p_f(stay) \wedge H$. Additionally, for every transition $(p, \mathtt{a}, p_1 \ldots p_m) \in \delta_A$ of the automaton A, we add the rule $p(\mathtt{a}, \eta) \to \bigwedge_{i \leq m} p_i(down_i)$.

In an accepting run of an ATWA, subruns for the same state and node are interchangeable. Thus, we define *uniform* runs as runs where for each state q and each node ϑ, the (q, ϑ)-subruns are the same.

Definition 11.3 (Uniform Run)**.** A (q, ϑ)-run ρ of an ATWA M on a tree u is called *uniform* if, for every state q, every node ϑ in u, and every two (q, ϑ)-subruns ρ_1, ρ_2 of ρ, $\rho_1 = \rho_2$.

If an ATWA is deterministic, i.e., for each state q, direction η and label $\mathtt{a}$, there is at most one rule of the form $q(\mathtt{a}, \eta) \to H$, then *every* (q', ϑ)-run is uniform. In general, this may not be the case. However, we have:

Lemma 11.3. *For an ATWA M and an input tree u, the following two statements are equivalent.*

1. *M has an accepting run for u.*
2. *M has a uniform accepting run for u.*

Proof. Every uniform accepting run is an accepting run. For the reverse direction, it suffices to prove that for every (q, ϑ)-run ρ of M on u, there is also a uniform (q, ϑ)-run of M on u. For that, we consider the set $B(\rho)$ of all pairs (q', ϑ') for which ρ contains more than one (q', ϑ')-run as a subtree. We proceed by induction on the cardinality of the set $B(\rho)$. If the set $B(\rho)$ is empty, ρ is already uniform. Now assume $B(\rho)$ is non-empty. Then ρ contains a subtree ρ_1 with the following two properties:

1. ρ_1 is a (q_1, ϑ_1)-run with $(q_1, \vartheta_1) \in B(\rho)$;
2. all subtrees ρ_1' of ρ_1 are (q', ϑ')-runs with $(q', \vartheta') \notin B(\rho)$.

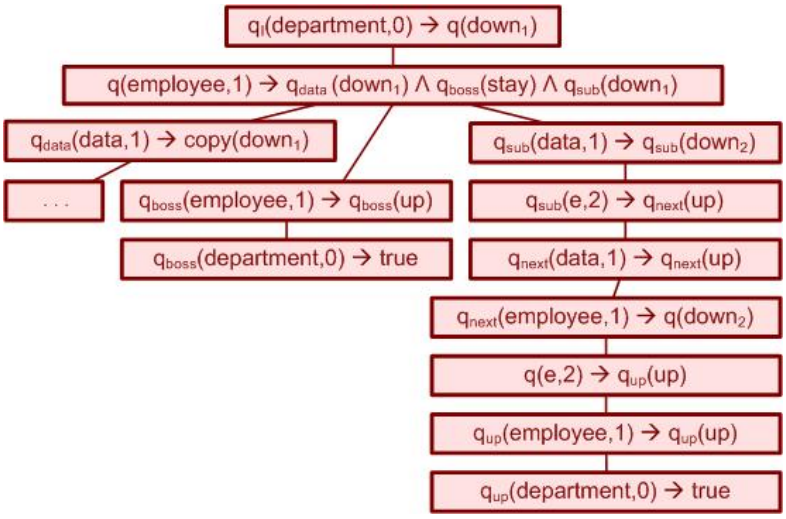
Then we construct from the run ρ a tree ρ' by replacing every occurrence of a subtree in ρ, which is a (q_1, ϑ_1)-run, with ρ_1. Then ρ' is again a (q, ϑ)-run on u but now, the set $B(\rho') \subseteq B(\rho) \backslash \{(q_1, \vartheta_1)\}$ contains at least one element less. Thus, by induction hypothesis applied to ρ', there is a (q, ϑ)-run on u, which is uniform. □

Figure 9.2(a) shows the behavior of the 2TT M_{staff} on the tree u_e. The corresponding ATWA M'_{staff} yields the same behavior. The Figure 9.2(b) shows an accepting run ρ of M_{staff} on u_e. The corresponding run $[\rho]$ of M'_{staff} (Figure 11.1(a)) is uniform. In order to decide emptiness of an ATWA M and, accordingly, of a 2TT, we construct a *nondeterministic* bottom-up finite state tree automaton A_M. In order to accept the domain of M, the BTA A_M guesses uniform accepting runs. Since A_M visits each node in the input tree at most once, it *guesses* at every node all transitions, which are applied at this node during a uniform run of M.

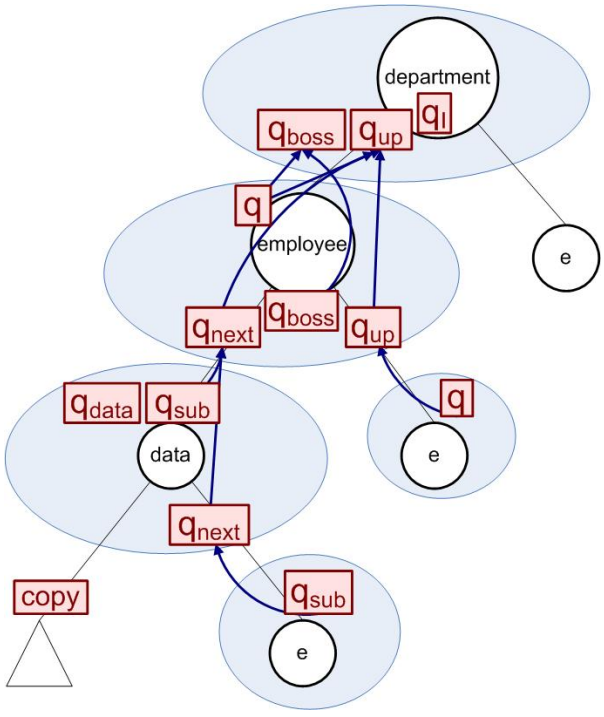
Technically, the states of the BTA A_M consist of guessed directions together with *partial* mappings $\mu : Q \dashrightarrow 2^Q$ of states to sets of states: $\mu(q) = B$ at a given node ϑ means that the (q, ϑ)-run on the input tree will cause calls $q'(up)$ at ϑ only for states q' from B. For a mapping μ, we refer to the domain as $\text{dom}(\mu) = \{q \mid \mu(q) \text{ is defined}\}$. Furthermore, to each node ϑ in the input tree we implicitly attach $\mathcal{R}_\vartheta = \mathit{rules}_\rho(\vartheta)$ collecting the ATWA rules which are applied to the node ϑ in an accepting run ρ of the ATWA on the input tree u. Since ρ is uniform, the set $\mathit{rules}_\rho(\vartheta)$ contains at most one q-rule for every q.

Example 11.2. Consider again the run ρ in Figure 11.1(a) on the tree u_e. To simulate this run on node 1.2 (with label `e` and direction 2), a run of the BTA A_M assigns to 1.2 the partial mapping $\mu_{1.2} = \{q \mapsto \{q_{up}\}\}$. In addition, we attach $\mathcal{R}_{1.2} = \{q(\mathtt{e}, 2) \to q_{up}(up)\}$ to this node. At node 1.1 with label `data` and direction 1, we obtain the partial mapping

$$\mu_{1.1} = \{q_{data} \mapsto \emptyset, q_{next} \mapsto \{q_{next}\}, q_{sub} \mapsto \{q_{next}\}\}$$



(a) A uniform run of the ATWA M'_{staff} on the tree u_e.



(b) The mappings μ for that run.

Figure 11.1: Behavior of M'_{staff} on u_e.

because q_{next} at node 1 also depends on q_{sub} at node 1.1. For the run ρ we get

$$\begin{aligned}\mathcal{R}_{1.1} = \{&q_{data}(\mathtt{data}, 1) \to copy(down_1),\\ &q_{sub}(\mathtt{data}, 1) \to q_{sub}(down_2),\\ &q_{next}(\mathtt{data}, 1) \to q_{next}(up)\}.\end{aligned}$$

The mappings for the run ρ are illustrated in Figure 11.1(b). ◁

A pair $\langle \mu, \eta \rangle$ is accepting if $\eta = 0$, for all $q \in \mathrm{dom}(\mu)$ holds $\mu(q) = \emptyset$, and $\mathrm{dom}(\mu)$ contains an accepting state $q_0 \in Q_0$ of the ATWA M.

Now assume that $\mathtt{a}$ is a label of arity m, η is a direction, and $\mu, \mu_i : Q \dashrightarrow 2^Q$ are partial mappings ($i = 1, \ldots, m$). Then

$$(\langle \mu, \eta \rangle, \mathtt{a}, \langle \mu_1, 1 \rangle, \ldots, \langle \mu_m, m \rangle)$$

is a transition of the BTA A_M iff there is a set $\mathcal{R}$ of rules of the ATWA M with the following properties. Let Q_s, Q_u, and $Q_{d,i}$ ($i \in [m]$) denote the set of states q for which there is a q-rule in $\mathcal{R}$, the set of states q' with a recursive call $q'(up)$, and the sets of states q' with a recursive call $q'(down_i)$ in some right-hand side of rules in $\mathcal{R}$, respectively. Then the set $\mathcal{R}$ of rules should have the following properties:

1. All rules in $\mathcal{R}$ have a left-hand side of the form $q(\mathtt{a}, \eta)$ where $q \in Q$.
2. Assume $q(\mathtt{a}, \eta) \to q_1(op_1) \wedge \ldots \wedge q_c(op_c) \in \mathcal{R}$. Then we have for all $j \leq c$:
   - If $op_j = stay$, then $\mathcal{R}$ also contains a q_j-rule, i.e., $q_j \in Q_s$;
   - If $op_j = down_i$, then $q_j \in \mathrm{dom}(\mu_i)$.
3. Whenever $q' \in \mathrm{dom}(\mu_i)$ for i, then $\mu_i(q') \subseteq Q_s$.
4. Consider the following graph G with set of vertices

   $$V = \{q(stay) \mid q \in Q_s\} \cup \{q(down_i) \mid i \in [m], q \in Q_{d,i}\} \cup \{q(up) \mid q \in Q_u\}$$

   and the following set E of edges:

   - If a q-rule in $\mathcal{R}$ contains a call $q'(op)$, then $(q(stay), q'(op)) \in E$;
   - If $\mu_i(q) = B_i$ is defined, then $(q(down_i), q'(stay)) \in E$ for all $q' \in B_i$.

   The resulting directed graph $G = (V, E)$ should be acyclic, and the mapping μ is obtained from G as follows:

   - $q \in \mathrm{dom}(\mu)$ iff $q \in Q_s$ and
   - $\mu(q) = B$ iff the set B equals the set of all vertices $q'(up)$ which are reachable in G from $q(stay)$.

The size of A_M is exponential in the size of M. We give a detailed example for this construction in Example 11.3. Now we state the correlation of runs of A_M and of M.

Lemma 11.4. *For a tree u the following statements are equivalent.*

1. *There is a uniform accepting run of M on u.*
2. *There is an accepting run of A_M on u.*

Proof. (1) $\Rightarrow$ (2) : Let ρ be a uniform accepting run of the ATWA M on a tree u. For a node ϑ of u with label $\mathtt{a}$ and direction η, let $\mathcal{R}_\vartheta$ denote the set of all ATWA rules applied at the root in subruns of ρ starting at ϑ, i.e., $\mathcal{R}_\vartheta = \mathit{rules}_\rho(\vartheta)$. We then construct for every node ϑ of u with label $\mathtt{a} \in \Sigma^{(m)}$ and direction η a state μ_ϑ and a transition $r_\vartheta = (\langle \mu_\vartheta, \eta\rangle, \mathtt{a}, \langle \mu_{\vartheta 1}, 1\rangle \dots \langle \mu_{\vartheta m}, m\rangle)$ of the BTA A_M.

The sets $\mathcal{R}_\vartheta$ allow us to construct a directed graph G_u. The set V_u of vertices of G_u are given by the set of all pairs (q, ϑ) for nodes ϑ of u and states q for which there is a q-rule in $\mathcal{R}_\vartheta$. The set E_u of edges consists of:

- all pairs $((q, \vartheta), (q', \vartheta))$ where the q-rule in $\mathcal{R}_\vartheta$ contains a call $q'(\mathit{stay})$;
- all pairs $((q, \vartheta), (q', \vartheta i))$ where the q-rule in $\mathcal{R}_\vartheta$ contains a call $q'(\mathit{down}_i)$;
- all pairs $((q, \vartheta), (q', \vartheta'))$ where the q-rule in $\mathcal{R}_\vartheta$ contains a call $q'(\mathit{up})$ and $\vartheta = \vartheta' i$ for some i.

The graph G_u is acyclic. Moreover, since the uniform run ρ is accepting, every vertex in V_u is reachable from some vertex (q_0, ϵ) with $q_0 \in Q_0$.

The graph G_u allows to construct partial mappings μ_ϑ for every node ϑ. The state q is in $\mathrm{dom}(\mu_\vartheta)$ iff (q, ϑ) is a vertex of G_u. Assume (q, ϑ) is a vertex in the graph G_u. We consider two cases. If $\vartheta = \epsilon$, then $\mathcal{R}_\epsilon$ cannot contain any q-rule, which has an *up*-call. In this case, $\eta(\vartheta) = 0$, and we set $\mu_\epsilon(q) = \emptyset$. Otherwise, assume that $\vartheta = \vartheta' i$. Then $\eta(\vartheta) = i$ and $q' \in \mu_\vartheta(q)$ iff there is an edge $((q_1, \vartheta), (q', \vartheta'))$ in G_u where (q_1, ϑ) is reachable from (q, ϑ) by a path, which contains only vertices (q_2, ϑ_2) referring to nodes ϑ_2 from the subtree at node ϑ, i.e., to nodes, which have ϑ as a prefix. If no such state q' exists, then $\mu_\vartheta(q) = \emptyset$.

It now can be verified for every node ϑ with label $\mathtt{a} \in \Sigma^{(m)}$ and direction η that $(\langle \mu_\vartheta, \eta\rangle, \mathtt{a}, \langle \mu_{\vartheta 1}, 1\rangle \dots \langle \mu_{\vartheta m}, m\rangle)$ constitutes a transition of A_M (with $\mathcal{R}_\vartheta$ as set of rules of the ATWA). Since by construction, $\langle \mu_\epsilon, 0\rangle$ is an accepting state of A_M, we have, thus, constructed an accepting run of A_M for u.

(2) $\Rightarrow$ (1) : Let ρ' be an accepting run of A_M on the tree u, and let $\langle \mu_\vartheta, \eta\rangle$ and r_ϑ denote the state and transition of A_M attained for the node ϑ in u. We can find sets $\mathcal{R}_\vartheta$ conforming to the properties of the transition relation of M. These allow to construct a graph G'_u analogously to the graph G_u above. By definition of the transition relation of A_M, G'_u is acyclic. This allows us to define for every vertex (q, ϑ) in G'_u, the number $h(q, \vartheta)$ as the maximal length of a path in G'_u to a leaf, i.e., a vertex with out-degree 0. Using the sets $\mathcal{R}_\vartheta$ of ATWA rules, we now construct for every node ϑ and ATWA rule $r : q(\mathtt{a}, \eta) \to H$ from $\mathcal{R}_\vartheta$ with $H = q_1(\mathit{op}_1) \wedge \dots \wedge q_c(\mathit{op}_c)$, a tree $\rho[q, \vartheta]$ by

$$\rho[q, \vartheta] = r(\rho[q_1, \vartheta_1], \dots, \rho[q_c, \vartheta_c])$$

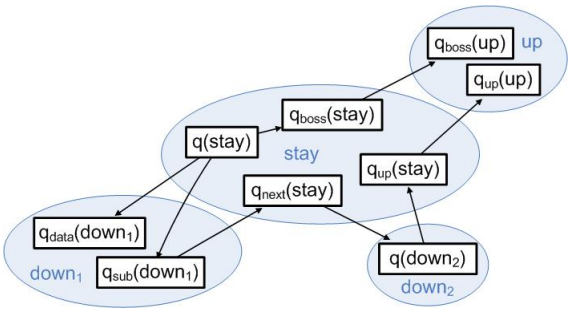


Figure 11.2: Graph G for transition $(\langle \mu_1, 1\rangle, \texttt{employee}, \langle \mu_{1.1}, 1\rangle, \langle \mu_{1.2}, 2\rangle)$.

where $\vartheta_j = [\![op_j]\!](\vartheta)$. Note that all these trees are well-defined, since the height of $\rho[q, \vartheta]$ precisely equals $h(q, \vartheta)$. Moreover, the tree $\rho[q, \vartheta]$ is a (q, ϑ)-run of the ATWA M on u. Since every (q', ϑ')-subrun of this tree equals the (q', ϑ')-run $\rho[q', \vartheta']$, this run is also uniform. In particular, the tree $\rho[q_0, \epsilon]$ constitutes a uniform accepting run of the ATWA M. □

By Lemmas 11.2, 11.3, and 11.4, the BTA A_M recognizes the domain of the given 2TT M, which gives us Theorem 11.5. Note that this implies, by Theorem 10.2, that also emptiness of M's domain (and hence of M's translation τ_M) can be decided in exponential time.

