SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS

Catalin Ionut Tirnauca

Cătălin-Ionuţ Tîrnaucă

SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS

PhD Dissertation

Supervised by

Prof. Emeritus Magnus Steinby

Dr. María Dolores Jiménez López

Departament de Filologies Romàniques

UNIVERSITAT ROVIRA i VIRGILI

Tarragona,

2016

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

UNIVERSITAT ROVIRA i VIRGILI
DEPARTAMENT DE FILOLOGIES ROMÀNIQUES

Av. Catalunya, 35 (Campus Catalunya)
43002) Tarragona
Tel. +34 977 559 555
Fax +34 977 558 386
E-mail: secdfr@urv.cat
http://pizarro.fll.urv.es

I STATE that the present study, entitled "Syntax-directed translations, tree transformations and bimorphisms", presented by CATALIN IONUT, TIRNAUCA for the award of the degree of Doctor, has been carried out under my supervision at the Department of Romance Philology of this university and fulfils the requirements for the doctoral diploma with the European distinction.

Doctoral Thesis Supervisors

Prof. Emeritus Magnus Steinby

Turku, 27/11/2015

Dr. María Dolores Jiménez López

Tarragona, 23/11/2015

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

Contents

Acknowledgements **9**

List of Tables **11**

List of Figures **13**

1 Introduction **15**

2 Preliminaries **23**
2.1 Sets, relations and mappings . . . 23
2.2 Strings and languages . . . 24
2.3 Specifying languages . . . 26
2.3.1 Recognizers . . . 26
2.3.2 Generating devices . . . 26
2.3.3 The Chomsky hierarchy . . . 28
2.3.4 Decision problems . . . 30
2.4 Context-free languages . . . 30

3 Translations **35**
3.1 Basic notions . . . 35
3.2 Regular translations . . . 37
3.3 Syntax-directed translations . . . 40
3.3.1 Syntax-directed translation generating devices . . . 40
3.3.2 Recognizers of syntax-directed translations . . . 44
3.3.3 Properties of syntax-directed translations . . . 45
3.3.4 Hierarchies of syntax-directed translations . . . 50

4 Trees and Tree Languages **55**
4.1 Trees and contexts . . . 55
4.2 Tree homomorphisms . . . 58
4.3 Tree languages . . . 62
4.4 Recognizable tree languages . . . 63
4.4.1 Tree recognizers . . . 63
4.4.2 Regular tree grammars . . . 66
4.4.2.1 Tree substitution grammars . . . 68
4.4.3 Properties of recognizable tree languages . . . 70
4.5 Local tree languages . . . 70
4.6 Grammars and production trees . . . 72
4.7 Context-free tree languages . . . 74
4.8 A hierarchy of families of tree languages . . . 75

5 Tree Transformations **77**
5.1 Basic notions . . . 78
5.2 Tree transducers . . . 79
5.2.1 Top-down tree transducers . . . 80
5.2.2 Bottom-up tree transducers . . . 83
5.2.3 Some special tree transformation classes . . . 85
5.2.4 Extended top-down tree transducers . . . 86
5.2.5 Properties and hierarchies of tree transducers . . . 88
5.3 Synchronous tree grammars . . . 91
5.3.1 Synchronous tree substitution grammars . . . 92
5.3.2 Generalized synchronous tree substitution grammars . . . 96

6 Tree Bimorphisms and Translations **101**
6.1 Basic notions . . . 101
6.2 Quasi-alphabetic tree bimorphisms . . . 103
6.2.1 Properties . . . 106
6.2.2 Translation power . . . 111
6.2.3 Implementation by tree transducers . . . 122
6.3 Alphabetic and permuting tree bimorphisms . . . 128
6.3.1 Properties . . . 128
6.3.2 Translation power . . . 129
6.3.3 Implementation by tree transducers . . . 130
6.4 Linear non-deleting tree bimorphisms . . . 131
6.5 Fine tree bimorphisms . . . 135
6.6 Comparisons and relations between tree bimorphisms . . . 138
6.7 Other types of tree bimorphisms . . . 140
6.7.1 Miscellaneous . . . 141
6.7.2 $\mathbf{\Sigma}$-rational and $\mathbf{\Sigma}$-algebraic tree transformations . . . 143
6.7.3 Extended bimorphisms: magmoids and bitransformations . . . 146

7 Synchronous Translation and Tree Transformation Generators **149**
7.1 Synchronous translation generators . . . 149
7.2 Synchronous tree transformation generators . . . 155

8 Concluding Remarks, Further Topics and Bibliographic Notes **159**
8.1 A summary of the main results . . . 159
8.2 Topics to be considered . . . 159
8.2.1 Future work on Section 5.3.2 . . . 159
8.2.2 Future work on Chapter 7 . . . 161
8.3 Bibliographic notes . . . 163
8.3.1 On Chapter 2 . . . 163
8.3.2 On Chapter 3 . . . 164
8.3.3 On Chapter 4 . . . 165
8.3.4 On Chapter 5 . . . 167

Bibliography **171**

List of Notation **203**
Acronyms 203
Devices 203
Greek alphabet 203
Language classes 204
Miscellaneous 204
Operations, relations and sets related to strings 205
Operations, relations and sets related to trees 205
Roman alphabet 206
Translation classes 207
Tree bimorphism classes 207
Tree homomorphism classes 208
Tree transformation classes 208

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

Acknowledgements

Magnus Steinby introduced me to the tree world back in 2005 during his course "Tree automata" given to the 4th International PhD School on Formal Languages and Applications at GRLMC in Tarragona. He is the supervisor every PhD student deserves: the perfect combination of astute scientist, invaluable advisor, deft proof-reader, deep listener and lovely human being who always freely shares his wisdom about the field. Moreover, he has been a cheerleader and good friend from start to finish. Actually, words are not enough to express my deep gratitude and admiration for him.