Theorem 11.5. *Assume M is a 2TT. Then a BTA A can be constructed in exponential time such that $\mathcal{L}(A) = dom(M)$. Thus, emptiness for a 2TT can be decided in deterministic exponential time.* □

We continue with our example to illustrate the construction of the bottom-up tree automaton:

Example 11.3. We consider again the tree walking transducer M_{staff} and its corresponding ATWA M'_{staff}. The size of the BTA $A_{M'_{\text{staff}}}$ is exponential in the size of the ATWA or 2TT. Therefore, we only construct states occurring in a run of $A_{M'_{\text{staff}}}$ on the tree in Figure 9.2(a):

$$u_e = \texttt{department}(\texttt{employee}(\texttt{data}(\ldots, \texttt{e}), \texttt{e}), \texttt{e})$$

The uniform accepting run ρ in Figure 11.1(a) yields the sets $\mathcal{R}_\vartheta$. For instance, for the node labeled with `employee`, we get

$$\begin{aligned}\mathcal{R}_1 = \{ & q(\texttt{employee}, 1) \to q_{boss}(stay) \wedge q_{sub}(down_1) \wedge q_{data}(down_1),\\ & q_{boss}(\texttt{employee}, 1) \to q_{boss}(up),\\ & q_{up}(\texttt{employee}, 1) \to q_{up}(up),\\ & q_{next}(\texttt{employee}, 1) \to q(down_2)\} \,.\end{aligned}$$

The graph G_{u_e} in the proof of Lemma 11.4 is similar to the graph in Figure 9.2(a). There, it spans the tree u_e. To obtain the graph G_{u_e}, we have to replace a vertex q, which is located at a node ϑ of u_e, by (q, ϑ).

The mappings μ_ϑ are illustrated in Figure 11.1(b). For each state q fixed at a node ϑ, $\mu_\vartheta(q)$ is defined. If q has no outgoing edges then $\mu_\vartheta(q) = \emptyset$. Otherwise, it is the set of all direct successors of q in this Figure 11.1(b). For example:

$$\begin{array}{lcl}
\mu_\epsilon & = & \{q_I \mapsto \emptyset, q_{up} \mapsto \emptyset, q_{boss} \mapsto \emptyset\} \\
\mu_1 & = & \{q \mapsto \{q_{up}, q_{boss}\}, q_{next} \mapsto \{q_{up}\}, q_{boss} \mapsto \{q_{boss}\}, q_{up} \mapsto \{q_{up}\}\} \\
\mu_{1.1} & = & \{q_{data} \mapsto \emptyset, q_{sub} \mapsto \{q_{next}\}, q_{next} \mapsto \{q_{next}\}\} \\
\mu_{1.2} & = & \{q \mapsto \{q_{up}\}\}
\end{array}$$

Note that $q_{up} \notin \mu_{1.1}(q_{next})$ although $(q_{up}, 1)$ is reachable from $(q_{next}, 1.1)$ in G_{u_e}, but the path contains $(q, 1.2)$ and 1.1 is not a prefix of 1.2. In order to illustrate a transition of BTA $A_{M'_{\text{staff}}}$, consider, e.g., the transition

$$(\langle \mu_1, 1\rangle, \texttt{employee}, \langle \mu_{1.1}, 1\rangle, \langle \mu_{1.2}, 2\rangle) \in \delta_A$$

with the set $\mathcal{R}_1$. All rules in $\mathcal{R}_1$ agree in the input label `employee` and the direction 1 (Condition 1, Page 161). Also it contains a q_{boss}-rule for the call $q_{boss}(stay)$. For all states of occurring $down_1$-calls, i.e., states q_{sub} and q_{data}, the mapping $\mu_{1.1}$ is defined. Likewise, for q with a $down_2$-call, $q \in \text{dom}(\mu_{1.2})$. Thus, $\mathcal{R}_1$ also conforms with Condition 2 (Page 161). For Condition 3, we verify that $\mathcal{R}_1$ has rules both for q_{next} and q_{up}. For the last property, we construct the graph G. The set of vertices is

$$\begin{aligned}
V = \{&q(stay), q_{boss}(stay), q_{next}(stay), q_{up}(stay), \\
&q_{data}(down_1), q_{sub}(down_1), q(down_2), \\
&q_{boss}(up), q_{up}(up)\}
\end{aligned}$$

The edges are illustrated in Figure 11.2. As the last condition (Page 161) requires, the graph G is acyclic, and we can read off the mapping μ_1.

We also verify that $\langle \mu_\epsilon, 0\rangle$ is an accepting state. According to Lemma 11.4, the resulting run ρ' is an accepting run of $A_{M'_{\text{staff}}}$ on u_e. ◁

11.3 Efficient Subcases

In the previous section, we have provided an algorithm for deciding emptiness of ATWAs and thus, also of 2TTs, which runs in exponential time. This algorithm is indeed worst-case optimal. Not withstanding that, this algorithm allows us to identify subclasses of transducers where emptiness can be decided in polynomial time.

We call a tree walking transducer M *b-bounded* if every accepting run ρ of M has at most b subruns starting at node ϑ. We call M *strictly b-bounded* if every accepting run ρ of M visits each node ϑ in the input tree at most b times, i.e.,

has at most b subrun *occurrences* starting at node ϑ. The same definitions are also employed for alternating tree walking automata.

Note that the definition of b-boundedness does not exclude that the same node ϑ is traversed arbitrarily often: If so, however, these traversals will be copies of at most b distinct traversals. Note further that for a given transducer M, it is decidable whether or not there exists a b such that M is b-bounded; the same holds for strict b-boundedness. To see this, we add for each input symbol a new marked symbol of the same rank. We then consider input trees in which exactly one node is labeled by a marked symbol (this is a regular input tree language). Finally, we change the transducer M in such a way that it produces a specific output tree for each subrun that starts at the marked input node (resp. for each time the marked node is visited), and other than that does not produce any output. The output of the new transducer, when applied to input trees with exactly one node marked, is finite if and only if the transducer is b-bounded for some b (resp. strictly b-bounded for some b). The finiteness is decidable for a very large class of tree transformations [DE98], which contains the pebble tree transducers (and hence also the 2TTs) by [EM03b].

Example 11.4. Our current example 2TT M_{staff} is b-bounded — but not strictly b-bounded. Let ρ be an arbitrary run of M_{staff} on a tree u. Because the transducer is deterministic, ρ is uniform. Hence, for every node ϑ in u, there is at most one (q, ϑ)-subrun for every state q. For a node ϑ with label `employee`, there are 4 different states q' with rules of the form $q'(\texttt{employee}, \eta) \to \zeta$ in M_{staff}. For every other label, there are 3 or 2 different states. Thus, M_{staff} is 4-bounded.

Note the difference between subruns and occurrences of subruns. There may be different occurrences of the same subrun. Consider for example a tree u with a node ϑ, which represents the boss of n subordinates. Then, the transducer searches the boss for every subordinate again. Hence, for an accepting run of M_{staff} on u we have n occurrences of (q_{boss}, ϑ)-subruns. Thus, the 2TT M_{staff} is *not* strictly b-bounded for every b. ◁

Consider the construction from the last section of a BTA A_M which accepts the same language as an ATWA M. If the alternating tree walking automaton M is b-bounded, it suffices to consider sets $\mathcal{R}$ of ATWA rules of size b. Also, this means that partial mappings μ need to be taken into account, which are of the form: $B \dashrightarrow 2^{B'}$ for subsets B and B' of states of cardinalities at most b. Note that the number of subsets of size at most b of a set with n elements is bounded by $\frac{1}{b!}(n+1)^b$. Thus, the number of the partial mappings μ can be bounded by:

$$(n+1)^{2b}\frac{2^{b^2}}{b!}$$

if n is the number of states of M. We will not provide an explicit estimation of the number of transitions of the BTA since it crucially also depends on other parameters such as the number of rules of M that agree in input symbol and direction (which is typically small). We just note that for b-bounded M, the size of the BTA A_M is polynomial in the size of M. The occurring exponent, though, is bounded by $\mathcal{O}(kb^2)$ where k is the maximal rank of an input symbol.

Summarizing, we find that emptiness for b-bounded 2TTs can be done in polynomial time. Note, however, that neither b-boundedness or strict b-boundedness is preserved by our construction to reduce the number of state calls in right-hand sides. For an efficient method for type checking, we also require that the bound on the number of visits to every node in the input tree is preserved under the intersection construction (Section 11.1). In this respect, we observe:

Lemma 11.6. *If M is strictly b-bounded and A is a BTA, then M_A is also strictly b-bounded. If M is just b-bounded, this need not be the case.*

We thus obtain a polynomial-time algorithm for the class of strictly b-bounded 2TTs where the number of occurrences of state calls in right-hand sides is also bounded.

11.4 Conclusion

In this chapter, we presented techniques to type check tree walking transducers against regular tree languages. Our approach is forward type inference. For that purpose, for a given 2TT M, we build a second tree walking transducer, which produces only output trees in the complement of the output type, and otherwise realizes the same transformation as the original transducer M. For a BTA A describing the complement of the output type, the size of the new 2TT M_A is in $\mathcal{O}(|M| \cdot |A|^{d+1})$ where d is the maximal number of occurrences of states in right-hand sides in M; for binary trees and with Lemma 9.1, it is in $\mathcal{O}(|M| \cdot |A|^3)$.

For this intersection 2TT M_A, we build an alternating tree walking automaton M', which imitates the behavior of M_A, but does not produce any output. This construction can be done in linear time. Then, in order to decide emptiness of the ATWA M' and, accordingly, of the transducer M_A, we construct a nondeterministic bottom-up finite state tree automaton $A_{M'}$. In general, this construction is exponential in the size of M'. Hence, emptiness of a 2TT can be decided in exponential time — a result, which has already been known for a long time, see the notes at the end of Section 11.5. The general approach, however, allowed us to identify more efficient subclasses that have been discussed in Section 11.3. If M' is b-bounded, emptiness can be decided in polynomial time where the exponent of the polynomial only depends on b^2 if the transducer is two-way, i.e., uses *up*-operations. A closer inspection of the construction of a bottom-up tree automaton from an ATWA, though, reveals that the exponent can be reduced to b if the transducer is stay top-down, i.e., uses no *up*-operations (but possibly *stay*-operations). The construction for the intersection, on the other hand, is polynomial in the sizes both of the 2TT and the BTA — but may be exponential in the number of occurrences of state calls in right-hand sides. Also, if we start with a b-bounded 2TT M, the construction may not preserve b-boundedness. Instead, the bound on the number of visits to an input node may be increased as much as by a factor of the number of

states of the BTA. If the tree walking transducer M is *strictly* b-bounded, this property will be retained and also the alternating tree walking automaton M' is strictly b-bounded.

As a last step of verifying whether a tree walking transducer M type checks w.r.t. input and output types $\mathcal{L}(A_I)$ and $\mathcal{L}(A)$ (for a BTA A_I), we construct a further bottom-up tree automaton C with $\mathcal{L}(C) = L(A_{M'}) \cap \mathcal{L}(A_I)$ using the obvious product construction (see, e.g., Section 1.3 in [CDG+07]) such that

$$|C| \in \mathcal{O}(|A_{M'}| \cdot |A_I|) \; .$$

According to Theorem 10.2, we can test whether $\mathcal{L}(C) = \emptyset$ (which means that tree walking transducer M type checks w.r.t. $\mathcal{L}(A_I)$ and $\mathcal{L}(A)$) in time linear in $|C|$.

Theorem 11.7. *Deciding whether a strictly b-bounded 2TT M type checks w.r.t. regular tree languages I and O, given by BTAs A_I and A_O, is polynomial in the size of M, A_I, and A_O, but exponential in $b^2 \cdot (d+1)$ where d is the maximal number of occurrences of state calls in right-hand sides. If M has no up-operations, the exponent can be improved to $b \cdot (d+1)$.*

11.5 Notes and References

Our definition of ATWAs is equivalent to the alternating two-way finite tree automata of [Slu85]. Note that alternating tree automata have recently been used in the context of a practical implementation of type checking for tree transducers [FH07].

The intersection of a 2TT with a given output type (Theorem 11.1) can be seen as a sequential composition of the 2TT with a translation in FTA; the latter is the class of partial identity mappings for regular tree languages. With this in mind, we can, for instance, obtain that top-down tree transducers allow a similar result as the one in Theorem 11.1: By Corollary 2(1) of [Bak79], top-down tree transducers are closed under composition with linear and nondeleting top-down tree transducers; since FTA is included in the latter class, we obtain the desired result for top-down tree transducers. It is an interesting open problem whether a similar composition result holds for 2TTs, i.e., whether 2TTs are closed under composition with linear and nondeleting 2TTs.

The notion of b-boundedness is similar to the notion of finite-copying in tree transducers, see, e.g., [ERS80, EM99]. Similar to the results of [EM03c], it probably holds that b-bounded transformations are of linear size increase. A more static version of b-boundedness is the single-use restriction known for attribute grammars [Gie88]. According to [EM99], it can probably be shown that total, deterministic, strictly b-bounded 2TTs are equivalent to single-use restricted attribute grammars. In Section 5 of [MPS07], a similar result as Theorem 11.7 has been shown for stay-macro tree transducers (cf. also discussion in Section 12.4)

Engelfriet et al. show in [EHS07] (Theorem 5) that for every 2TT M (TT in [EHS07]), a regular tree grammar G can be constructed in exponential time

such that G generates the domain of τ_M. They refer to the relationship between 2TTs and attributed tree transducers explained in [EM03b] and a result of [Bar82] — giving the Theorem 11.5 above. The result of Theorem 11.5 has also been stated in Theorem A.2 of [CGKV88] with a proof sketch that uses a game theoretic interpretation of acceptance due to Muller and Schupp [MS87]. The result also appears as Theorem 1 in [Eng09], where a finite state automaton for the domain of a tree-walking tree transducer (twtt) is constructed in exponential time; Theorem 2 of that paper states that inverse type inference is in k-fold exponential time, for k-fold compositions of twtts.

Chapter 12

Macro Tree Walking Transducers

In our current example, we have considered the 2TT M_{staff} which lists the staff members of a department. Although in general, several employees may have the same boss, the transducer spawns for every employee a separate computation to determine the corresponding boss. Conceptually as well as technically, it would be more convenient to determine the boss first, store it in some accumulating parameter and then propagate it to each of his employees. For this reason, we enhance tree transducers with accumulating parameters. A tree transducer with accumulating parameters is also called *macro tree transducer*.

Example 12.1. We omit the state q_{boss} and store the data of the boss in the first parameter y_b. The transformation of the next employee, which is not a subordinate of the current, is then stored in the second parameter (y_n). By this construction, we completely omit the states q_{sub}, q_{next}, and q_{up}. The transducer consists of the following rules, for all $\eta \in \{1, 2\}$:

$$
\begin{aligned}
&q_I(\texttt{department}, 0) && \to \texttt{staff}(q(down_1, \texttt{e}, \texttt{e}), \texttt{e})\\
&q(\texttt{employee}, \eta, y_b, y_n) && \to \texttt{employee}(\texttt{data}(q_{data}(down_1), y_b),\\
&&& \qquad q(down_1, \texttt{boss}(q_{data}(down_1), \texttt{e}),\\
&&& \qquad\qquad q(down_2, y_b, y_n)))\\
&q(\texttt{e}, \eta, y_b, y_n) && \to y_n\\
&q(\texttt{data}, 1, y_b, y_n) && \to q(down_2, y_b, y_n)\\
&q(\texttt{subordinates}, 2, y_b, y_n) && \to q(down_1, y_b, y_n)\\
&q_{data}(\texttt{data}, 1) && \to copy(down_1)
\end{aligned}
$$

Here state *copy* is meant to copy the content of `data` (i.e., the left child in the binary representation). ◁

For the formal definition of macro tree walking transducer, we assume that every

state $q \in Q$ has a fixed rank, i.e., $Q = \biguplus_{n\in\mathbb{N}} Q^{(n)}$ where $Q^{(n)}$ is the set of all states with rank n.

Definition 12.1 (2MTT)**.** A *macro tree walking transducer* M (2MTT for short) is a tuple (Q, Σ, R, Q_0) where Q is a set of ranked states, Σ is a ranked alphabet, $Q_0 \subseteq Q^{(1)}$ is a set of initial states, and R is a finite set of rules of the form $q(\mathtt{a}, \eta, y_1, \dots y_n) \to \zeta$, where $q \in Q^{(n+1)}$, $\mathtt{a} \in \Sigma^{(m)}$, $n, m \geq 0$, $\eta \geq 0$ is a direction and $y_1, \dots, y_n$ are the accumulating parameters of q. Possible right-hand sides are described by the grammar

$$\zeta \ ::= \ \mathtt{b}(\overbrace{\zeta, \dots, \zeta}^{m' \text{ times}}) \mid y_j \mid q'(op, \overbrace{\zeta, \dots, \zeta}^{n' \text{ times}}),$$

with $m', n' \geq 0$, symbol $\mathtt{b} \in \Sigma^{(m')}$, $j \in [n]$, state $q' \in Q^{(n'+1)}$, and operation $op \in \{stay, up\} \cup \{down_i \mid i \in [m]\}$.