I would also like to thank with all my heart to my dear friend and co-author Andreas Maletti for getting interested into quasi-alphabetic tree bimorphisms and being my longtime source for a consistent reality check. I will always remember his mentoring and contribution to the advances of my research together with the long talks and constructive criticism.

Additionally, none of this would have been possible without the financial support of the one-year scholarship ref. no. 2007BRDI/06-11 provided by Rovira i Virgili University. I would also like to acknowledge the financial support of the European Science Foundation, under the AutoMathA Programme, for sponsoring my two-month stay at the Department of Mathematics of the University of Turku, Finland, in 2008, the two-week visit to the Department of Mathematics of the Aristotle University of Thessaloniki, Greece, in 2009, and my participation in the WATA conferences in Leipzing in 2006 and 2010 and Dresden in 2008. Further, I am grateful to the "Summer 2008 Internship on Natural Language Processing" at the University of Southern California, Information Science Institute (USC/ISI), Marina del Rey, CA, USA, when I worked with Kevin Knight on the project "New Automata for Natural Language" as part of the TREEWORLD. Moreover, I am grateful to the Spanish Ministry of Education that through the project "Automata, Languages and Formal Logic" (ref. MTM2007-63422) of "Plan Nacional de Investigación Científica" supported my participation in the international scientific meetings NCMA 2009, CAI 2009, and CIAA 2010. Also, they have provided me with a 3 month mobility grant "Movilidad de estudiantes para la obtención de la Mención Europea en el Título de Doctor" through which I visited the University of Turku in 2009. In addition, I acknowledge the support of the "Programa de Foment de la Recerca de la URV, Eix A. Ajuts per realitzar estades europees" of Rovira i Virgili University that financed a 2-week visit to the University of Turku in 2011. I also express my gratitude to Prof. Masami Ito for inviting me in 2006 to the Department of Mathematics of Sangyo University in Kyoto, Japan, where I gave one of my first talks ever: "Syntax-Directed Translation Schemata and Tree Bimorphisms". I further acknowledge my current work place, the University of Cantabria, for allowing me the use of its facilities. I am very grateful for the excellent individual study conditions of the Emilio Botín Library of the university. Moreover, this manuscript would never be if it was not for the office computer that help me with all the compiling when my own computer broke.

This manuscript would not have been possible without the contribution of so many other people: Carlos Martín Vide, the team members of the GRLMC research group and the professors of our 4th PhD School for constant advice and support - I learnt so much from all of you; Kevin Knight and the "tree people" in his research group at the USC/ISI, John May and Steve DeNeefe, who hosted me as an intern back in 2008 and infused me with

a totally different viewpoint; Heiko Vogler and Manfred Droste for providing me in the beginning with papers to read and supporting my attendance to WATA Conferences in Dresden and Leipzig; George Rahonis for being an excellent host and sharing with me various research topics related to the connection between tree bimorphisms and tree transducers; Zoltán Fülöp, Mark-Jan Nederhof, Giorgio Satta, and many other colleagues interested in trees for their continuous advice, interminable discussions and good time; Gemma Bel Enguix and Loli Jiménez for advices and including me in the project "Automata, Languages and Formal Logic"; Lilica Voicu for encouragement and helping me with all the bureaucratic matters; MATESCO Department of the University of Cantabria, especially to my dear friends Domingo Gómez, Carlos Beltran, Pepe Montaña, Rafa Duque, Dani Sadornil, and Pablo Sánchez for adopting me since November 2009 and always encouraging me; my colleagues from the European Projects Office of the University of Cantabria, Ruth Arroyo, Rim Bouzgarrou, and Juanjo San Miguel, for their kindness and patience; my close friends Areti Panou, Kim Solin, Amittai Axelrod, José Luís 'balqui' Balcázar, Georgiana Stoica, Mihai and Raluca Ionescu, 'Dl.' Dan and Simona Libotean, Robert 'robynete' Mercaş, Radu 'răducu' Atanasiu, Florin 'melcu' Sora, Mădălin 'MD' Bălan, Că tălin 'dintzi' Topoloiu, Renato 'renny' Ridiche, Andrei 'pac' Partenie, Adrian Frumuşelu and Dragoş 'mastercyb' Deliu for pursuing me to finish the PhD and always offering to help. My deep appreciation to all of you!

My mom, Suzana, who always encouraged me to study and sacrificed so much for me to be able to do that. Thank you for your constant love and patience, and, most of all, for the way you have raised me.

My children, Marc and Rebeca, who would make any parent proud. Thank you for your understanding and for transforming my life the way you did. And Marc, I truly hope that one day I would be able to better answer the question you addressed me in the final days of the preparation of the manuscript: "Daddy, why do you need to finish this thesis NOW instead of playing with me?"