In practice, states q may differ in their *rank*, i.e., the numbers of their accumulating parameters. Let $X = \{x_1, x_2, \dots\}$ denote a countable set of variables of rank (not necessarily 0), and assume that Σ, X and Y are disjoint. For a right-hand side ζ, we write also

$$\zeta = z[q_1(op_1), \dots, q_c(op_c)]$$

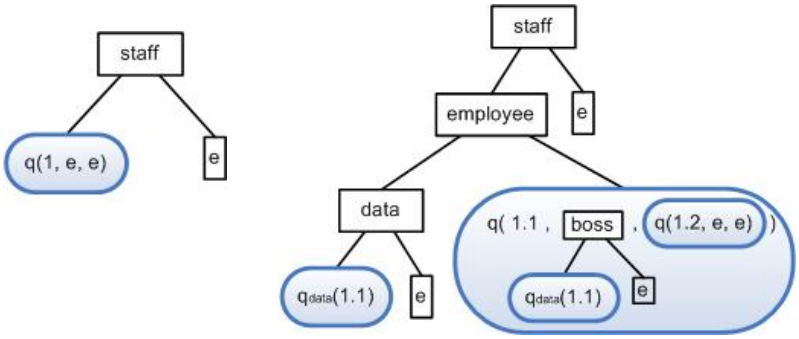
to refer to all occurrences of (maybe nested) state calls in the right-hand side. Here, $z \in \mathcal{T}_{\Sigma\cup X}(Y)$ is a tree, which contains each variable $x_1, \dots, x_c$ exactly once with $\zeta = z[q_1(op_1)/x_1, \dots, q_c(op_c)/x_c]$ where $z[q_i(op)/x_i]$ denotes the substitution of the state call $q_i(op, z_1, \dots, z_n)$ for the subtree $x_i(z_1, \dots, z_n)$ in z where n is the rank of x_i and $n+1$ is the rank of q_i. Note that in z no state call occurs anymore. The right-hand side of the rule in Lines 2-4 in the Example 12.1 can, for example, be written as

$$z[q_{data}(down_1), q(down_1), q_{data}(down_1), q(down_2)]$$

where $z = \mathtt{employee}(\mathtt{data}(x_1, y_b), x_2(\mathtt{boss}(x_3, \mathtt{e}), x_4(y_b, y_n)))$.

Intuitively, the meaning of the expressions of a right-hand side is as follows: The produced output is defined analogously to the output of a 2TT up to the accumulating parameters. Here, we consider *call-by-value* parameter passing only. Thus, the expression ζ_j in parameter position j is evaluated first; then the result (which is a tree without state calls) may be copied to the various uses of the formal parameter y_j. This evaluation strategy is also called *inside-out* (IO for short). Note that we slightly abuse Definition 12.1 and use accumulating parameters with names other than $y_1, y_2, \dots$ (e.g., in Example 12.1 where we use y_b and y_n). Clearly, this is without loss of generality, as parameters can easily be renamed according to the definition.

Example 12.2. The rules in the beginning of this chapter with the initial state q_I form a 2MTT $M_{y,\text{staff}}$. To transform the tree $u_B{}'$ (the binary representation of our common example tree, see the left tree in Figure 9.1), the macro



(a) The first step: Analyzing $q_I(\epsilon)$

(b) and the second step: Analyzing $q(1, e, e)$.

Figure 12.1: Applying the 2MTT $M_{y,\text{staff}}$ to u_B.

tree walking transducer starts at the root and applies the initial state. Thus, for the first step, we get $\texttt{staff}(q(down_1, \texttt{e}, \texttt{e}), \texttt{e})$ where $down_1$ refers to the node 1. Applying the state q to this `employee`-node, we get several state calls. These state calls are partially nested (Figure 12.1). In the left figure is the first output with one state call. The right figure shows the tree after processing $q(1, \texttt{e}, \texttt{e})$. There, we get the state call

$$q(down_1, \texttt{boss}(q_{data}(down_1), \texttt{e}), q(down_2, y_b, y_n))$$

with nested calls. The first parameter accumulates a tree with root `boss`, whereas the second parameter is a further state call. ◁

The order in which nested state calls are evaluated indeed matters. Consider, e.g., a transducer with the rules

$$\begin{aligned} q_I(\texttt{a}, 0) &\to p(stay, q'(stay)) \\ p(\texttt{a}, 0, y) &\to \texttt{a} \\ q'(\texttt{b}, 0) &\to \texttt{b} \,. \end{aligned}$$

If we evaluate the outermost calls first, the tree $u = \texttt{a}(u_1, \ldots, u_k)$ will be transformed into `a`. In this case, the accumulating parameter of p need not to be evaluated. If we start with the innermost calls, there is no rule to evaluate the state call $q'(stay)$ in the right-hand side of the first rule. Thus, the output is empty.

We specify the translation induced by a 2MTT using a denotational formulation. Later, we will also consider an operational semantics based on runs. In the denotational semantics, the meaning $[\![q]\!]_u$ of state q of transducer M with n accumulating parameters (with respect to an input tree u) is defined as a mapping from nodes in the input tree to sets of trees with parameters in $Y_n = \{y_1, \ldots, y_n\}$, i.e.,

$$[\![q]\!]_u : \texttt{Nodes}(u) \to 2^{\mathcal{T}_\Sigma(Y)} \,.$$

When we evaluate an innermost call $q(\vartheta, v_1, \ldots, v_n)$ on a node ϑ during a computation, it suffices to substitute actual parameters v_j for the formal parameters y_j of all terms from $[\![q]\!]_u(\vartheta)$ to obtain the set of produced outputs. The values $[\![q]\!]_u$ for all q are jointly defined as the least mappings satisfying:

$$[\![q]\!]_u(\vartheta) \supseteq [\![\zeta]\!]_u \quad \text{for rule } q(\mathtt{a}, \eta, \bar{y}) \to \zeta$$

where $\bar{y}$ denotes the sequence $y_1, \ldots, y_n$ of parameters and ϑ is a node of u with $lab(\vartheta) = \mathtt{a}$, $\eta(\vartheta) = \eta$, and $[\![\zeta]\!]_u$ is defined by:

$$\begin{aligned}
[\![y_j]\!]_u &= \{y_j\} \\
[\![\mathtt{b}(\zeta_1, \ldots, \zeta_m)]\!]_u &= \{\mathtt{b}(v_1, \ldots, v_m) \mid v_i \in [\![\zeta_i]\!]_u\} \\
[\![q'(op, \zeta_1, \ldots, \zeta_{n'})]\!]_u &= \{z[v_1/y_1, \ldots, v_{n'}/y_{n'}] \mid z \in [\![q']\!]_u([\![op]\!]_u(\vartheta)), v_i \in [\![\zeta_i]\!]_u\}
\end{aligned}$$

Again, op stands for $down_i$, $stay$ or up. Recall that the meaning $[\![op]\!]$ is defined by

$$[\![stay]\!]_u(\vartheta) = \vartheta, \quad [\![down_i]\!]_u(\vartheta) = \vartheta i, \quad \text{and} \quad [\![up]\!]_u(\vartheta i) = \vartheta \; .$$

Also, $z[v_1/y_1, \ldots, v_n/y_n]$ denotes the simultaneous substitution of the trees v_j for all occurrences of the variables y_j in the tree z. Note that the call-by-value semantics is reflected in the last equation: The same trees v_j are used for all occurrences of a variable y_j in the tree z corresponding to a potential evaluation of the state q'. The transformation τ_M, realized by the 2MTT M on an input tree u and sets U of input trees, respectively, is, thus, defined by

$$\tau_M(u) = \bigcup\{[\![q_0]\!]_u(\epsilon) \mid q_0 \in Q_0\} \quad \text{and} \quad \tau_M(U) = \bigcup\{\tau_M(u) \mid u \in U\} \; .$$

For the operational semantics, runs of a 2MTT M on a tree u may be similarly defined as for a 2TT. It is a ranked tree over the set of rules. Here, the rank of a rule $q(\mathtt{a}, \eta, y_1, \ldots, y_n) \to \zeta$ is given by the number of occurrences of recursive calls $q'(op)$ in ζ to states q' in Q. These calls may be nested. Figure 12.2 shows an accepting run ρ of the (deterministic) example 2MTT $M_{y,\text{staff}}$ on the tree $u_e = \mathtt{department}(\mathtt{employee}(\mathtt{data}(\ldots, \mathtt{e}), \mathtt{e}), \mathtt{e})$, which describes a department with one employee.

The denotational view on the semantics of a 2MTT allows us to use fixpoint arguments for proving the correctness of constructions, whereas the operational view is better suited for more combinatorial arguments. In particular, we can show that the number of occurrences of states in right-hand sides can be restricted to the maximum of the ranks of output symbols and states. We have:

Lemma 12.1. *For every 2MTT M, there exists a 2MTT M' with*

1. $\tau_{M'} = \tau_M$
2. *the number of states occurring in each right-hand side is bounded by k*
3. $|M'| \in \mathcal{O}(|M| \cdot k^2)$

where k is the maximum of the ranks of output symbols and states of M.

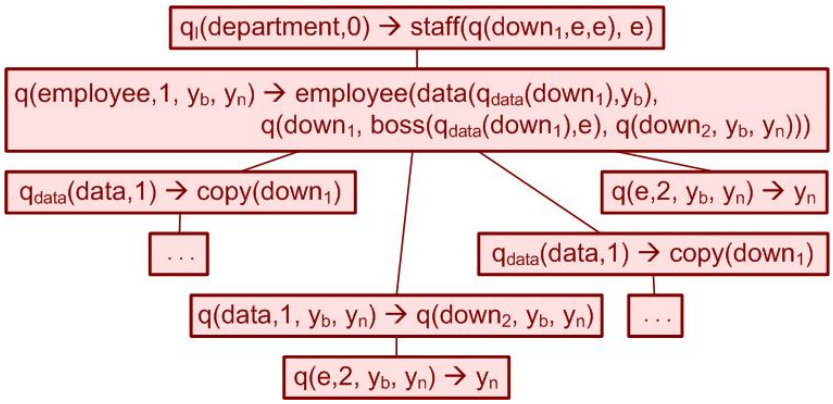


Figure 12.2: A run of the 2MTT $M_{y,\text{staff}}$ on the tree u_e.

Proof. The construction proceeds in two phases. In the first phase, we replace every *complicated* call $q'(op, \zeta_1, \ldots, \zeta_n)$ in the right-hand side of a rule $q(\mathtt{a}, \eta, y_1, \ldots, y_{n'}) \to \zeta$ by the simple call

$$\langle q, op, \zeta_1, \ldots, \zeta_n \rangle(stay, y_1, \ldots, y_{n'})$$

for a new state $\langle q, op, \zeta_1, \ldots, \zeta_n \rangle$. Let $[\zeta]$ denote the resulting tree. For the new state $\langle q, op, \zeta_1, \ldots, \zeta_n \rangle$, we introduce the rule

$$\begin{aligned}&\langle q, op, \zeta_1, \ldots, \zeta_n \rangle(\mathtt{a}, \eta, y_1, \ldots, y_{n'}) \\ &\quad \to q(op, \langle \zeta_1, n' \rangle(stay, y_1, \ldots, y_{n'}), \ldots, \langle \zeta_n, n' \rangle(stay, y_1, \ldots, y_{n'}))\end{aligned}$$

for again fresh states $\langle \zeta_j, n' \rangle$, which are meant to produce the output of ζ_j using n' parameters. For these states, we introduce the rules:

$$\langle \zeta_j, n' \rangle(\mathtt{a}, \eta, y_1, \ldots, y_{n'}) \to [\zeta_j] \,.$$

As a result of this first transformation phase, we achieve that all right-hand sides either are of the form $q(op, q_1(stay, y_1, \ldots, y_{n'}), \ldots, q_n(stay, y_1, \ldots, y_{n'}))$ or contain only non-nested calls, i.e., calls of the form $q(op, y_1, \ldots, y_{n'})$. In order to restrict the number of calls in right-hand sides of the second type, we essentially proceed as in the proof of Lemma 9.1, i.e., we introduce extra auxiliary states for every proper subtree of right-hand sides of the second kind, which contain more than one call.

The resulting transducer has at most one fresh state for every node of a right-hand side while the total sum of sizes of right-hand sides, may increase by a factor of k^2 in order to spell out all the auxiliary lists of parameters for the new states. □

12.1 Type Checking 2MTTs

As for 2TTs, we now consider type checking for *macro* tree walking transducer. For a 2MTT M and a regular language $\mathcal{L}$, we again construct a transducer M' with $\tau_{M'}(t) = \tau_M(t) \cap \mathcal{L}$. As in Section 11.1, the language $\mathcal{L}$ consists of all erroneous outputs. In our application scenario of type checking, the language $\mathcal{L}$ is the complement of the output type which is either specified by a complete DBTA or by a DTTA. Beyond the case of 2TTs, we now additionally have to deal with accumulating parameters. The macro tree walking transducer M' has to keep track of the states of an automaton for $\mathcal{L}$ on the current values of the respective parameters. We start with a general construction for deterministic bottom-up automata.

Theorem 12.2. *For every 2MTT M and every DBTA A, a 2MTT M_A can be constructed with*

$$\tau_{M_A}(u) = \tau_M(u) \cap \mathcal{L}(A)$$

for all $u \in \mathcal{T}_\Sigma$. The 2MTT M_A is of size $\mathcal{O}(|M| \cdot |A|^{l\cdot(d+1)})$ where l is the maximal rank of a state in M and d is the maximal number of occurrences of states in right-hand sides of M.

Proof. Let $M = (Q, \Sigma, R, Q_0)$ and $A = (P, \Sigma, \delta, F)$. For each state q in $Q^{(n+1)}$ and all states $p_0, \ldots, p_n \in P$, the 2MTT M_A has a state $\langle q, p_0 p_1 \ldots p_n\rangle$, which is meant to generate all trees z (possibly with variables from $\{y_1, \ldots, y_n\}$) that could be produced by M, and for which, additionally, there is a run of A starting at the leaves y_j with state p_j, and reaching the root of z in state p_0. The rules of M_A are:

$$\langle q, p_0 p_1 \ldots p_n\rangle(\mathtt{a}, \eta, y_1, \ldots, y_n) \to \zeta'$$

for every rule $q(\mathtt{a}, \eta, y_1, \ldots, y_n) \to \zeta$ of M and $\zeta' \in \tau^{p_0,p_1,\ldots,p_n}[\zeta]$ where the sets $\tau^{p_0,p_1,\ldots,p_n}[.]$ are inductively defined by:

$$\begin{aligned}
\tau^{p_j,p_1,\ldots,p_n}[y_j] &= \{y_j\} \\
\tau^{p_0,p_1,\ldots,p_n}[\mathtt{b}(\zeta_1, \ldots, \zeta_m)] &= \{\mathtt{b}(\zeta'_1, \ldots, \zeta'_m) \mid \exists p'_1, \ldots, p'_m \in P : \\
&\qquad (p_0, \mathtt{b}, p'_1 \ldots p'_m) \in \delta \ \wedge \forall j : \zeta'_j \in \tau^{p'_j p_1 \ldots p_n}[\zeta_j]\} \\
\tau^{p_0,p_1,\ldots,p_n}[q'(op, \zeta_1, \ldots, \zeta_{n'})] &= \{\langle q', p_0 p'_1 \ldots p'_{n'}\rangle(op, \zeta'_1, \ldots, \zeta'_{n'}) \mid \\
&\qquad \exists \zeta'_1, \ldots, \zeta'_{n'} \in Z : \ \forall j : \zeta'_j \in \tau^{p'_j p_1 \ldots p_n}[\zeta_j]\}
\end{aligned}$$

where Z denotes the set of all subterms of possible right-hand sides of rules of M_A. The set of initial states of M_A is given by $Q'_0 = Q_0 \times F$. By fixpoint induction, we verify for every state q of rank $n+1$, every input tree $u \in \mathcal{T}_\Sigma$, every node $\vartheta \in \mathtt{Nodes}(u)$ and states $p_0, \ldots, p_n$ of A that

$$[\![\langle q, p_0, \ldots, p_n\rangle]\!]_u(\vartheta) = [\![q]\!]_u(\vartheta) \cap \{z \in \mathcal{T}_\Sigma(Y) \mid \delta^*(z, p_1 \ldots p_n) = p_0\} \qquad (12.1)$$

where $Y = \{y_1, \ldots, y_n\}$ and δ^* is the extension of the transition function of A to trees containing variables from Y, namely, for $\bar{p} = p_1 \ldots p_n$:

$$\delta^*(y_j, \bar{p}) = p_j$$
$$\delta^*(\mathtt{a}(t_1, \ldots, t_m), \bar{p}) = \delta(\mathtt{a}, \delta^*(t_1, \bar{p}) \ldots \delta^*(t_m, \bar{p}))$$

The correctness of the construction follows from Equation 12.1.