My wife, Cristina, who remains my greatest source of love, smile, kindness, support and encouragement. Without you, this thesis would never have been written. From all my heart, thank you for every single day!

Finally, I would like to dedicate this work to the memory of two persons that stayed too little in this world, but yet influenced me so much: my father Ilie, who was my best friend as a kid but could never see me grow into a real man, and my grandmother Florica, who suffered so much when I left Romania 11 years ago. I hope you both are proud of me, wherever you are!

MAMA, sper că ştii cât de mult te iubesc şi că nu te voi uita niciodată! Îţi sunt atât de recunoscător pentru felul în care m-ai crescut şi pentru toate sacrificiile pe care le-ai făcut pentru ca mie să nu îmi lipsească nimic şi astfel să mă pot dedica mereu studiilor. Nona Coca, vă mulţumesc pentru ajutorul dat în momentele grele şi pentru că m-aţi suportat în toţi aceşti ani! Familiei mele, sunt mândru de voi şi sper că şi voi de mine! Iar tuturor prietenilor mei dragi de 'acasă' (Dintzi, Pac, Melcu, MD, Rombacu, Moshu, Cip, Mariachi, Danezu şi mulţi mulţi alţii), nu pot decât să le mulţumesc din suflet pentru încurajări şi pentru că mi-aţi rămas mereu aproape!

List of Tables

2.3.1 Main classes of languages and their specification methods. 29

5.2.1 Overview of formal properties of various TOP-, BOT- and XTT- transducers (+ stands for 'the property holds' and - means 'the property does not hold'). . . . 91

8.1.1 Weak and strong equivalences of classes of tree transformations and translations defined by tree bimorphisms, synchronous grammars or tree transducers. Properties of such classes – a summary of results. 160

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

List of Figures

2.3.1 The Chomsky hierarchy. 30
2.4.1 Derivation tree associated to the production $S \to \texttt{if}\ S\ \texttt{else}$ (a) and to the terminal string `if if else` (b). 32

3.3.1 HASSE diagram of various types of syntax-directed translations. 54

4.1.1 The ΣX-context $f(g(x), \xi)$ (on the right) and the ΣX-tree represented by $f(g(x, f(e, x)))$ (on the left). The alphabets are given in Example 4.1.3. 56
4.2.1 Application of various tree homomorphisms. 61
4.8.1 HASSE diagram representing an hierarchy of well-known families of tree languages, where $\mathrm{DRec} \cap \mathrm{Loc}^1$ denotes the class of all dT-recognizable local tree languages with exactly one root symbol. 75

5.2.1 A sample rule in a TOP-transducer and an illustration of a derivation step using that rule. 81
5.2.2 HASSE diagram representing the inclusion relations among some top-down and bottom-up tree transformations. 89
5.2.3 HASSE diagram representing the inclusion relations between the classes of tree transformations computed by extended top-down tree transducers. 90

6.1.1 A pictorial representation of a tree bimorphism (φ, R, ϕ). 102
6.2.1 Illustration of the construction in Theorem 6.2.9. 107
6.2.2 An element in the tree transformation $\tau(TB_{SC})$ defined by the SCFG SC of Example 6.2.26: the tree $t\psi$ in $T_{\Sigma^{\mathrm{out}}}(Y)$ (down) is a transform of the tree $t\varphi$ in $T_{\Sigma^{\mathrm{in}}}(X)$ (middle) for the production tree t in $P(SC)$ (up). 119
6.2.3 HASSE diagram representing the inclusion relations among well-known classes of tree transformations computed by tree transducers and quasi-alphabetic tree transformations. 127
6.6.1 HASSE diagrams of classes of tree transformations defined by tree bimorphisms and tree transducers (a), and corresponding translations (b). 140
6.7.1 Example of the canonical representation of rational relations of binary trees used by Takahashi (1977). By applying the projections on the first and second component, respectively, on the balanced tree on the left, we get the two trees on the right. 141

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

Chapter 1
Introduction

"Translation is like a woman. If it is beautiful, it is not faithful. If it is faithful, it is most certainly not beautiful."
Yevgeny Yevtushenko

Automatic translation has been brought to the attention of researchers long before computers were even invented. The first ideas can be traced back to the 17th century when Descartes envisioned the existence of a universal language in a form of a collection of ciphers: the lexical equivalents of a word in all known languages would be given the same code number. Translation between any two languages would then be possible by using this "mechanical dictionary". Actual examples of such dictionaries were published by Cave Beck in 1657[1], by Johann Joachim Becher in 1661[2], and by Athanasius Kircher in 1663[3]. These ideas could be seen as genuine forerunners of *machine translation* (MT), although the first explicit proposals for "translating machines" did not appear until 1933, when two patents for mechanical dictionaries were issued independently in France and Russia. In fact, the idea of using a "unique language" metamorphoses into the "center language" of a *bimorphism*, which is a mathematical mechanism that has been successfully used in theorem proving for MT systems (and to which we shall dedicate a chapter of this dissertation).

The first mention of computer translation is credited to Warren Weaver. On March 4, 1947 he wrote to Norbert Wiener: "Recognizing fully, even though necessarily vaguely, the semantic difficulties because of multiple meanings, etc., I have wondered if it were unthinkable to design a computer which would translate". Wiener's response on April 30 disappointed Weaver: "I frankly am afraid the boundaries of words in different languages are too vague and the emotional and international connotations are too extensive to make any quasi mechanical translation scheme very hopeful". Nevertheless, Weaver did not give up, and his article[4] written on July 15, 1949 is credited as being the document that had the most widespread and profound influence in the history of MT: it brought the idea of MT to general notice and inspired many projects. It is worth mentioning that the possibility of language universals was one of the key points raised by the memorandum.

Early work in MT had a very simplistic view: the only differences between languages resided in their vocabularies and the permitted word orders. It was Chomsky (1957) and his much celebrated *Syntactic Structure* book who revolutionized the field with a hierarchy of generative grammars (with their corresponding language classes), that established a rule

[1] *The Universal Character, by which all Nations in the World may understand one another's Conceptions, Reading out of one Common Writing their own Mother Tongues. An Invention of General Use, the Practise whereof may be Attained in two Hours' space, Observing the Grammatical Directions. Which Character is so contrived, that it may be Spoken as well as Written.* London, 1957.

[2] *Character pro notitia linguarum universali.*

[3] *Polygraphia Nova et universalis ex combinatoria arte detecta.* Rome, 1663.

[4] *Translation memorandum.* The Rockefeller Foundation, 1949. Available at `http://www.mt-archive.info/Weaver-1949.pdf`.

based system of syntactic structures. In this hierarchy, *context free grammars* (CFGs) and their corresponding language class, the *context free languages* (CFLs), turned out to have a very important role for natural language processing (NLP), and in particular, for MT. Further research revealed great generality, mathematical elegance, and wide applicability of generative grammars.

Unfortunately, the syntactic analysis of phrases in a natural language also showed the depth of ambiguity in English, for example:

> *"Time flies like an arrow" may seem fairly straightforward to us, but a machine sees a number of other possibilities, for example "Time the speed of flies as quickly as you can" ("time" being interpreted as a verb rather than a noun) and "Certain flies enjoy an arrow" ("time" being interpreted as an adjective, and "like" being interpreted as a verb). The machine could be instructed to rule out these particular offbeat parsings, but how would it handle the sentence, "Fruit flies like bananas"?* **Burck (1965)**

After more than a decade of enthusiastic research on MT, the ALPAC[5] report concluded in 1966 that the results obtained in this period had failed to fulfill the expectations, and therefore recommended against funding further research in MT. The field regained its luster in the late 1980s with the introduction of statistical machine learning tools, the increase of computational power, and the availability of multilingual textual corpora. Nevertheless, one cannot base an MT system exclusively on statistics, as Chomsky (1957) demonstrated with this famous example:

> 1. *Colorless green ideas sleep furiously.*
> 2. **Furiously sleep ideas green colorless.*
>
> *It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse. Hence, in any statistical model for grammaticalness, these sentences will be ruled out on identical grounds as equally "remote" from English. Yet (1), though nonsensical, is grammatical, while (2) is not grammatical.* **Chomsky (1957)**

Most modern translation systems are a mix of syntax-directed and statistical based translations. As the title suggests, this work is primarily concerned with the former one.

Although we are used to thinking about "translation" in the context of natural languages, the word itself means a correspondence between any two strings in different languages that have the same meaning. These may occur, for example, in computer science in the transcription of a program written in a high level programming language into the corresponding machine code, or in biology in the process by which messenger RNA directs the amino acid sequence of a growing polypeptide during protein synthesis.

Formally speaking, *translations* are sets of pairs of strings, where a string is a finite sequence of symbols from a finite alphabet. The input string of a translation often must have a certain structure. For example, the input strings of a natural language translation must be correct utterances of the input language. An intuitive way to describe the structure of a sentence is with a *derivation tree* (Aho and Ullman, 1972, Section 2.4.1) or a *production*

[5] Automatic Language Processing Advisory Committee of the National Academy of Science - National Research Council.

tree (Engelfriet, 1975c, Gécseg and Steinby, 1984). These show how to derive the sentence using grammatical rules. The translation process can then be done on the "structure level" by relating derivation trees of the input language to derivation trees of the output language. Thus, a *tree transformation* is defined as this relation between input and output derivation trees. The translation pair is found by taking the yields of the trees, i.e., the sequences of labels of the leaves, read from left to right.

The notion of translation appeared in computer science at the same time as programming languages (FORTRAN - 1957, ALGOL - 1958, COBOL - 1959) and their compilers. In particular, a notational variation of CFGs – Backus-Naur form – was used to define the syntax of ALGOL by Backus et al. (1960, 1963). CFGs are still employed nowadays for defining the syntax of many programming languages. Compilers used to be large hand-written programs, and extremely difficult to modify because it was hard to identify those parts which would be affected by a change in the source code. For this reason, Irons (1961), followed by Barnett and Futrelle (1962) and Čulík (1965), proposed to separate the syntactic description of a programming language (input) from its semantics (translation into output), and to use CFGs to represent the former.

The core principle of a *syntax-directed translation* (SDT) is that the meaning of an input is related to its syntactic structure. Čulík (1965), Lewis II and Stearns (1968), and Aho and Ullman (1972) were among the first scientists who defined SDTs by means of synchronous grammars. This syntax-directed translation schemata can be seen as a CFG with translation elements attached to each production. Whenever a production is used in a derivation of an input string, the associated translation element generates a part of an output string. Since then many devices defining SDTs have been introduced and investigated, some of them being exhibited in this manuscript.