For each state in M, we have at most $|A|^l$ new states in M_A where l is the maximal rank of states in M. Assume that d is the maximal number of occurrences of states in right-hand sides of rules of M. Then each rule of M gives rise to at most $|A|^{l\cdot(d+1)}$ new rules in M_A. Therefore, the new 2MTT is of size $\mathcal{O}(|M| \cdot |A|^{l\cdot(d+1)})$. □

Lemma 10.1 provides a BTA describing the complement of a DTTA — which, however, is not necessarily deterministic. Theorem 12.2, on the other hand, only holds for deterministic BTAs. A similar construction is also possible if the BTA A is nondeterministic — but then only for transducers, which are output-*linear*. Here, we call a 2MTT output-linear if every accumulating parameter occurs at most once in a right-hand side. Nonetheless, we are able to handle complements of output types described by DTTAs directly. For that, however, we introduce a dedicated construction of a 2MTT M_A.

Theorem 12.3. *For every 2MTT M and every DTTA A, a 2MTT $M_{\bar{A}}$ can be constructed with*

$$\tau_{M_{\bar{A}}}(u) = \tau_M(u) \cap \overline{\mathcal{L}(A)}$$

for all $u \in \mathcal{T}_\Sigma$. The 2MTT $M_{\bar{A}}$ is of size $\mathcal{O}(|M| \cdot (h \cdot |A|)^{d+2})$ where $h+1$ is the maximum of the maximal rank of a state in M and the maximal rank of an output symbol, and d is the maximal number of occurrences of state calls in right-hand sides of M.

Proof. Let $M = (Q, \Sigma, R, Q_0)$ and $A = (P, \Sigma, \delta, p_0)$. Our goal is to construct a 2MTT $M_{\bar{A}}$, which simulates the behavior of M, while at the same time guessing a path in the output tree, which proves non-containment in the set $\mathcal{L}(A)$. For that, the set Q' is defined as:

$$Q' = Q \cup \{\langle q, p\rangle \mid q \in Q, p \in P\}$$
$$\cup \{\langle q, p, j, p'\rangle \mid q \in Q^{(n)}, p, p' \in P, j \in [n]\}$$

Here, a state $q \in Q$ of $M_{\bar{A}}$ behaves like the state q of M. States $\langle q, p\rangle$ or $\langle q, p, j, p'\rangle$ behave like q in M but, additionally, make sure that there is a path in the generated output starting from a state p of A at the root, which verifies that there is no p-run of A on the output. Thereby, a state $\langle q, p\rangle$ will directly generate the end point of such a path whereas state $\langle q, p, j, p'\rangle$ will only generate a path with p at the root reaching a parameter y_j with state p'. Accordingly, the 2MTT $M_{\bar{A}}$ has the following rules:

$$q(\mathtt{a},\eta,y_1,\ldots,y_n)\to\zeta$$
$$\langle q,p\rangle(\mathtt{a},\eta,y_1,\ldots,y_n)\to\zeta' \text{ with } \zeta'\in[\zeta]^p$$
$$\langle q,p,j,p'\rangle(\mathtt{a},\eta,y_1,\ldots,y_n)\to\zeta' \text{ with } \zeta'\in[\zeta]^{p,j,p'}$$

for every rule $q(\mathtt{a},\eta,y_1,\ldots,y_n)\to\zeta$ of M. The sets $[.]^x$ are inductively defined in Figure 12.3.

First, we verify that for every tree $z\in\mathcal{T}_\Sigma(Y)$, the sets $[z]^p$ and $[z]^{p,j',p'}$ either are empty or equal $\{z\}$ where following holds:

1. $[z]^p=\{z\}$ iff z contains a node $\vartheta=i_1\ldots i_r$ such that there are transitions $(p_0^{(j)},\mathtt{a}_j,p_1^{(j)}\ldots p_{m_j}^{(j)})\in\delta$ for every $j\in[r-1]$, such that
   (a) the label of the node $i_1\ldots i_j$ equals $\mathtt{a}_j$;
   (b) $p_0^{(1)}=p$ and for every $j\in[r-2]$, $p_0^{(j+1)}=p_{i_j}^{(j)}$;
   (c) there is no $p_{i_r}^{(r-1)}$-transition of A for $\mathtt{a}_r=lab(\vartheta)$.
2. $z\in[z]^{p,j',p'}$ iff z contains a node $\vartheta=i_1\ldots i_{r+1}$, which is labeled with $y_{j'}$ and there are transitions $(p_0^{(j)},\mathtt{a}_j,p_1^{(j)}\ldots p_{m_j}^{(j)})\in\delta$ for every $j\in[r]$, such that:
   (a) For every $j=1,\ldots,r$, the label of the node $i_1\ldots i_j$ equals $\mathtt{a}_j$;
   (b) $p_0^{(1)}=p$ and $p_{i_{r+1}}^{(r)}=p'$ and for every $j\in[r-1]$, $p_0^{(j+1)}=p_{i_j}^{(j)}$.

Note in particular that by this definition, $z\notin\mathcal{L}(A)$ iff $[z]^{p_0}=\{z\}$ for the initial state p_0 of A. Let us extend the operators $[\,.\,]^p$ and $[\,.\,]^{p,j',p'}$ by:

$$[Z]^p=\bigcup\{[z]^p\mid z\in Z\}\qquad\qquad [Z]^{p,j',p'}=\bigcup\{[z]^{p,j',p'}\mid z\in Z\}$$

for $Z\subseteq\mathcal{T}_\Sigma(Y)$ with $Y=\{y_1,\ldots,y_n\}$. By fixpoint induction, we verify for every state q of rank $n+1$, every $j\in[n]$, every input tree $u\in\mathcal{T}_\Sigma$ and states $p,p'\in P$ that:

$$[\![\langle q,p,j,p'\rangle]\!](u)=[\,[\![q]\!](u)\,]^{p,j,p'}$$
$$[\![\langle q,p\rangle]\!](u)=[\,[\![q]\!](u)\,]^{p}$$

For each state p and right-hand side ζ of a rule, we assign states of the deterministic top-down tree automaton to the nodes of the tree. Either we immediately hit a node certifying the non-existence of a p-run of the DTTA on the output generated from ζ, or we hit an occurrence of a state call $q(op,\ldots)$. If $n\geq 1$ is the rank of q, we have n choices here: Either we expect a certificate for the failure of the DTTA A inside the evaluation of q or in one of the parameters of q. Overall, we find that every rule of M, thus, gives rise to $(h\cdot|A|)^{d+2}$ rules. Thus, we have $|M_{\bar{A}}|\in\mathcal{O}(|M|\cdot(h\cdot|A|)^{d+2})$. □

Assume that we have a binary ranked alphabet and that at least one state has an accumulating parameter, i.e., the maximal rank l of states is at least 2. By

$$
\begin{aligned}
[y_i]^p &= \emptyset \\
[\mathtt{b}(\zeta_1,\ldots,\zeta_m)]^p &= \{\mathtt{b}(\zeta_1,\ldots,\zeta_m) \mid \forall \bar{p} \in P^m :\ (p,\mathtt{b},\bar{p}) \notin \delta\} \\
&\quad \cup \{\mathtt{b}(\zeta_1,\ldots,\zeta_{\nu-1},\zeta'_\nu,\zeta_{\nu+1},\ldots,\zeta_m) \mid \nu \geq 1, (p,\mathtt{b},p_1\ldots p_m) \in \delta,\ \zeta'_\nu \in [\zeta_\nu]^{p_\nu}\} \\
[q(op,\zeta_1,\ldots,\zeta_n)]^p &= \{\langle q,p,\nu,p_\nu\rangle(op,\zeta_1,\ldots,\zeta_{\nu-1},\zeta'_\nu,\zeta_{\nu+1},\ldots,\zeta_n) \mid \zeta'_\nu \in [\zeta_\nu]^{p_\nu}\} \\
&\quad \cup \{\langle q,p\rangle(op,\zeta_1,\ldots,\zeta_n)\}
\end{aligned}
$$

$$
\begin{aligned}
[y_j]^{p,j,p'} &= \{y_j \mid p = p'\} \\
[\mathtt{b}(\zeta_1,\ldots,\zeta_m)]^{p,j,p'} &= \{\mathtt{b}(\zeta_1,\ldots,\zeta_{\nu-1},\zeta'_\nu,\zeta_{\nu+1},\ldots,\zeta_m) \mid \nu \geq 1, (p,\mathtt{b},p_1\ldots p_m) \in \delta,\ \zeta'_\nu \in [\zeta_\nu]^{p_\nu,j,p'}\} \\
[q(op,\zeta_1,\ldots,\zeta_n)]^{p,j,p'} &= \{\langle q,p,\nu,p_\nu\rangle(op,\zeta_1,\ldots,\zeta_{\nu-1},\zeta'_\nu,\zeta_{\nu+1},\ldots,\zeta_n) \mid \zeta'_\nu \in [\zeta_\nu]^{p_\nu,j,p'}\}
\end{aligned}
$$

Figure 12.3: The definition of the sets $[.]^x$ in the proof of Theorem 12.3.

Lemma 12.1 it is then possible to restrict the number of occurrences of state calls in right-hand sides of the 2MTT to l. This implies that the size of the intersection 2MTT M_A in Theorem 12.2 for a DBTA describing the output language is in $\mathcal{O}(|M| \cdot |A|^{l\cdot(l+1)})$. Furthermore, the size of the 2MTT $M_{\bar{A}}$ in Theorem 12.3 for a deterministic top-down tree automaton describing the output language is in $\mathcal{O}(|M| \cdot (l \cdot |A|)^{l+2})$.

12.2 Deciding Emptiness of 2MTTs

To decide emptiness of a 2MTT M, we follow the approach taken for 2TTs: We construct an alternating tree walking automaton A_M, which is then tested for emptiness. The ATWA A_M has the same set of states as M (but they are not ranked anymore now) where the initial states of M and A_M coincide. For every rule $q(\mathtt{a}, \eta, y_1, \ldots, y_n) \to \zeta$ of the 2MTT M, the ATWA A_M has a rule of the form

$$q(\mathtt{a}, \eta) \to q_1(op_1) \wedge \ldots \wedge q_c(op_c)$$

if $q_1(op_1, \ldots), \ldots, q_c(op_c, \ldots)$ is the sequence of calls to states of M (possibly nested inside each other), in any order. Since we use 2MTTs with call-by-value semantics, M has an accepting run on some input tree u iff A_M has an accepting run on u. Note that this construction is wrong for call-by-need semantics because M could have an accepting run on a tree $u \notin \mathcal{L}(A_M)$; for instance, the 2MTT with the rules $q_I(\mathtt{a}, 0) \to q(stay, q'(stay))$ and $q(\mathtt{a}, 0, y) \to \mathtt{a}$ on the tree $\mathtt{a}$.

Theorem 12.4. *For every 2MTT M, an ATWA A_M can be constructed in polynomial time such that $\mathcal{L}(A_M) = dom(M)$. Thus, it can be decided in deterministic exponential time whether the translation of a 2MTT is empty or not.*

12.3 Input-Linear 2MTTs

The notions of b-boundedness and strict b-boundedness, which we have defined for 2TTs stay meaningful also in presence of accumulating parameters, i.e., for 2MTTs. Analogously, we find that emptiness for b-bounded macro tree walking transducers is decidable in polynomial time — independent on the number of accumulating parameters of states.

In order to identify classes of 2MTTs where full type checking is tractable, we therefore take a closer look at the construction for the intersection of 2MTTs with (complements of) output types. For simplicity, we first consider macro tree walking transducers, which are *strictly 1-bounded*. This notion is only meaningful for *top-down* MTTs, i.e., 2MTTs without operations *up* or *stay*. A top-down transducer M is guaranteed to visit each node of the input tree at most once if for the same i, the operation $down_i$ does not occur twice in the same right-hand side of M. This property can easily be checked syntactically. Tree transducers satisfying this restriction are called *input-linear*.

Note that input-linearity for a tree transducer implies that the number of state calls in right-hand sides is bounded by the maximal rank of input symbols. Moreover, the output language can be described by rules that are obtained by simply deleting all directives from the transducer's rules. The resulting rules no longer specify a transformation but constitute a *context-free tree grammar* (short: CFTG) for generating output trees.

Example 12.3. As an example of an input-linear MTT consider the following macro tree transducer, which produces the same output as the common example 2MTT $M_{y,\text{staff}}$ without the `boss`-subtrees. The transducer only needs one parameter (for the next employee) and has a new state q_{empl}. For $\eta \in \{1, 2\}$, it has the rules:

$$
\begin{aligned}
&1 \quad q_I(\texttt{department}, 0) && \to \texttt{staff}(q(down_1, \texttt{e}), \texttt{e})\\
&2 \quad q(\texttt{employee}, \eta, y_n) && \to q_{empl}(down_1, q(down_2, y_n))\\
&3 \quad q_{empl}(\texttt{data}, 1, y_n) && \to \texttt{employee}(\texttt{data}(q_{data}(down_1), \texttt{e}), q(down_2, y_n))\\
&4 \quad q(\texttt{subordinates}, 2, y_n) && \to q(down_1, y_n)\\
&5 \quad q(\texttt{e}, \eta, y_n) && \to y_n\\
&6 \quad q_{data}(\texttt{data}, 1) && \to copy(down_1).
\end{aligned}
$$

The grammar characterizing its output language looks as follows:

$$
\begin{aligned}
&1 \quad q_I && \to \texttt{staff}(q(\texttt{e}), \texttt{e})\\
&2 \quad q(y_n) && \to q_{empl}(q(y_n)) \mid q(y_n) \mid y_n\\
&3 \quad q_{empl}(y_n) && \to \texttt{employee}(\texttt{data}(q_{data}, \texttt{e}), q(y_n))\\
&4 \quad q_{data} && \to copy
\end{aligned}
$$

where q_I, q, q_{empl}, q_{data}, and $copy$ are nonterminals. Selection of rules depending on input symbols and directions now has been replaced with nondeterministic choice. ◁

Context-free tree grammars generalize context-free grammars to trees. Formally, a CFTG G can be represented by a tuple (E, Σ, P, E_0) where

- E is a finite ranked set of function symbols or nonterminals,
- $E_0 \subseteq E$ is a set of initial symbols of rank 0,
- Σ is the ranked alphabet of terminal nodes, and
- P is a set of rules of the form $q(y_1, \ldots, y_n) \to \zeta$ where $q \in E$ is a nonterminal of rank $n \geq 0$. The right-hand side ζ is a tree built up from variables $y_1, \ldots, y_n$ by means of application of nonterminal and terminal symbols.

As for 2MTTs, inside-out (IO) and outside-in evaluation order for nonterminal symbols must be distinguished. Here, we use the IO or call-by-value evaluation

order. The least fixpoint semantics for the CFTG G is obtained straightforwardly along the lines for 2MTTs — simply by removing the corresponding directive components, i.e., by removing in the last line of the definition of $[\![\zeta]\!]$ for 2MTTs (Page 171) the *op* and $[\![op]\!]_t(v)$. In particular, this semantics assigns to every nonterminal q of rank $n \geq 0$, a set $[\![q]\!] \subseteq \mathcal{T}_\Sigma(Y)$ for $Y = \{y_1, \ldots, y_n\}$. The language generated by G is $\mathcal{L}(G) = \bigcup\{[\![q_0]\!] \mid q_0 \in E_0\}$.

It is easy to see that the output language of an input-linear MTT M can be characterized by a CFTG G_M, which can be constructed from M in linear time. During this construction, every rule $q(\mathtt{a}, \eta, y_1, \ldots, y_n) \to \zeta$ is rewritten as a production $q(y_1, \ldots, y_n) \to \zeta'$ where ζ' is obtained from ζ by deleting all occurrences of navigation operators.

The characterization of output languages for input-linear MTTs by CFTGs is useful because emptiness for (IO-)CFTGs is decidable using a similar algorithm as the one for ordinary context-free (word) grammars, and hence, can be done in linear time.