Any SDT can be viewed as a three-step process (Maneth, 2004, Figure 3): first, parse the input string and obtain a derivation tree, then perform a tree transformation (derivation trees of the input grammar are transformed into derivation trees of the output grammar), and finally, output the yield of the transformed tree. In particular, this allows to design parsing algorithms that check whether a program is syntactically correct. If the program is correct, a derivation tree is returned as a convenient structural representation of the program. Then the tree transformation (i.e., syntax-directed translation device) turns this parse tree into a representation of the compiled program (for example, a program in machine code or pseudo-code). This further exemplifies the natural use of SDTs in program compilation.

Chomsky (1957) used trees for representing the syntax of sentences, and the connection between finite tree automata and context-free languages was shown in some of the very first papers on tree automata by Thatcher (1967) and Doner (1970). Moreover, Rounds (1970a) introduced tree transducers explicitly as models of Chomsky's transformational grammars:

> *Recent developments in the theory of automata have pointed to an extension of the domain of definition of automata from strings to trees ... Why pursue such a generalization? ... because parts of mathematical linguistics can be formalized easily in a tree-automaton setting. The theories of transformational grammars and of syntax-directed compilation are two examples.* **Rounds (1970a)**

However, although the theory of tree automata, tree languages and tree transformations has grown in the past four decades into a rich and well-founded discipline (check Thatcher,

1973, Gécseg and Steinby, 1984, Gécseg and Steinby, 1997, Fülöp and Vogler, 1998, Comon et al., 2007, for expositions and further references), it is only recently that researchers have more extensively turned to tree-based approaches for NLP (see Knight, 2007, Knight and Graehl, 2005, May, 2010, Maletti et al., 2009, Maletti, 2010a, for overviews).

Compiler design and machine translation, or, as Bar-Hillel (1964) used to call it, fully automatic high quality translation are fields in which accuracy is an important issue, so the whole theory should rely on a solid mathematical background. Among the most desirable properties of syntax-based systems defining tree transformations are (see Knight, 2007, Maletti, 2010a, for example):

- EXPR: The model is suitably expressive for application in machine translation meaning it has the ability to handle local rotations, which are re-orderings of parts of sentences that come up in translations between different languages. Abeillé et al. (1990), Yamada and Knight (2001), and Chiang and Knight (2006) give some examples of such local rotations.

- SYM: The model is symmetric, which means that for each tree transformation definable by the model, its inverse tree transformation can also be defined. This property allows us to invert a tree transducer that translates Romanian to Spanish sentences to produce a translator from Spanish to Romanian.

- PRES and PRES^{-1}: The tree transformations defined by the model preserve regularity of tree languages and so do their inverses. In other words, the property demands that the image and pre-image of a regular tree language remain regular under the tree transformation. In practice, such properties are also called forward and backward applications (see May, 2010, for example).

- COMP: The class of tree transformations defined by the model is closed under composition. In other words, it means that complex systems can be constructed by compositions of simpler ones. An example of how such a property improves the quality of translations is shown by Tîrnăucă (2008, Section 5). Having two small fragments of Romanian-to-English and English-to-Spanish translations defined by syntax-directed translation schemata and using Theorem 7.4 of Steinby and Tîrnăucă (2009), a direct and correct translation from Romanian to Spanish is obtained.

Synchronous grammars consist of two formal grammars whose productions are linked by some mechanism. These two grammars can easily model syntax-sensitive transformations because the derivation on either side (assume input, without loss of generality) can be seen as a syntax tree of the sentence it generates. Thus, synchronous grammars can describe local rotations commonly used in phrase-based machine translation, where a phrase is any part of the input sentence. Moreover, the links can communicate information about the shape of the input parse tree to the output side, but the mechanism is limited by the requirement that the input and output trees have similar (or matching) shapes (Chiang and Knight, 2006, Chiang, 2006, Satta, 2004, 2009). Consequently, properties that may significantly improve the translation process like closure under composition and preservation of recognizability of tree languages are difficult to prove for such formalisms and hence were largely unknown in the linguistics community (cf. Shieber, 2004, Chiang and Knight, 2006, for example).

A *tree transducer* dynamically transforms an input tree into the output tree using a finite state control mechanism. In the last decade, new types of tree transducers were used with

considerable success in modeling translations between natural languages especially because of their ability to capture syntax-sensitive transformations and complex re-orderings of the syntax trees of sentences. Those tree transducers are now an essential device in the field of syntax-based machine translation (see Knight and Graehl, 2005, Graehl et al., 2008, Knight, 2007, Maletti, 2010a, and the references therein). Unfortunately, closure under composition and preservation of recognizable tree languages do not hold in general for most of the main tree transducer types (Baker, 1979, Engelfriet, 1975c, Gécseg and Steinby, 1984, Maletti et al., 2009), which shows that the added power comes with severe drawbacks.

A *tree bimorphisms* offers an elegant algebraic way to define tree transformations by being formed by two tree homomorphisms defined on the same common tree language called center. Such a formalism was used with considerable success in proving properties like closure under composition and preservation of recognizability by imposing suitable restrictions on its constituents (Takahashi, 1972, 1977, Arnold and Dauchet, 1982, Steinby, 1986, Bozapalidis, 1992, Steinby and Tîrnăucă, 2007). Moreover, by taking the yields of the input and output trees, they can be seen as devices that generate string-to-string relations. Dauchet and Tison (1992) and Raoult (1992), and to some extent Maletti (2010a) and Tîrnăucă (2008), briefly surveyed the main classes of tree bimorphisms and their main characteristics.