Theorem 12.5. *It can be decided in linear time for a CFTG G whether or not $\mathcal{L}(G) = \emptyset$.*

Here, we are interested in testing whether a given input-linear MTT M type checks w.r.t. input and output types I and O. Assume that I is given by a (possibly non-deterministic) BTA B with only productive states, i.e., for every state p of B, there exists a p-run of B on a tree. As a first step, we construct a new input-linear MTT M_B such that M_B's range, i.e., the set $\tau_{M_B}(\mathcal{T}_\Sigma)$, is equal to $\tau_M(I)$. This is done by a straightforward product construction of the BTA B and the input-linear MTT M. Note that it may happen that M does not visit a certain subtree u of the input tree. In such a case, the checking of u w.r.t. B cannot be done by the new transducer M_B. This does not affect the corresponding output language though. We now construct the intersection 2MTT for M_B and the complement of the output type O. In case, O is given by a DBTA, this can be done along the lines of the proof of Theorem 12.2. If O is given by a DTTA, we rely on the construction from Theorem 12.3. Since M is input-linear, the intersection 2MTT is again input-linear — meaning that its range can be described by a CFTG (thus generating all "illegal outputs" of M w.r.t. I and O). Therefore, Theorem 12.5 gives us:

Theorem 12.6. *Assume M is an input-linear MTT where the ranks of input symbols are bounded, and let I and O denote input and output types for M where I is given by a BTA.*

1. *Assume that the output type O is specified with a DBTA and the maximal rank of states of M is bounded. Then M can be type checked relative to I and O in polynomial time.*

2. *Assume that the output type O is specified with a DTTA. Then M can be type checked relative to I and O in polynomial time — even in presence of unbounded ranks of states.*

The worst-case complexity bounds for the construction of Theorem 12.6 are exponential in $l \cdot (k+1)$ (for output types given through DBTAs) or $k+2$ (for output types given through DTTAs) where l is the maximal rank of states and k is the maximal rank of an input symbol of M. In practical applications, both k and l may be moderately small. Still, we want to point out that in case of input-linear MTTs, the intersection construction can be organized in such a way that only "useful" states are constructed. In order to see this, consider again an input-linear MTT M and a DBTA A (representing the incorrect output trees). The idea is to introduce for every q of M of rank $n+1$, a *Datalog* predicate $q/n+1$. Every rule $q(\mathtt{a}, \eta, y_1, \ldots, y_n) \to \zeta$ of M then gives rise to the *Datalog* implication:

$$q(Y_0, \ldots, Y_n) \;\Leftarrow\; \mathcal{D}[\zeta]_{Y_0}$$

where $\mathcal{D}[\zeta]_X$ (X is a variable) is defined by

$$\begin{array}{lcl} \mathcal{D}[y_j]_X & = & X = Y_j \\ \mathcal{D}[\mathtt{b}(\zeta_1, \ldots, \zeta_m)]_X & = & \delta(X, \mathtt{b}, X_1 \ldots X_m) \;\wedge\; \mathcal{D}[\zeta_1]_{X_1} \wedge \ldots \wedge \mathcal{D}[\zeta_m]_{X_m} \\ \mathcal{D}[q'(\zeta_1, \ldots, \zeta_m)]_X & = & q'(X, X_1, \ldots, X_m) \;\wedge\; \mathcal{D}[\zeta_1]_{X_1} \wedge \ldots \wedge \mathcal{D}[\zeta_m]_{X_m} \end{array}$$

and the variables $X_1, \ldots, X_m$ in the last two rows are fresh. For subsets $X, X_1, \ldots, X_{m'}$ of the set of states of A, $\delta(X, \mathtt{a}, X_1 \ldots X_{m'})$ denotes the fact that $(x, \mathtt{a}, x_1 \ldots x_{m'}) \in \delta_A$ for all $x \in X$ and $x_j \in X_j$, $j = 1, \ldots, m'$. A bottom-up evaluation of the resulting *Datalog* program computes for every $q/(n+1)$, the set of all tuples $(p_0, \ldots, p_n)$ such that the translation of $\langle q, p_0 \ldots p_n \rangle$ is non-empty. If we, additionally, want to restrict these predicates only to tuples, which may contribute to a terminal derivation of initial nonterminals $\langle q_0, p_f \rangle$, we may top-down query the program with queries $\Leftarrow q_0(p_f)$. Practically, top-down solving organizes the construction such that only useful nonterminals of the intersection grammar are considered. Using this approach, the number of newly constructed nonterminals often will be much smaller than the bounds stated in the theorem. A similar construction is also possible for the intersection of MTTs with the complements of DTTA languages.

The algorithm for *input-linear* MTTs can also be applied to *non-input-linear* 2MTTs. Then, the constructed *Datalog* program does no longer precisely characterize the non-empty functions of the intersection 2MTT because dependencies on input subtrees (viz. several transformations of the same input node) have been lost. Accordingly, a *superset* is returned. By means of CFTGs, we can express this observation as follows:

Theorem 12.7. *Let G_M be the CFTG constructed for a 2MTT M. Then*

$$\tau_M(\mathcal{T}_\Sigma) \subseteq \mathcal{L}(G_M)\,.$$

Since the CFTG still provides a safe *superset* of produced outputs, type checking based on context-free tree grammars is sound in the sense that if it does not flag an error, the transformation also will not go wrong. On the other hand, a flagged error may be possibly spurious, i.e., due to the over-approximation of

the output language through the CFTG. Consider a top-down transducer M with the rules

$$\begin{aligned} q_0(\mathtt{a}, 0) &\to \mathtt{c}(p(down_1), p(down_1)) \\ p(\mathtt{a}, \eta) &\to \mathtt{a}(p(down_1)) \\ p(\mathtt{b}, \eta) &\to \mathtt{b}(p(down_1)) \\ p(\mathtt{e}, \eta) &\to \mathtt{e} \end{aligned}$$

where p realizes the identity. In this case, the corresponding approximating CFTG G_M is rather coarse: It generates $\mathtt{c}(u, v)$ with $u, v \in \{\mathtt{a}, \mathtt{b}\}^*\mathtt{e}$ (seen as monadic trees). Exact tree copying, however, can be realized through the use of parameters: The transducer with the rules

$$\begin{aligned} q_0(\mathtt{a}, 0) &\to q(down_1, p(down_1)) \\ q(\sigma, \eta, y_1) &\to \mathtt{c}(y_1, y_1) \qquad \text{for all } \sigma \in \Sigma \end{aligned}$$

and the same p-rules as M will realize the same translation as M. The context-free tree grammar for the resulting transducer now does not provide an over-approximation but precisely captures the output language of the top-down transducer M.

When approximating the output languages of general 2MTTs with CFTGs, we no longer can assume that the maximal number d of occurrences of nonterminals in a right-hand side of this grammar is bounded by a small constant. If d turns out to be unacceptably large, we still can apply Lemma 12.1 to limit the maximal number of occurrences of nonterminals in each right-hand side of the transducer to a number k which is the maximal rank of output symbols and states. This construction, however, introduces *stay*-moves and thus destroys input-linearity.

Example 12.4. For our example 2MTT $M_{y,\mathrm{staff}}$, we construct for each rule of the transducer a production of the context-free tree grammar $G_{M_{y,\mathrm{staff}}}$ and obtain:

$$\begin{aligned} q_I &\to \mathtt{staff}(q(\mathtt{e}, \mathtt{e}), \mathtt{e}) \\ q(y_b, y_n) &\to \mathtt{employee}(\mathtt{data}(q_{data}, y_b), q(\mathtt{boss}(q_{data}, \mathtt{e}), q(y_b, y_n))) \\ &\mid y_n \\ &\mid q(y_b, y_n) \\ q_{data} &\to copy \end{aligned}$$

◁

12.4 Notes and References

Macro tree transducers [EV85] are a combination of top-down tree transducers and macro grammars [Fis68]. Macro grammars are just like context-free tree grammars (CFTGs), but produce strings (a CFTG can be seen as a special

macro grammar because terms are particular strings). In [Fis68], Fischer already distinguishes IO and OI for macro grammars, and proves that the corresponding classes of languages are incomparable (which also holds in the tree case). Our normal form of Lemma 12.1 can be seen as a variant of Fischer's IO standard form, which in fact is very similar to Chomsky normal form of context-free grammars: there are exactly 2 or 0 nonterminals in every right-hand side of a grammar in IO standard form. A similar normal form might be possible for macro tree walking transducers, too, but will cause a larger size increase of the transducer (note that Fischer does not report on grammar sizes in his constructions). Fischer remarks, just after Corollary 3.1.6, that emptiness of IO macro languages can be reduced to emptiness of context-free languages, by simply dropping all parentheses, commas, argument, and terminal symbols. The resulting context-free (word) grammar generates the empty word if and only if the original language is empty. Since emptiness for a context-free grammar can be decided in linear time (see, e.g., [HMU01]), we obtain a linear time procedure for checking emptiness of IO context-free tree languages, as stated in Theorem 12.5. Context-free tree grammars were considered in [Rou70] and extensively studied in [ES77, ES78].

The fact that output languages of input-linear macro tree transducers are IO context-free tree languages is mentioned in Corollary 5.7 of [EV85] (the class of input-linear MTTs is called LMT_{IO} there). A 2MTT without up-moves is called "stay-MTT". Results similar as the ones obtained in this chapter for 2MTTs (Chapter 12) have been obtained already in [MPS07] for the restricted case of stay-MTTs. For instance, Proposition 3 of that paper is similar to our Theorem 12.1, Theorem 2 of [MPS07] corresponds to our Theorem 12.2, and Theorem 5 of that paper corresponds to our Theorem 12.6. In Lemma 34 of [EM03b] it is shown that every transformation definable by a 2TT (0-ptt in that paper) can be realized by a stay-MTT (sMTT). Note however, that this construction leads to a blow-up of parameters, which keep track of all possible runs on subtrees. Thus, the algorithm of [MPS07] falls short on our efficient subclasses of 2TTs (Section 11.3), the polynomial complexity bounds would not hold true for 2TTs using stay-MTTs.

Just before Theorem 12.6, we describe how to incorporate an input type into an input-linear transducer, so that the corresponding output language is preserved. The technical details are exactly as in the proof of Theorem 3.2.1 of [ERS80] where this result was proved for top-down tree transducers. 2MTTs are essentially the same as the k-pebble macro tree transducers (k-PMTT) of [EM03b] for the case $k = 0$. For k-PMTTs, a normal form similar to our Lemma 12.1 was shown in Theorem 16 of [EM03b].

Chapter 13

Macro Forest Walking Transducers

Conceptually, XML documents are not trees but forests. Therefore, we extend the concept of tree walking transducers (without or with parameters) to a transformation formalism of forests. Forests are introduced in Definition 2.1.

Example 13.1. We consider again a transformation from company structures (cf. Figure 1.2) to lists of employees under new root nodes labeled `staff` (cf. Chapter 9). In contrast to tree walking transducers, forest transducers do not depend on a ranked alphabet. They can deal with arbitrarily many subtrees of nodes. Here, we define a transformation, which returns trees of the form $\texttt{staff}\langle f\rangle$ where f is a forest composed of `employee`-trees, i.e., trees of the form $\texttt{employee}(\texttt{data}(\ldots), \texttt{boss}(\ldots))$. The input trees are described by the DTD in the beginning of Chapter 10. Additionally to the operations *up*, *stay* as in tree walking transducers, forest transducers may use a directive *down* for proceeding to the first child as well as directives *left* and *right* for proceeding to the left or right sibling, respectively.

$$
\begin{array}{lll}
1 & q_I(\texttt{department}, 0) & \to \texttt{staff}\langle q(\mathit{down}, \epsilon)\rangle \\
2 & q(\texttt{employee}, \eta, y_b) & \to \texttt{employee}\langle \texttt{data}\langle q_{\mathit{data}}(\mathit{down})\rangle\ y_b\rangle \\
3 & & \quad q(\mathit{down}, \texttt{boss}\langle q_{\mathit{data}}(\mathit{down})\rangle) \\
4 & & \quad q(\mathit{right}, y_b) \\
5 & q(\texttt{data}, 2, y_b) & \to q(\mathit{right}, y_b) \\
6 & q(\texttt{subordinates}, 3, y_b) & \to q(\mathit{down}, y_b) \\
7 & q(\epsilon, \eta, y_b) & \to \epsilon \\
8 & q_{\mathit{data}}(\texttt{data}, 2) & \to \mathit{copy}(\mathit{down})
\end{array}
$$

where state *copy* in Line 8 is meant to copy the forest f of a subtree $\texttt{data}\langle f\rangle$. The right-hand side of the second rule is a composition of three forests (Lines $2-4$). The initial state is q_I, which means that we start with state q_I at the root

of the first tree of a forest. Here, the transducer walks only the first tree of an input forest. Note also that now rules may be selected depending on the current label of a node in the forest together with its forest direction. Thus, the transducer can check whether the current node is the leftmost node on the top-level (value 0), is on the top-level, but not leftmost (value 1), is leftmost but not on the top-level (value 2), or is neither leftmost nor on the top-level (value 3). ◁

Definition 13.1 (2MFT)**.** A *macro forest walking transducer* (2MFT for short) is a tuple $M = (Q, \Sigma, Q_0, R)$ where

- Q is a finite ranked set of states,
- Σ is a finite alphabet with $Q \cap \Sigma = \emptyset$,
- $Q_0 \subseteq Q^{(1)}$ is the set of initial states, and
- R is a finite set of rules of the form:

$$q(\epsilon, \eta, y_1, \ldots, y_n) \to \zeta \quad \text{or} \quad q(\mathtt{a}, \eta, y_1, \ldots, y_n) \to \zeta$$

with $\mathtt{a} \in \Sigma$, direction $\eta \in \{0, \ldots, 3\}$ and $q \in Q^{(n+1)}$ where the right-hand sides are forests ζ of the following form:

$$\zeta ::= \epsilon \mid y_j \mid q'(op, \zeta_1, \ldots, \zeta_{n'}) \mid \mathtt{b}\langle\zeta_1\rangle \mid \zeta_1\,\zeta_2$$

where $q' \in Q^{(n'+1)}$, $\mathtt{b} \in \Sigma$, $op \in \{up, stay, down, left, right\}$ and $j \in [n]$. Moreover, the right-hand sides for empty input forests ϵ must not contain occurrences of the operations *down*, *right* or *left*.

In case of several rules for the same q, the same direction η and the same symbol $\mathtt{a}$ (or ϵ), we also write $\zeta_1 \mid \ldots \mid \zeta_k$ to list all occurring right-hand sides. In case, no operation *up* is used, the 2MFT is also called *top-down* (short: *1MFT* or *MFT*). Likewise, if all states are of rank 1, i.e., have no accumulating parameters, the 2MFT is an (ordinary) *forest walking* transducer (short: *2FT*). Finally, a 1MFT without parameters is also called *forest transducer* (short: *1FT* or *FT*). As for macro tree walking transducers, in practice, states q may differ in their *ranks*, i.e., the numbers of their accumulating parameters plus 1. The set R of rules in Example 13.1 constitutes the transducer $M_{y,\mathrm{staff},f} = (Q, \Sigma, Q_0, R)$ with $Q = \{q_I, q, q_{data}, copy\}$ and $Q_0 = \{q_I\}$, which happens to be a 1MFT.

A forest transducer behaves similar to a corresponding tree transducer: While walking over the input forest, the transducer chooses rules corresponding to the current states, input symbols and directions at the respective current nodes in the input, and then evaluates the right-hand sides of the rules. Again, we just consider the *inside-out* (IO or call-by-value) strategy for evaluating parameters. There are two significant differences between forest walking and tree walking transducers: First, a forest walking transducer produces output *forests* and therefore, as an extra operation, also supports *concatenation* of output

forests. Second, the forest transducer has a different set of directions as well as a different set of navigational directives: *up* now means that the transducer moves to its ancestor in the input forest, *left* and *right* means that the transducer moves to its left or right sibling, whereas *down* means that the transducer moves to its first child. Formally, the semantics of these operations is defined by:

$$\begin{aligned} [\![up]\!](vi) &= v \\ [\![left]\!](vi) &= v(i-1) \quad \text{if } i > 0 \\ [\![right]\!](vi) &= v(i+1) \\ [\![down]\!](v) &= v0 \end{aligned}$$

Note that only the operations *down* and *right* have immediate equivalents in commands of a transducer on the *first-child next-sibling* encoding of forests as binary trees where they correspond to the commands $down_1$ and $down_2$, respectively. The *up*-command on the tree, on the other hand, may correspond to the forest commands *left* or *up* — depending on whether the current node is a right or left child. Analogously to Lemma 12.1, we find:

Lemma 13.1. *For every 2MFT M, there exists a 2MFT M' with*

- $\tau_{M'} = \tau_M$,
- *there are at most l occurrences of states on a right-hand side of rules in M',*
- $|M'| \in \mathcal{O}(|M| \cdot l^2)$

where l is the maximum of 2 and the maximal rank of states of M.

Note that it is unfortunately *not* possible to simulate 2MFTs by 2MTTs, which work on (possibly enriched) first-child next-sibling encodings of input and output trees. To see this, consider first the 1MFT case. As shown in [PS04], one can easily construct a 1MFT, which takes as input a binary tree with m nodes, and outputs a forest consisting of 2^m leaves, i.e., a string of length 2^m. Consider the height increase of the corresponding translation on encoded trees: it is *double*-exponential. However, the height-increase of MTTs is at most exponential (see [EV85]). Now consider the 2MFT case. Clearly, a 2MTT can translate a binary tree with m nodes into a monadic tree with 2^m nodes by doing a depth-first left-to-right traversal and at each step generating a duplicated state call in a parameter position. As intermediate sentential form, the transducer generates $q(\varepsilon, q(\varepsilon, \ldots, q(\varepsilon, \mathtt{e}))\ldots)$, which has 2^m-many occurrences of q; it then replaces $q(\varepsilon, t)$ by $\mathtt{g}(t)$ where $\mathtt{g}$ is an output symbol of rank 1. A 2MFT can generate the same sentential form, but can replace $q(\varepsilon, t)$ by tt, i.e., the forest of concatenating two copies of t. In this way, a forest consisting of 2^{2^m}-many $\mathtt{e}$'s is generated. On first-child next-sibling encodings, this corresponds to a tree of height 2^{2^m}. Thus, the translation on encodings has *double*-exponential size-to-height increase. However, it is not difficult to see that the size-to-height increase of 2MTTs is at most exponential.