Using the tree bimorphism formalism, Shieber (2004) was the first one who linked tree transducers and synchronous grammars in an attempt to improve the mathematical framework of the latter devices:

> *The bimorphism characterization of tree transducers has led to a series of composition closure results. Similar techniques may now be applicable to synchronous formalisms, where no composition results are known.* **Shieber (2004)**

Immediately a series of such new results emerged: Shieber (2006), Huang et al. (2006), Maletti (2007, 2008, 2010a), Steinby and Tîrnăucă (2007, 2009), Maletti and Tîrnăucă (2009, 2010), Tîrnăucă (2011), Tîrnăucă (2009, 2007), Tîrnăucă (2008), and Nederhof and Vogler (2012).

In this manuscript we give a comprehensive study of the theory and properties of SDT systems seen from these three very different perspectives that perfectly complement each other: as generating devices (synchronous grammars), as acceptors (transducer machines) and as algebraic structures (bimorphisms). These systems are investigated and compared both as tree transformation and translation defining devices. We focus on tree bimorphisms given their recent applications to NLP, and we propose a more complete and up-to-date overview on tree transformations classes defined by tree bimorphisms, linking them with well-known types of synchronous grammars and tree transducers. Moreover, we prove or recall most desired properties such classes possess improving thus the mathematical knowledge on synchronous grammars and tree transducers as Shieber (2004) suggested. Also, by means of Hasse diagrams, we clearly show for the first time the inclusion relations between the main classes of tree bimorphisms both on string level (as translation devices) and on tree level (as tree transformation mechanisms). In addition, we further exhibit how to extend previous results to more general classes of tree bimorphisms and synchronous grammars.

The work is structured as follows, the main results being outlined in Table 8.1.1.

Chapter 2 fixes the general notation to be used later and reviews some basic notions concerning formal language theory such as set theory, strings, languages, and the Chomsky

hierarchy (HASSE diagram of Figure 2.3.1). Special attention is paid to CFLs and their properties in Section 2.4, as they are intensively used in Chapters 3, 4, and 6, for example.

Chapter 3 establishes the general notation and terminology related to the translation theory (Section 3.1), and surveys two of the most investigated classes of translations: *regular translations* (Section 3.2) and *syntax-directed translations* (Section 3.3). We give uniform definitions for various subclasses of syntax-directed translations, whose usefulness is mentioned in Section 8.3.2. *Linear syntax-directed translations* is such a subclass that only recently received an increased interest in the NLP community because of its application in word alignment. In Theorem 3.3.15(iii) we give an alternative characterization of linear syntax-directed translations by string bimorphisms. This new result can be further used to prove mathematical properties of linear syntax-directed translations. For example, Proposition 3.3.17 elegantly shows a few composition results between various types of syntax-directed translations. Finally, in Section 3.3.4, we study the connections between the different types of syntax-directed translation defining devices introduced in Definition 3.3.3. The results are summarized in the HASSE diagram of Figure 3.3.1.

Chapter 4 is a gentle introduction to the theory of tree languages and tree automata. The focus is on *recognizable tree languages* and their specification methods (Section 4.4) as they form the core of the whole dissertation. We briefly present other well-known classes of tree languages, too: *deterministic recognizable tree languages* (Section 4.4.1), *tree languages generated by tree substitution grammars* (Section 4.4.2.1), *local tree languages* (Section 4.5), and *context-free tree languages* (Section 4.7). The inclusion relations among them are established by the HASSE diagram of Figure 4.8.1. Moreover, in Section 4.6, a special attention is paid to *production trees* and connections between recognizable tree languages and context-free languages. This concept will be extended and frequently used later in Chapter 6.

As an aside, we mention the uniform presentation and comparison between all common tree homomorphism classes to be found in the literature (Section 4.2), and the definition of *tree substitution grammars*, which is essential in getting new characterizations of syntax-directed translations in Sections 5.3 and 6.4.

Chapter 5 surveys the main tree transducer types in the literature: *top-down tree transducers* (Section 5.2.1), *bottom-up tree transducers* (Section 5.2.2), and *extended top-down tree transducers* (Section 5.2.4), together with their formal properties (Table 5.2.1) and the inclusion relations between the classes of tree transformations defined by them (Figures 5.2.2 and 5.2.3). In Section 5.3, we present two synchronous grammars that generate pairs of trees, widely used to model linguistic phenomena encountered in machine translation, language interpretation, and natural language generation: *synchronous tree-substitution grammars* (Section 5.3.1) and *generalized synchronous tree-substitution grammars* (Section 5.3.2), also known in the literature as synchronous tree-substitution grammars with states. Using our formal definition, we get new characterizations of syntax-directed translations (Proposition 5.3.3 and Corollary 5.3.9). In addition, we give a direct and effective characterization in terms of generative devices of (linear non-deleting) extended top-down tree transducers, which was differently proved by Maletti (2008).