13.1 Intersecting 2MFTs with Output Types

In this section, we consider general techniques for intersecting forest walking transducers with output types. Assume that we are given a regular forest language $\mathcal{L}$. Our goal is to construct for a given 2MFT M another 2MFT M', which behaves similar to M but produces only outputs in $\mathcal{L}$. If $\mathcal{L}$ describes the set of all invalid outputs, type checking for M, thus, reduces to checking emptiness of the transformation M'.

In order to provide a general construction for regular forest languages, let us first assume that $\mathcal{L}$ is given as the language defined by a finite forest monoid A, i.e., $\mathcal{L} = \mathcal{L}(A)$. A *finite forest monoid* (short: FFM) can be considered as a deterministic bottom-up automaton, which combines the individual states for the trees u_i in a forest $f = u_1 \dots u_m$ by means of a monoid operation $\circ$ (compare, e.g., the discussion in [BW05]). Formally, a *finite forest monoid* consists of a finite monoid G with a neutral element e, a finite subset $F \subseteq G$ of accepting elements, together with a function $up : \Sigma \times G \to G$ mapping a symbol of Σ together with a monoid element for its content to a monoid element representing a forest of length 1. A finite forest monoid accepts a forest f if $up^*(f) \in F$ where

$$\begin{aligned} up^*(f_1 f_2) &= up^*(f_1) \circ up^*(f_2) \\ up^*(\mathtt{a}\langle f' \rangle) &= up(\mathtt{a}, up^*(f')) \\ up^*(\epsilon) &= e \end{aligned}$$

Given a complete deterministic bottom-up forest automaton $A = (P, \Sigma, \delta, F_A)$, i.e., a DBTA operating on the first-child next-sibling representation of forests, we construct a finite forest monoid as follows. Let G be the monoid of functions $P \to P$ where the monoid operation is function composition. In particular, the neutral element of this monoid is the identity function. Moreover, the function up is defined by

$$up(\mathtt{a}, g) = p \mapsto \delta(\mathtt{a}, g(\delta(\mathtt{e}))\, p) \ .$$

Finally, the set of accepting elements is given by

$$F = \{g \in G \mid g(\delta(\mathtt{e})) \in F_A\} \ .$$

On the other hand, every forest monoid G gives rise to a finite tree automaton A_G (running on first-child next-sibling representations) whose set of states is given by the elements of G. The transition function δ of A_G is defined by:

$$\delta(\mathtt{e}) = e \quad \text{and} \quad \delta(\mathtt{a}, g_1\ g_2) = up(\mathtt{a}, g_1) \circ g_2 \ .$$

Then the set of accepting states simply is given by the accepting elements of G. These constructions show that every recognizable forest language can be recognized by a finite forest monoid and vice versa. Although the FFM for a bottom-up tree automaton generally can be exponentially larger, this need not always be the case.

Example 13.2. For our current example, the BTA in Example 10.1 is not a complete deterministic bottom-up forest automaton. We get a complete DBFA by adding an extra error state $\bullet$. The new transition function δ' then is defined by:

$$\begin{aligned}
\delta'(\texttt{staff}, r_{\text{empl}}\ r_{\texttt{e}}) &= r_{\text{staff}} & \delta'(\texttt{staff}, r_{\texttt{e}}\ r_{\texttt{e}}) &= r_{\text{staff}} \\
\delta'(\texttt{employee}, r_{\text{data}}\ r_{\text{empl}}) &= r_{\text{empl}} & \delta'(\texttt{employee}, r_{\text{data}}\ r_{\texttt{e}}) &= r_{\text{empl}} \\
\delta'(\texttt{data}, r_{\text{name}}\ r_{\text{boss}}) &= r_{\text{data}} & \delta'(\texttt{data}, r_{\text{name}}\ r_{\texttt{e}}) &= r_{\text{data}} \\
\delta'(\texttt{boss}, r_{\text{name}}\ r_{\texttt{e}}) &= r_{\text{boss}} & \delta'(\texttt{name}, r_{\text{content}}\ r_{\texttt{e}}) &= r_{\text{name}} \\
\delta'(\texttt{e}, \epsilon) &= r_{\texttt{e}}
\end{aligned}$$

and $\delta'(\texttt{a}, r_1\ r_2) = \bullet$ for all other combinations of a label $\texttt{a} \in \Sigma$ and states $r_1, r_2 \in P \cup \{\bullet\}$. In the corresponding finite forest monoid $A = (G, \Sigma, up, F)$, the monoid G contains the following functions:

$$\begin{aligned}
g_{\text{staff}} &= \{r_{\texttt{e}} \mapsto r_{\text{staff}}\} \\
g_{\text{empl}} &= \{r_{\text{empl}} \mapsto r_{\text{empl}}, r_{\texttt{e}} \mapsto r_{\text{empl}}\} \\
g_{\text{data}} &= \{r_{\text{boss}} \mapsto r_{\text{data}}, r_{\texttt{e}} \mapsto r_{\text{data}}\} \\
g_{\text{boss}} &= \{r_{\texttt{e}} \mapsto r_{\text{boss}}\} \\
g_{\text{name}} &= \{r_{\texttt{e}} \mapsto r_{\text{name}}\} \\
g_{\text{dataBoss}} &= \{r_{\texttt{e}} \mapsto r_{\text{data}}\} \\
g_{\text{content}} &= \{r_{\texttt{e}} \mapsto r_{\text{content}}\} \\
g_{\bullet} &= \emptyset \\
Id &= \{r \mapsto r \mid r \in P\}
\end{aligned}$$

where we have omitted all entries $r \mapsto \bullet$. Note that in this example, the forest monoid has only one element more than the underlying finite automaton. Also, the composition table of these functions is given by $Id \circ g = g \circ Id = g$ for all g and furthermore:

$$\begin{aligned}
g_{\text{data}} \circ g_{\text{boss}} &= g_{\text{dataBoss}} \\
g_{\text{empl}} \circ g_{\text{empl}} &= g_{\text{empl}} \\
g \circ g' &= g_{\bullet} \quad \text{otherwise}
\end{aligned}$$

For the function up, we find:

$$\begin{aligned}
up(\texttt{staff}, Id) &= g_{\text{staff}} & up(\texttt{staff}, g_{\text{empl}}) &= g_{\text{staff}} \\
up(\texttt{employee}, g_{\text{data}}) &= g_{\text{empl}} & up(\texttt{employee}, g_{\text{dataBoss}}) &= g_{\text{empl}} \\
up(\texttt{data}, g_{\text{name}}) &= g_{\text{data}} \\
up(\texttt{boss}, g_{\text{name}}) &= g_{\text{boss}} & up(\texttt{name}, g_{\text{content}}) &= g_{\text{name}} \\
up(\texttt{a}, g) &= g_{\bullet} \quad \text{otherwise}
\end{aligned}$$

where the set of accepting functions is given by $F = \{g_{\text{staff}}\}$. ◁

Theorem 13.2. *For every 2MFT M and every finite forest monoid A, a 2MFT M_A can be constructed such that for all $f \in \mathcal{F}_\Sigma$,*

$$\tau_{M_A}(f) = \tau_M(f) \cap \mathcal{L}(A) \ .$$

The size of M_A is in $\mathcal{O}(|M| \cdot |A|^{l\cdot(d+1)})$ where l is the maximal rank of a state q of M and d is the maximal number of occurrences of states in right-hand sides in M.

Proof. Let $M = (Q, \Sigma, R, Q_0)$ and $A = (G, \Sigma, up, F)$. For each state q in Q with rank $n+1$ and all monoid elements $g_0, \ldots, g_n \in G$, we generate new states for the intersection 2MTT M_A of the form $\langle q, g_0 g_1 \ldots g_n\rangle$. Such a state is meant to generate all forests $f \in \mathcal{F}_\Sigma(\{y_1, \ldots, y_n\})$ for which there is a run of A starting at the leaves y_i with monoid element g_i and reaching the root of f in g_0. The rules of the new 2MFT M_A are:

$$\langle q, g_0 g_1 \ldots g_n\rangle(\mathtt{a}, \eta, y_1, \ldots, y_n) \to \zeta'$$

for every rule $q(\mathtt{a}, \eta, y_1, \ldots, y_n) \to \zeta$ of M and $\zeta' \in \tau^{g_0 g_1 \cdots g_n}[\zeta]$ where the sets $\tau^{g_0 g_1 \cdots g_n}[.]$ are inductively defined by:

$$\begin{array}{lcll}
\tau^{g_0 g_1 \cdots g_n}[y_j] & = & \{y_j & \mid g_0 = g_j\} \\
\tau^{g_0 g_1 \cdots g_n}[\mathtt{b}\langle\zeta\rangle] & = & \{\mathtt{b}\langle\zeta'\rangle & \mid up(\mathtt{b}, g') = g_0 \ \wedge \ \zeta' \in \tau^{g' g_1 \cdots g_n}[\zeta]\} \\
\tau^{g_0 g_1 \cdots g_n}[\epsilon] & = & \{\epsilon & \mid g_0 = e\} \\
\tau^{g_0 g_1 \cdots g_n}[\zeta_1\zeta_2] & = & \{\zeta_1'\zeta_2' & \mid g_0 = g_1' \circ g_2' \ \wedge \ \forall \nu : \zeta_\nu' \in \tau^{g_\nu' g_1 \cdots g_n}[\zeta_\nu]\} \\
\tau^{g_0 g_1 \cdots g_n}[q'(op, \zeta_1, \ldots, \zeta_{n'})] & & & \\
\qquad = & \{\langle q', g_0 g_1' \ldots g_{n'}'\rangle(op, \zeta_1', \ldots, \zeta_{n'}') & \mid \forall \nu : \zeta_\nu' \in \tau^{g_\nu' g_1 \cdots g_n}[\zeta_\nu]\}
\end{array}$$

The set of initial states of M_A is $Q_0' = Q_0 \times F$. By fixpoint induction, we verify for every state q of rank $n \geq 1$, every input forest $f \in \mathcal{F}_\Sigma$, every node $\vartheta \in \mathtt{Nodes}(f)$ and monoid elements $g_0, \ldots, g_n$ that:

$$[\![\langle q, g_0, \ldots, g_n\rangle]\!]_f(\vartheta) = [\![q]\!]_f(\vartheta) \cap \{f' \in \mathcal{F}_\Sigma(Y) \mid up^*(f', g_1 \ldots g_n) = g_0\} \quad (13.1)$$

where $Y = \{y_1, \ldots, y_n\}$ and up^* is the extension of up to forests containing variables from Y, namely, for $\underline{g} = g_1 \ldots g_n$, we have

$$\begin{aligned}
up^*(y_i, \underline{g}) &= g_i \\
up^*(\epsilon, \underline{g}) &= e \\
up^*(\mathtt{a}\langle f'\rangle, \underline{g}) &= up(\mathtt{a}, up^*(f', \underline{g})) \\
up^*(f_1 f_2, \underline{g}) &= up^*(f_1, \underline{g}) \circ up^*(f_2, \underline{g})
\end{aligned}$$

The correctness of the construction follows from Equation 13.1.

For each state in M, we have at most $|A|^l$ new states in M_A, if l is the maximal rank of states in M. If we have d occurrences of states in the right-hand side of a rule r of M, we obtain $|A|^{l\cdot(d+1)}$ new rules for r in M_A. Therefore, the new 2MFT is of size $\mathcal{O}(|M| \cdot |A|^{l\cdot(d+1)})$ where l is the maximal rank of a state in M and d bounds the number of occurrences of states in right-hand sides in M. □

Note that this construction differs from the corresponding construction for 2MTTs in that we now additionally have to take concatenations of forests into account. It is precisely for this operation that we rely on the monoid structure of the set G.

Example 13.3. Consider the 2MFT M of Example 13.1 and the FFM A in Example 13.2. We get an intersection 2MFT with the following rules, for both directions $\eta \in \{2, 3\}$:

$$
\begin{array}{ll}
1 & \langle q_I, g_{\text{staff}}\rangle(\texttt{department}, 0) \to \texttt{staff}\langle\langle q, \mathit{Id}\ \mathit{Id}\rangle(\mathit{down}, \epsilon)\rangle \\
2 & \qquad\qquad\qquad\qquad\qquad\quad \mid \texttt{staff}\langle\langle q, g_{\text{empl}}\mathit{Id}\rangle(\mathit{down}, \epsilon)\rangle \\
3 & \langle q, g_{\text{empl}}\mathit{Id}\rangle(\texttt{employee}, \eta, y_b) \to \texttt{employee}\langle\texttt{data}\langle\langle q_{\mathit{data}}, g_{\text{name}}\rangle(\mathit{down})\rangle\ y_b\rangle \\
4 & \qquad\qquad\qquad\qquad\qquad\quad \langle q, g_2 g_{\text{boss}}\rangle(\mathit{down}, \texttt{boss}\langle\langle q_{\mathit{data}}, g_{\text{name}}\rangle(\mathit{down})\rangle) \\
5 & \qquad\qquad\qquad\qquad\qquad\quad \langle q, g_3\mathit{Id}\rangle(\mathit{right}, y_b) \\
6 & \langle q, g_{\text{empl}} g_{\text{boss}}\rangle(\texttt{employee}, \eta, y_b) \to \texttt{employee}\langle\texttt{data}\langle\langle q_{\mathit{data}}, g_{\text{name}}\rangle(\mathit{down})\rangle\ y_b\rangle \\
7 & \qquad\qquad\qquad\qquad\qquad\quad \langle q, g_2 g_{\text{boss}}\rangle(\mathit{down}, \texttt{boss}\langle\langle q_{\mathit{data}}, g_{\text{name}}\rangle(\mathit{down})\rangle) \\
8 & \qquad\qquad\qquad\qquad\qquad\quad \langle q, g_3 g_{\text{boss}}\rangle(\mathit{right}, y_b) \\
9 & \langle q, g_0 g_b\rangle(\texttt{data}, 2, y_b) \to \langle q, g_0 g_b\rangle(\mathit{right}, y_b) \\
10 & \langle q, g_0 g_b\rangle(\texttt{subordinates}, 3, y_b) \to \langle q, g_0 g_b\rangle(\mathit{down}, y_b) \\
11 & \langle q, \mathit{Id}\ g_b\rangle(\epsilon, \eta, y_b) \to \epsilon \\
12 & \langle q_{\mathit{data}}, g_{\text{name}}\rangle(\texttt{data}, 2) \to \langle \mathit{copy}, g_{\text{name}}\rangle(\mathit{down})
\end{array}
$$

For the two monoid elements g_2 and g_3 in the third and the forth rules holds $g_{\text{empl}} = g_{\text{empl}} \circ g_2 \circ g_3$. Thus, g_2 and g_3 are in $\{g_{\text{empl}}, \mathit{Id}\}$. The element g_0 in Lines 9 and 10 is either g_{empl} or Id, whereas g_b in Lines 9-11 is in $\{g_{\text{boss}}, \mathit{Id}\}$. Additionally, there are rules resulting in states $\langle q', g_\bullet g_b\rangle$ or $\langle q', g_\bullet\rangle$ for states $q' \in Q$ of the 2MFT. ◁

The draw-back of this general construction, though, is that the (complement of the) output type, with which we aim to intersect, first must be represented as a finite forest monoid. In general, this alone may incur an exponential blow-up. If, however, the macro forest transducer is *output-linear*, i.e., uses each parameter at most once, then a much cheaper direct construction is possible. In particular, this cheaper construction applies to 2FTs since these transducers have no parameters at all.

Theorem 13.3. *Assume that M is an output-linear 2MFT. Then for every (possibly nondeterministic) BFA A, a 2MFT M_A can be constructed with*

$$\tau_{M_A}(f) = \tau_M(f) \cap \mathcal{L}(A)$$

for all $f \in \mathcal{F}_\Sigma$. The size of the 2MFT $|M_A|$ is in $\mathcal{O}(|M| \cdot |A|^{2l\cdot(d+1)})$ where l is the maximal rank of a state q of M and d is the maximal number of occurrences of states in right-hand sides in M.