Chapter 6 presents in detail some classes of tree transformations defined by means of tree bimorphisms that have good properties such as SYM, PRES, PRES^{-1}, and COMP, together with their connection with synchronous grammars and tree transducers: *quasi-alphabetic tree transformations and translations* of Steinby and Tîrnăucă (2007, 2009) in Section 6.2, *primitive transformations* of Takahashi (1972) in Section 6.3, *linear complete bimorphisms* of Arnold and Dauchet (1976a, 1982) in Section 6.4, and *alphabetic tree rela-*

tions of Bozapalidis (1992) in Section 6.5. Moreover, the HASSE diagram of Figure 6.6.1 shows for the first time the inclusion relations between these classes on both the tree and string level. Furthermore, Section 6.7 presents an up-to-date overview of other less-known classes of tree transformations defined by means of tree bimorphisms that share surprising properties.

In **Chapter 7**, two new types of synchronous grammars are defined for the first time. *Synchronous translation generators* are introduced in Section 7.1 as a class of translation-defining devices that generalize synchronous context-free grammars, and hence the basic syntax-directed translation schemata studied in Section 3.3.1. They also generalize the syntax-connected transduction schemes of Schreiber (1975, 1976) but are essentially equivalent in power with these. In addition, Section 7.2 considers *synchronous tree transformation generators* that in a natural way correspond to synchronous translation generators and generalize the synchronous tree-substitution grammars and generalized tree-substitution grammars, that were presented in Section 5.3.

Chapter 8 presents a summary of the classes of translations and tree transformations that we have investigated in this thesis, along with some of their main properties (Table 8.1.1). In Section 8.2, we propose to the reader a series of future work ideas related to Section 5.3.2 and Chapter 7. Moreover, Section 8.3 exhibits a set of bibliographic notes for the previous chapters that hopefully provide inspiration for further reading. They primarily contain the very first papers on the topic and recent advances (up to 2014) of related formalisms that could not be covered in this work. The idea was also to point out possible applications of the notions introduced. The literature on formal language theory and natural language processing is extensive and we have generally limited the bibliography to papers which we have read and found interesting or useful. To ease keeping track of the cited items and their use in the thesis, the bibliography contains a reference to the places where each paper has been mentioned. We apologize in advance to those who may feel that their work was inadequately referenced or omitted, and hope they understand the limitations of both space and time.

Moreover, the dissertation makes use of many notions and devices from both linguistics and formal language theory defined over strings as well as on trees. The appendix **List of Notation** summarizes all the symbols used in the manuscript together with the place where they were defined for the first time.

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

CHAPTER 2

Preliminaries

Although we assume that the reader is familiar with the theory of formal languages, we review some basic notions and fix the general notation to be used throughout the monograph. We borrowed most of them from Aho and Ullman (1972), Linz (2001), Hopcroft et al. (2001), and Rozenberg and Salomaa (1997) but also other textbooks have been consulted: Salomaa (1973), Harrison (1978), Sudkamp (1997), Sipser (1997) and Taylor (1998).

2.1 Sets, relations and mappings

We may express the fact that P is defined to be Q by writing $P := Q$. The basic set-theoretical symbols $\cap, \cup, \times$, etc. have their usual meanings. Nevertheless, we indicate some conventions regarding the notation used.

(i) $\emptyset$ denotes the *empty set*, i.e., the unique set having no elements.

(ii) If a set X is a *subset* of Y and Y is a *superset* of X, we write $X \subseteq Y$. The *proper inclusion* is denoted by $X \subset Y$.

(iii) We write $X \parallel Y$ if two sets X and Y are *incomparable*.

(iv) The *difference* of the sets X and Y is denoted by $X \setminus Y$.

(v) X^C denotes the *complement* of the set X with respect to a given superset (universe).

(vi) The *power set* of a set X, written $\wp(X)$, is the set of all subsets of X. Moreover, $\wp_F(X)$ denotes the set of finite subsets of X.

(vii) $|X|$ denotes the *cardinality* of the set X.

(viii) The union of a family $(X_j)_{j\in J}$ of some sets (indexed by J) is written as $\bigcup_{j\in J} X_j$. Similarly, $\bigcap_{j\in J} X_j$ is their intersection.

(ix) We often write x for the one-element set $\{x\}$.

(x) For any $n \geq 0$, $X^n := \underbrace{X \times X \ldots \times X}_{n \text{ times}}$.

The numbers we deal with here are always non-negative integers. When we write "... for all $m \geq 1$..." we mean "... for all integers $m \geq 1$...". Let $\mathbb{N} := \{0, 1, 2, \ldots\}$ and $\mathbb{N}_+ := \mathbb{N} \setminus \{0\}$ be the sets of all the natural numbers and the positive integers, respectively. For any $n \in \mathbb{N}_+$, let $[n]$ denote the set $\{i \mid 1 \leq i \leq n\}$.