Proof. Let $M = (Q, \Sigma, R, Q_0)$ and $A = (P, \Sigma, \delta, \{p_0\})$. The idea for the new 2MFT M_A for the intersection is to maintain for every possibly produced output

forest f a *pair* of states $\langle p, p' \rangle$ so that the automaton A, when starting in p' to the right of f, possibly may arrive in state p to the left. Accordingly, the set Q' of M_A consists of all states

$$\langle q, p_0 p'_0 \dots p_n p'_n \rangle$$

where $q \in Q$ is of rank $n+1$, i.e., has n accumulating parameters and $p_i, p'_i \in P$ for all i. Accordingly, the rules of the new 2MFT are of the form:

$$\langle q, p_0 p'_0 \dots p_n p'_n \rangle(\mathtt{a}, \eta, y_1, \dots, y_n) \;\to\; f' \quad \text{with } f' \in \tau^{p_0 p'_0 \cdots p_n p'_n}[f]$$

for every rule $q(\mathtt{a}, \eta, y_1, \dots, y_n) \to f$ of M. The sets $\tau^{p_0 p'_0 \cdots p_n p'_n}[.]$ are defined by:

$$\begin{array}{lcll}
\tau^{p_j p'_j \; p_1 p'_1 \cdots p_n p'_n}[y_j] & = & \{y_j\} & \\
\tau^{p_0 p'_0 \; p_1 p'_1 \cdots p_n p'_n}[\mathtt{b}\langle\zeta\rangle] & = & \{\mathtt{b}\langle\zeta'\rangle & \mid (p_0, \mathtt{b}, p''_1 \; p'_0) \in \delta \wedge (p''_2, \mathtt{e}) \in \delta \\
 & & & \quad \wedge \zeta' \in \tau^{p''_1 p''_2 \; p_1 p'_1 \cdots p_n p'_n}[\zeta]\} \\
\tau^{p_0 p'_0 \; p_1 p'_1 \cdots p_n p'_n}[\epsilon] & = & \{\epsilon & \mid p_0 = p'_0\} \\
\tau^{p_0 p'_0 \; p_1 p'_1 \cdots p_n p'_n}[\zeta_1 \zeta_2] & = & \{\zeta'_1 \zeta'_2 & \mid \exists\, p:\; \zeta'_1 \in \tau^{p_0 p \; p_1 p'_1 \cdots p_n p'_n}[\zeta_1] \\
 & & & \quad \wedge \zeta'_2 \in \tau^{p p'_0 \; p_1 p'_1 \cdots p_n p'_n}[\zeta_2]\}
\end{array}$$

$$\begin{array}{l}
\tau^{p_0 p'_0 \; p_1 p'_1 \cdots p_n p'_n}[q'(op, \zeta_1, \dots, \zeta_m)] \\
\quad = \{\langle q', p_0 p'_0 \; p''_1 p'''_1 \dots p''_m p'''_m \rangle(op, \zeta'_1, \dots, \zeta'_m) \mid \forall \nu : \zeta'_\nu \in \tau^{p''_\nu p'''_\nu \; p_1 p'_1 \cdots p_n p'_n}[\zeta_\nu]\}
\end{array}$$

The set of initial states of M_A then consists of all states $\langle q, p_0 p' \rangle$ where $q \in Q_0$ and $p_0 \in P$ are accepting states of M and A, respectively, and $(p', \mathtt{e}) \in \delta$. The estimation of the size of the resulting transducer is similar to the case of forest monoids — only that we have to replace the number of monoid elements with the number of *pairs* of states. Thus, the new intersection transducer is of size $\mathcal{O}(|M| \cdot |A|^{2l \cdot (d+1)})$. □

13.2 Deciding Emptiness of 2MFTs

For deciding emptiness of a forest transducer M, we conceptually follow the approach taken for tree transducers. There, we first constructed an alternating tree walking automaton accepting the domain of M for which, in the second step, a nondeterministic tree automaton is constructed. In our case, this would mean that we first formally introduce the concept of alternating *forest walking* automata for which, in a separate construction, a nondeterministic forest automaton is constructed. In order to simplify this, we will not intermediately rely on forest walking automata. Instead, we consider for each forest f an *enriched* first-child next-sibling encoding through binary trees. This means that inside each node of the encoding, we additionally record whether or not the current tree node represents a node on the top-level of the forest. Let $\bar{\Sigma} = \{\bar{a} \mid a \in \Sigma\}$ denote a set of new symbols of rank 2. Then the ranked alphabet used by the encoding will be $\Sigma_2 = \Sigma \cup \bar{\Sigma} \cup \{\mathtt{e}, \bar{\mathtt{e}}\}$ where the barred symbols will only occur on the rightmost spine in the tree. A DTTA with two states $\{\mathtt{t}, \mathtt{n}\}$ can check whether a tree in $\mathcal{T}_{\Sigma_2}$ is the enriched encoding of a forest or not.

Since the encoding is injective, it suffices for a forest transducer M to construct an ATWA M', which defines the set of encodings of the domain of M. Then the set of states of the ATWA M' is given by $Q' = \{q'_0, \mathtt{t}, \mathtt{n}\} \cup Q \cup Q^{up}$ where $Q^{up} = \{q^{up} \mid q \in Q\}$ is a set of fresh copies of the states in Q and q'_0 serves as fresh initial state of M'. Assume that $q(\mathtt{a}, \eta, y_1, \ldots, y_n) \to \zeta$ is a rule of M and $q_1(op_1, \ldots), \ldots, q_c(op_c, \ldots)$ is the sequence of recursive calls in ζ. If $\eta \in \{2, 3\}$, i.e., if the rule is not applicable to nodes at the top-level of the input forest, then ATWA M' has the rule:

$$q(\mathtt{a}, \eta - 1) \to q'_1(op'_1) \wedge \ldots \wedge q'_c(op'_c)$$

where:

$$q'_j(op'_j) \quad = \quad \begin{cases} q_j(\mathit{stay}) & \text{if } op_j = \mathit{stay} \\ q_j(\mathit{down}_1) & \text{if } op_j = \mathit{down} \\ q_j(\mathit{down}_2) & \text{if } op_j = \mathit{right} \\ q_j(\mathit{up}) & \text{if } op_j = \mathit{left} \text{ and } \eta = 3 \\ q_j^{up}(\mathit{stay}) & \text{if } op_j = \mathit{up} \end{cases}$$

Furthermore, for states q^{up}, the ATWA M' has the rules:

$$\begin{array}{lcl} q^{up}(\mathtt{a}, 2) & \to & q(\mathit{up}) \\ q^{up}(\mathtt{a}, 3) & \to & q^{up}(\mathit{up}) \end{array}$$

If on the other hand $\eta \in \{0, 1\}$, i.e., the rule of the 2MFT refers to nodes at the top-level of the input forest, then the ATWA M' has the rule:

$$q(\bar{\mathtt{a}}, 2 \cdot \eta) \to q'_1(op'_1) \wedge \ldots \wedge q'_c(op'_c)$$

where:

$$q'_j(op'_j) \quad = \quad \begin{cases} q_j(\mathit{stay}) & \text{if } op_j = \mathit{stay} \\ q_j(\mathit{down}_1) & \text{if } op_j = \mathit{down} \\ q_j(\mathit{down}_2) & \text{if } op_j = \mathit{right} \\ q_j(\mathit{up}) & \text{if } op_j = \mathit{left} \text{ and } \eta = 1 \end{cases}$$

For every rule $q(\mathtt{a}, 0) \to \zeta$ of M with $q \in Q_0$, we add the rule

$$q'_0(\bar{\mathtt{a}}, 0) \to \mathtt{t}(\mathit{stay}) \wedge q(\mathit{stay})$$

where the rules for $\mathtt{t}$ and $\mathtt{n}$ simulate the computation of a top-down automaton to verify that the input tree is the enriched encoding of a forest. Thus, the rules of the ATWA for q'_0 are meant to spawn a subrun, which verifies the encoding and to spawn another subrun, which simulates an accepting run of the forest transducer on the binary encoding. In particular, the states q^{up} are auxiliary states to implement the operation *up* on the binary representation of the forest. More precisely, the rules for the state q^{up} perform the operation *up* as long as the current node is a right child. If the current node is a left child, a final *up*-operation is executed to arrive at the tree representation of the father node in state q. Overall, we find:

Theorem 13.4. *For every 2MFT M, an ATWA M' can be constructed in polynomial time such that $\mathcal{L}(M')$ is the set of enriched binary encodings of the set $\{f \mid \tau_M(f) \neq \emptyset\}$. In particular, $\mathcal{L}(M') \neq \emptyset$ iff $\tau_M \neq \emptyset$.*

We thus obtain an exponential algorithm for deciding emptiness of 2MFTs, which is optimal. Together with our intersection constructions, this algorithm then can be applied also for type checking 2MFT transducers w.r.t. regular input and output types.

In order to arrive at more tractable algorithms or sub-classes, we can apply the same ideas as for 2MTTs: in the first place, we can again approximate the set of output forests by means of a context-free forest grammar. A context-free forest grammar for the intersection with a regular forest language specified through a finite forest monoid is polynomial in the size of the grammar and the number of elements in the monoid and exponential only in $l \cdot (d+1)$ where l is the maximal rank of nonterminals and d is the number of occurrences of nonterminals in right-hand sides. Emptiness for this forest grammar again can be checked in time linear in the size of the grammar. A practical implementation again may construct a *Datalog* program for the sets of useful nonterminals of the grammar. We can also generalize the notions of b-boundedness and strict b-boundedness for 2MFTs. While emptiness for b-bounded 2MFTs is decidable in polynomial time (where the exponent again depends on b^2), it is only strict b-boundedness, which is preserved by our intersection constructions. Here we only state the corresponding result for output types specified through finite forest monoids.

Theorem 13.5. *Assume M is a strictly b-bounded 2MFT and I and O are regular forest languages where I and O are given by a finite forest automaton and a finite forest monoid, respectively. Assume further that l is the maximal rank of a state of M and d is the maximal number of occurrences of state calls in right-hand sides. Then M can be type checked w.r.t. I and O in time polynomial in the sizes of M, the automaton for I and the automaton for O where the exponent linearly depends on $(b+1)^2 \cdot l \cdot (d+1)$.*

13.3 Notes and References

Top-down macro forest transducers have been introduced by Perst and Seidl in [PS04]. They are closely related to the top-down transducers of Maneth and Neven [MN99] but slightly more general. It was shown in [PS04] that, even though MFTs are more powerful than MTTs, they can be type checked with the same complexity bounds as macro tree transducers. This idea was extended to two-fold compositions of deterministic MTTs, in [MN08]. In [MBPS05], a general forest transformation language TL is introduced, which captures most features of XML transformation languages such as XSLT. The language TL supports full MSO pattern matching both for the selection of rules applicable at a node in the input tree and for navigation inside the input tree. Thus, 2MFTs can be considered as a sub-language of TL where rules are selected depending

on the current state and input label only, and where navigation is restricted to immediate neighbors in the input forest. The main contribution of that paper is to show how such transformations can be decomposed into three stay macro tree transducers running on the first-child next-sibling encoding of the XML documents in question. The semantics considered there was OI evaluation of nested calls but similar results can also be proven for IO evaluation, i.e., call-by-value parameter passing as considered here [Per07].

Chapter 14

Conclusion

In this part, we have reviewed basic constructions for tree walking transducers, which allow to obtain algorithms for type checking the transducers w.r.t. regular input and output types. There are three orthogonal variations in which the basic concept of a finite state machine can be made more expressive:

- top-down versus walking
- without parameters versus with parameters
- on ranked trees versus on unranked forests

At the very heart of our algorithms for type checking is to check whether or not a transducer realizes an empty translation. Already for the weakest, i.e., one-way top-down transducers, emptiness turns out to be complete for deterministic exponential time. Still, however, we were able to pin-point one major source for the complexity, namely the number of visits to the same input node. If the transducer visits the same node only constantly often, i.e., is b-bounded for some constant b, then emptiness becomes decidable in polynomial time.

The second ingredient of our algorithm are constructions for computing intersection transducers, i.e., transducers, which only produce outputs outside a specified regular set. Here, we considered regular sets as specified by bottom-up deterministic automata (or monoids, in case of forests) or by deterministic top-down automata. The latter construction for forest transducers was at least applicable to *output-linear* transducers, i.e., transducers, which use each of their accumulating parameters at most once. Two separate constructions are crucial, since translating top-down deterministic automata into bottom-up automata may incur an extra exponentiation in the number of states. Since these constructions preserve strict b-boundedness, we, thus, overall arrive at a general class of transducers for which type checking is polynomial.

Bibliography

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[AD82] André Arnold and Max Dauchet. Morphismes et Bimorphismes d'Arbres. *Theoretical Computer Science*, 20:33–93, 1982.

[ADB00] Hiyan Alshawi, Shona Douglas, and Srinivas Bangalore. Learning Dependency Translation Models as Collections of Finite-state Head Transducers. *Computational Linguistics*, 26:45–60, 2000.

[AG68] Michael A. Arbib and Yehoshafat Give'on. Algebra Automata I: Parallel Programming as a Prolegomena to the Categorical Approach. *Information and Control*, 12(4):331–345, 1968.

[AU71] Alfred V. Aho and Jeffrey D. Ullman. Translations on a Context-Free Grammar. *Information and Control*, 19(5):439–475, 1971.

[Bak79] Brenda S. Baker. Composition of Top-Down and Bottom-Up Tree Transductions. *Information and Control*, 41(2):186–213, 1979.

[Bar82] Miklós Bartha. An algebraic definition of attributed transformations. *Acta Cybernetica*, 5:409–421, 1982.

[BCF+03] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, Jérôme Siméon, and Mugur Ştefănescu, editors. XQuery 1.0: An XML Query Language. W3C Working Draft. Available at http://www.w3.org/TR/xquery/, 2003.

[BPS98] Tim Bray, Jean Paoli, and C. Michael Sperberg-McQueen, editors. *Extensible Markup Language (XML) 1.0*. W3C, first edition, 10 February 1998.
Available at http://www.w3.org/TR/1998/REC-xml-19980210.

[BPS08] Tim Bray, Jean Paoli, and C. Michael Sperberg-McQueen et. al., editors. *Extensible Markup Language (XML) 1.0*. W3C, fifth edition, 26 November 2008.
Available at http://www.w3.org/TR/2008/REC-xml-20081126.

[BR00] Srinivas Bangalore and Owen Rambow. Exploiting a Probabilistic Hierarchical Model for Generation. In *Proceedings of the 18th conference on Computational linguistics - Volume 1*, COLING '00, pages 42–48, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[Bra68] Walter S. Brainerd. The Minimalization of Tree Automata. *Information and Control*, 13(5):484–491, 1968.

[Bra69] Walter S. Brainerd. Tree Generating Regular Systems. *Information and Control*, 14(2):217–231, 1969.

[BS01] Franz Baader and Wayne Snyder. Unification Theory. In John A. Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, pages 447–533. Elsevier Science Publishers, 2001.

[BW05] Mikolaj Bojańczyk and Igor Walukiewicz. Unranked Tree Algebra. Technical report, University of Warsaw, 2005.

[CD99] James Clark and Steve DeRose, editors. XML Path Language (XPath) 1.0. W3C Recommendation, November 1999.
Available at http://www.w3.org/TR/xpath.

[CDG+07] Hubert Comon, Max Dauchet, Remi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree Automata Techniques and Applications, 2007.
Available at http://www.grappa.univ-lille3.fr/tata release October, 12th 2007.

[CFZ82] Bruno Courcelle and Paul Franchi-Zannettacci. On the Equivalence Problem for Attribute Systems. *Information and Control*, 52(3):275–305, 1982.

[CGKV88] Stavros Cosmadakis, Haim Gaifman, Paris Kanellakis, and Moshe Vardi. Decidable Optimization Problems for Database Logic Programs. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 477–490, New York, NY, USA, 1988.

[Cho03] Christian Choffrut. Minimizing Subsequential Transducers: A Survey. *Theoretical Computer Science*, 292:131–143, January 2003.

[CM01] James Clark and Makoto Murata, editors. *RelaxNG Specification.* OASIS, 2001.
Available at http://www.oasis-open.org/committees/relax-ng.

[CNT04] Julien Carme, Joachim Niehren, and Marc Tommasi. Querying Unranked Trees with Stepwise Tree Automata. In Vincent van Oostrom, editor, *International Conference on Rewriting Techniques and*

Applications, volume 3091 of *Lecture Notes in Computer Science*, pages 105–118. Springer, June 2004.

[Cou78] Bruno Courcelle. A Representation of trees by languages II. *Theoretical Computer Science*, 7(1):25–55, 1978.

[DE98] Frank Drewes and Joost Engelfriet. Decidability of Finiteness of Ranges of Tree Transductions. *Information and Computation*, 145(1):1–50, 1998.

[DS11] Deepak D'Souza and Priti Shankar, editors. *Modern Applications of Automata Theory*, volume 2 of *IISc Research Monographs*. World Scientific, 2011. To appear.

[DT90] Max Dauchet and Sophie Tison. The Theory of Ground Rewrite Systems is Decidable. In *Logic in Computer Science*, LICS '90, pages 242 –248, June 1990.

[EHS07] Joost Engelfriet, Hendrik Jan Hoogeboom, and Bart Samwel. XML Transformation by Tree-walking Transducers with Invisible Pebbles. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '07, pages 63–72, New York, NY, USA, 2007.

[Eis03] Jason Eisner. Learning Non-isomorphic Tree Mappings for Machine Translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, ACL '03, pages 205–208, 2003.

[ELM08] Joost Engelfriet, Eric Lilin, and Andreas Maletti. Extended Multi Bottom-Up Tree Transducers. In *Proceedings of the 12th international conference on Developments in Language Theory*, volume 5257 of *DLT '08*, pages 289–300. Springer-Verlag, Berlin, Heidelberg, 2008.

[ELM09] Joost Engelfriet, Eric Lilin, and Andreas Maletti. Extended Multi Bottomup Tree Transducers. *Acta Informatica*, 46(8):561–590, October 2009.

[EM99] Joost Engelfriet and Sebastian Maneth. Macro Tree Transducers, Attribute Grammars, and MSO Definable Tree Translations. *Information and Computation*, 154(1):34–91, October 1999.

[EM03a] Abdessamad Echihabi and Daniel Marcu. A Noisy-channel Approach to Question Answering. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, ACL '03, pages 16–23, 2003.

[EM03b] Joost Engelfriet and Sebastian Maneth. A Comparison of Pebble Tree Transducers with Macro Tree Transducers. *Acta Informatica*, 39(9):613–698, 2003.

[EM03c] Joost Engelfriet and Sebastian Maneth. Macro Tree Translations of Linear Size Increase are MSO Definable. *SIAM Journal on Computing*, 32(4):950–1006, 2003.

[EM06] Joost Engelfriet and Sebastian Maneth. The Equivalence Problem for Deterministic MSO Tree Transducers is Decidable. *Information Processing Letters*, 100(5):206–212, 2006.

[EMS09] Joost Engelfriet, Sebastian Maneth, and Helmut Seidl. Deciding Equivalence of Top–down XML Transformations in Polynomial Time. *Journal of Computer and System Sciences*, 75(5):271–286, August 2009.

[Eng75] Joost Engelfriet. Bottom-up and Top-down Tree Transformations - A Comparison. *Mathematical Systems Theory*, 9(2):198–231, 1975.

[Eng77] Joost Engelfriet. Top-down Tree Transducers with Regular Look-ahead. *Mathematical Systems Theory*, 10(1):289–303, 1977.

[Eng80] Joost Engelfriet. Some Open Questions and Recent Results on Tree Transducers and Tree Languages. In R.V. Book, editor, *Formal Language Theory; Perspectives and Open Problems*, pages 241–286. Academic Press, New York, 1980.

[Eng09] Joost Engelfriet. The Time Complexity of Typechecking Tree-walking Tree Transducers. *Acta Informatica*, 46:139–154, March 2009.

[ERS80] Joost Engelfriet, Grzegorz Rozenberg, and Giora Slutzki. Tree Transducers, L Systems, and Two-Way Machines. *Journal of Computer and System Sciences*, 20(2):150–202, 1980.

[ES77] Joost Engelfriet and Erik Meineche Schmidt. IO and OI. I. *Journal of Computer and System Sciences*, 15(3):328–353, 1977.

[ES78] Joost Engelfriet and Erik Meineche Schmidt. IO and OI. II. *Journal of Computer and System Sciences*, 16(1):67–99, 1978.

[Ési80] Zoltán Ésik. Decidability Results Concerning Tree Transducers I. *Acta Cybernetica*, 5(1):1–20, 1980.

[EV85] Joost Engelfriet and Heiko Vogler. Macro Tree Transducers. *Journal of Computer and System Sciences*, 31(1):71–146, 1985.

[EW67] Samuel Eilenberg and Jesse B. Wright. Automata in General Algebras. *Information and Control*, 11(4):452–470, 1967.

[FBCC+10] David A. Ferrucci, Eric W. Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John M. Prager, Nico Schlaefer, and Christopher A. Welty. Building Watson: An Overview of the DeepQA Project. *AI Magazine*, 31(3):59–79, 2010.

[FH07] Alain Frisch and Haruo Hosoya. Towards Practical Typechecking for Macro Tree Transducers. In *Proceedings of the 11th International Conference on Database Programming Languages*, DBPL '07, pages 246–260. Springer, Heidelberg, 2007.

[Fis68] Michael J. Fischer. *Grammars with Macro-like Productions*. PhD thesis, Harvard University, Massachusetts, 1968.

[Fri04] Alain Frisch. Regular Tree Language Recognition with Static Information. In *Programming Language Technologies for XML (PLAN-X)*, 2004.

[FRR+10] Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. Properties of Visibly Pushdown Transducers. In *Proceedings of the 35th International Conference on Mathematical Foundations of Computer Science*, MFCS '10, pages 355–367. Springer, Heidelberg, 2010.

[FSM10] Sylvia Friese, Helmut Seidl, and Sebastian Maneth. Minimization of Deterministic Bottom-up Tree Transducers. In *Proceedings of the 14th International Conference on Developments in Language Theory*, DLT '10, pages 185–196. Springer, Heidelberg, 2010.

[FSM11] Sylvia Friese, Helmut Seidl, and Sebastian Maneth. Earliest Normal Form and Minimization for Bottom-Up Tree Transducers. *International Journal of Foundations of Computer Science*, 2011. To appear.

[Fül81] Zoltán Fülöp. On Attributed Tree Transducers. *Acta Cybernetica*, 5:261–279, 1981.

[FV95] Zoltán Fülöp and Sándor Vágvölgyi. Attributed Tree Transducers Cannot Induce All Deterministic Bottom-up Tree Transformations. *Information and Computation*, 116(2):231–240, 1995.

[FV98] Zoltán Fülöp and Heiko Vogler. *Syntax-Directed Semantics: Formal Models Based on Tree Transducers*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[FV09] Zoltán Fülöp and Heiko Vogler. Weighted Tree Automata and Tree Transducers. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science, pages 313–403. Springer Berlin Heidelberg, 2009.

[FW04] David C. Fallside and Priscilla Walmsley. XML Schema Part 0: Primer Second Edition. W3C Recommendation, October 2004. Available at http://www.w3.org/TR/xmlschema-0/.

[Gaz06] Zsolt Gazdag. Decidability of the Shape Preserving Property of Bottom-up Tree Transducers. *International Journal of Foundations of Computer Science*, 17(2):395–414, 2006.

[Gie88] Robert Giegerich. Composition and Evaluation of Attribute Coupled Grammars. *Acta Informatica*, 25(4):355–423, 1988.

[GKM08] Jonathan Graehl, Kevin Knight, and Jonathan May. Training Tree Transducers. *Computational Linguistics*, 34(3):391–427, September 2008.

[Gol78] E. Mark Gold. Complexity of Automaton Identification from Given Data. *Information and Control*, 37(3):302–320, 1978.

[Gre78] Sheila A. Greibach. Hierarchy Theorems for Two-Way Finite State Transducers. *Acta Informatica*, 11:80–101, 1978.

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, Hungary, 1984.

[GS97] Ferenc Gécseg and Magnus Steinby. Tree languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, chapter 1, pages 1–68. Springer, Heidelberg, 1997.

[HFC05] Haruo Hosoya, Alain Frisch, and Giuseppe Castagna. Parametric Polymorphism for XML. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '05, pages 50–62. ACM Press, New York, 2005.

[HG01] Lynette Hirschman and Robert Gaizauskas. Natural Language Question Answering: The View from here. *Natural Language Engineering*, 7(4):275–300, December 2001.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, New York, second edition, 2001.

[HP02] Haruo Hosoya and Benjamin C. Pierce. Regular Expression Pattern Matching for XML. *Journal of Functional Programming*, 13(6):961–1004, 2002.

[HP03] Haruo Hosoya and Benjamin C. Pierce. XDuce: A Statically Typed XML Processing Language. *ACM Transactions on Internet Technology*, 3(2):117–148, 2003.

[Iro61] Edgar T. Irons. A Syntax Directed Compiler for ALGOL 60. *Communications of the ACM*, 4:51–55, January 1961.

[Kay03] Michael Kay, editor. XSL Transformations (XSLT) 2.0. W3C Working Draft, W3C, November 2003.
Available at http://www.w3.org/TR/xslt20.

[KB70] Donald E. Knuth and Peter B. Bendix. Simple Word Problems in Universal Algebras. In John Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[KG05] Kevin Knight and Jonathan Graehl. An Overview of Probabilistic Tree Transducers for Natural Language Processing. In Alexander F. Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing, 6th International Conference*, CICLing '05, pages 1–24. Springer, Heidelberg, 2005.

[KM02] Kevin Knight and Daniel Marcu. Summarization beyond sentence extraction: a probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1):91–107, July 2002.

[KMIS04] Christian Kirkegaard, Anders Møller, and Michael I. Schwartzbach. Static Analysis of XML Transformations in Java. *IEEE Transactions on Software Engineering (TSE)*, 30(3):181–192, March 2004.

[Knu68] Donald E. Knuth. Semantics of Context-Free Languages. *Mathematical Systems Theory*, 2(2):127–145, June 1968.

[Knu97] Donald E. Knuth. *Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd Edition)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1997.

[KS81] Tsutomu Kamimura and Giora Slutzki. Parallel and Two-Way Automata on Directed Ordered Acyclic Graphs. *Information and Control*, 49(1):10–51, 1981.

[LLN+11] Grégoire Laurence, Aurélien Lemay, Joachim Niehren, Slawomir Staworko, and Marc Tommasi. Normalization of Sequential Top-down Tree-to-word Transducers. In *5th International Conference on Language Automata Theory and Appliciations*, LNCS, Tarragona, Spain, 2011. Springer.

[LMN10] Aurelien Lemay, Sebastian Maneth, and Joachim Niehren. A Learning Algorithm for Top-down XML Transformations. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems of Data*, PODS '10, pages 285–296. ACM, New York, NY, USA, 2010.

[Löd11] Christof Löding. Basics on Tree Automata. In Deepak D'Souza and Priti Shankar, editors, *Modern Applications of Automata Theory*, volume 2 of *IISc Research Monographs*. World Scientific, 2011. To appear.

[Mal11] Andreas Maletti. *Survey:* weighted extended top-down tree transducers — part II: Application in machine translation. *Fundamenta Informaticae*, 2011. submitted.

[MBPS05] Sebastian Maneth, Alexandru Berlea, Thomas Perst, and Helmut Seidl. XML Type Checking with Macro Tree Transducers. In Chen Li, editor, *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '05, pages 283–294. ACM, New York, NY, USA, June 2005.

[MFS11] Sebastian Maneth, Sylvia Friese, and Helmut Seidl. Type-Checking Tree Walking Transducers. In Deepak D'Souza and Priti Shankar, editors, *Modern applications of automata theory*, volume 2 of *IISc Research Monographs*. World Scientific, 2011. To appear.

[MGHK09] Andreas Maletti, Jonathan Graehl, Mark Hopkins, and Kevin Knight. The Power of Extended Top-Down Tree Transducers. *SIAM Journal on Computing (SICOMP)*, 39:410–430, June 2009.

[MLM00] Makoto Murata, Dongwon Lee, and Murali Mani. Taxonomy of XML Schema Languages using Formal Language Theory. In *Extreme Markup Languages*, 2000.

[MM99] Inderjeet Mani and Mark T. Maybury. Advances in Automatic Text Summarization. pages 123–136. The MIT Press, 1999.

[MN99] Sebastian Maneth and Frank Neven. Structured Document Transformations Based on XSL. In *Revised Papers from the 7th International Workshop on Database Programming Languages: Research Issues in Structured and Semistructured Database Programming*, DBPL '99, pages 80–98. Springer-Verlag, 1999.

[MN04] Wim Martens and Frank Neven. Frontiers of Tractability for Typechecking Simple XML Transformations. In Alin Deutsch, editor, *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '04, pages 23–34. ACM Press, New York, 2004.

[MN05] Wim Martens and Frank Neven. On the Complexity of Typechecking Top-down XML Transformations. *Theoretical Computer Science*, 336(1):153–180, 2005.

[MN08] Sebastian Maneth and Keisuke Nakano. XML Type Checking for Macro Tree Transducers with Holes. In *Programming Language Technologies for XML (PLAN-X)*, 2008.

[Moh00] Mehryar Mohri. Minimization Algorithms for Sequential Transducers. *Theoretical Computer Science*, 234(1–2):177–201, 2000.

[MOS07] Anders Möller, Mads Østerby Olesen, and Michael I. Schwartzbach. Static Validation of XSL Transformations. *ACM Transactions on Programming Languages and Systems*, 29(4), 2007.

[MPS07] Sebastian Maneth, Thomas Perst, and Helmut Seidl. Exact XML Type Checking in Polynomial Time. In Thomas Schwentick and Dan Suciu, editors, *11th International Conference Database Theory (ICDT'07)*, volume 4353 of *LNCS*, pages 254–268. Springer, 2007.

[MS87] David E. Muller and Paul E. Schupp. Alternating Automata on Infinite Trees. *Theoretical Computer Science*, 54(2–3):267–276, 1987.

[MS99] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, June 1999.

[MS05] Anders Møller and Michael I. Schwartzbach. The Design Space of Type Checkers for XML Transformation Languages. In *Proc. 10th International Conference on Database Theory (ICDT '05)*, volume 3363 of *LNCS*, pages 17–36. Springer-Verlag, January 2005.

[MSV00] Tova Milo, Dan Suciu, and Victor Vianu. Typechecking for XML Transformers. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '00, pages 11–22. ACM, New York, NY, USA, 2000.

[MSV03] Tova Milo, Dan Suciu, and Victor Vianu. Typechecking for XML transformers. *Journal of Computer and System Sciences*, 66(1):66–97, 2003.

[Myh57] John Myhill. Finite Automata and Representation of Events. Technical Report WADC-TR-57-624, Wright-Patterson Air Force Base, Ohio, 1957.

[Ner58] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[Nev02] Frank Neven. Automata Theory for XML Researchers. *SIGMOD Record*, 31(3):39–46, September 2002.

[OGV93] Jose Oncina, Pedro García, and Enrique Vidal. Learning Subsequential Transducers for Pattern Recognition Interpretation Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458, May 1993.

[Per07] Thomas Perst. *Type Checking XML Transformations*. PhD thesis, Technische Universität München, Institut für Informatik, 2007.

[PS04] Thomas Perst and Helmut Seidl. Macro Forest Transducers. *Information Processing Letters*, 89(3):141–149, 2004.

[Rou70] William C. Rounds. Mappings and Grammars on Trees. *Mathematical Systems Theory*, 4(3):257–287, 1970.

[Sei92] Helmut Seidl. Single-Valuedness of Tree Transducers is Decidable in Polynomial Time. *Theoretical Computer Science*, 106(1):135–181, January 1992.

[Slu85] Giora Slutzki. Alternating Tree Automata. *Theoretical Computer Science*, 41(2-3):305–318, 1985.

[Tak75] Masako Takahashi. Generalizations of Regular Sets and Their Applicatin to a Study of Context-Free Languages. *Information and Control*, 27(1):1–36, 1975.

[Tar72] Robert Endre Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[Tha69] James W. Thatcher. Transformations and Translations from the Point of View of Generalized Finite Automata Theory. In *Proceedings of the 1st Annual ACM Symposium on Theory of Computing (STOC '69)*, pages 129–142. ACM, New York, NY, USA, 1969.

[Tha70] James W. Thatcher. Generalized^2 sequential machine maps. *Journal of Computer and System Sciences*, 4(4):339–367, 1970.

[Tha73] James W. Thatcher. Tree Automata: An Informal Survey. In Alfred V. Aho, editor, *Currents in the Theory of Computing*, pages 143–172. Prentice Hall, 1973.

[Vir80] János Virágh. Deterministic Ascending Tree Automata I. *Acta Cybernetica*, 5(1):33–42, 1980.

[Wu97] Dekai Wu. Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora. *Computational Linguistics*, 23(3):377–403, September 1997.

[Zac80] Z. Zachar. The Solvability of the Equivalence Problem for Deterministic Frontier-to-root Tree Transducers. *Acta Cybernetica*, 4:167–177, 1980.

Danke ...

Helmut Seidl für die Unterstützung und die vielen guten Ideen und Diskussionen.

Sebastian Maneth für die angenehme Zusammenarbeit und die hilfreichen Tipps von der anderen Seite der Erde.

Joachim Niehren für die äußerst interessanten und zielführenden Diskussionen sowie die detaillierten und umfangreichen Verbesserungsvorschläge.

Aurélien Lemay, merci pour notre discussion enrichissante à Lille.

Christian Pott, dass Du (als Nicht-Informatiker) die komplette Dissertation so detailliert Korrektur gelesen hast.

Alexander Herz für das ausführliche Korrekturlesen der Dissertation.

liebe Kollegen für die hilfreichen Diskussionen und, dass Ihr Euch immer wieder bereitwillig meine Vorträge über die Bäumchen angehört habt.

liebe Familie (Pott, Schulze-Eickenbusch und Friese) für Eure Unterstützung und Eure immer offenen Ohren.

Thomas Friese für Alles!