Let X, Y, and Z be sets, and consider a (binary) *relation* $\rho \subseteq X \times Y$. The fact that $(x, y) \in \rho$ for some elements $x \in X$ and $y \in Y$ is also expressed by writing $x\rho y$. For any $x \in X$, let $x\rho := \{y \mid x\rho y\}$. More generally, for any $X' \subseteq X$, $X'\rho$ is the set of all

$y \in Y$ such that $(x, y) \in \rho$ for some $x \in X'$. The *converse*, or *inverse* of ρ is the relation $\rho^{-1} := \{(y, x) \mid (x, y) \in \rho\}$ $(\subseteq Y \times X)$. The *domain* of ρ is the subset $\mathrm{Dom}(\rho) := Y\rho^{-1}$ of X, and its *range* is the subset $\mathrm{Range}(\rho) := X\rho$ of Y. The *composition* of two relations $\rho \subseteq X \times Y$ and $\rho' \subseteq Y \times Z$ is the relation

$$\rho \circ \rho' := \{(x, z) \mid x \in X, z \in Z, (\exists y \in Y)(x, y) \in \rho, (y, z) \in \rho'\} \ .$$

Often we write $\rho\rho'$ for $\rho \circ \rho'$. The *total relation* of a set X is $\mathrm{tot}_X := X \times X$, and the *identity relation* $\{(x, x) \mid x \in X\}$ of a set X is denoted by id_X. The *reflexive transitive closure* of a relation $\rho \subseteq X \times X$ is $\rho^* := \bigcup_{i \geq 0} \rho^i$, where $\rho^0 := \mathrm{id}_X$ and $\rho^i := \rho^{i-1} \circ \rho$ for every $i \in \mathbb{N}_+$. Then, the *transitive closure* of ρ is $\rho^+ := \bigcup_{i \geq 1} \rho^i = \rho\rho^*$.

A (*total*) *mapping* ϕ from a set X to a set Y is a relation $\phi \subseteq X \times Y$ such that for every $x \in X$ there is exactly one y in Y satisfying $x\phi y$. If we want to emphasize that ϕ may be not defined for all $x \in X$, then we shall say that ϕ is a *partial* mapping from X to Y. In either case, we write $\phi\colon X \to Y$. If $x\phi y$ for some $x \in X$ and $y \in Y$, then y is called the *image* of x and x a *pre-image* of y. This is expressed by writing $\phi(x) = y$, $x\phi = y$ or $\phi\colon x \mapsto y$. Moreover, for any $X' \subseteq X$, the set $\{x\phi \mid x \in X'\}$ is the image of X' under ϕ and is denoted by $X'\phi$ or $\phi(X')$. The *inverse* ϕ^{-1} of ϕ is always defined as a relation $(\subseteq Y \times X)$, but usually it is not a mapping from Y to X. For any $Y' \subseteq Y$, the set $\{x \in X \mid x\phi \in Y'\}$ is the pre-image of Y' under ϕ, and it is denoted by $Y'\phi^{-1}$ or $\phi^{-1}(Y')$. The set of all mappings from X to Y is denoted by Y^X.

The *composition* of two mappings $\phi\colon X \to Y$ and $\phi'\colon Y \to Z$ is the mapping $\phi\phi'\colon X \to Z$, where $\phi\phi'$ is the composition of ϕ and ϕ' as relations. Then, $x\phi\phi' = (x\phi)\phi'$ for every $x \in X$.

For any $\phi\colon X \to Y$ and $Z \subseteq X$, the *restriction* of ϕ to Z is the mapping $\phi|_Z\colon Z \to Y$ such that $\phi|_Z(x) = \phi(x)$ for all $x \in Z$. If $\phi'\colon Z \to Y$ is the restriction of ϕ to Z, i.e., $Z \subseteq X$ and $\phi' = \phi|_X$, then we also say that ϕ is an *extension* of ϕ' to X.

A mapping $\phi\colon X \to Y$ is called

(i) *injective* if $\phi(x) = \phi(x')$ implies $x = x'$ for all $x, x' \in X$,

(ii) *surjective* if for every $y \in Y$, there is an $x \in X$ such that $y = \phi(x)$, and

(iii) *bijective* (or a *bijection*) if it is injective and surjective.

We denote the *identity mapping* of a set X that maps each element of X to itself by 1_X. Obviously, 1_X is a bijection. Any bijection from a set X to itself is called a *permutation* on X. We shall frequently use permutations on the sets $[n]$ $(n \in \mathbb{N}^+)$, and they are usually denoted by σ. Such a permutation σ is often denoted by $(i_1\, i_2\, \ldots\, i_n)$, where $\sigma(j) := i_j$ for every $j \in [n]$. If $n = 0$, the permutation of 0 elements is called the *empty permutation*.

Finally, we note that the term *effective* used for a given relation (e.g., equality) between two classes of objects means in fact that there exists an algorithm (Aho and Ullman, 1972, Section 0.4) which relates each object from one class to the other.

2.2 Strings and languages

An *alphabet* is any set of symbols, which are also called *letters*. As a rule, alphabets are assumed to be finite and usually denoted by X, Y and Z. The letters x, y, and z usually

denote symbols in X, Y, and Z. If $Y \subseteq X$, then Y is a *subalphabet* of X. A *string* over an alphabet X is any finite sequence $x_1x_2 \dots x_n$ ($n \in \mathbb{N}$) of letters x_i ($i \in [n]$) in X. There is one special string denoted by ε which has no letters ($n = 0$). It is called the *empty string*. We reserve v and w for denoting strings. The set of all strings over an alphabet X is denoted by X^*, and $X^+ := X^* \setminus \{\varepsilon\}$ is the set of non-empty strings over X.

If v, v', and w are arbitrary strings, then vw is the *concatenation* of v and w. We call v a *prefix* and w a *suffix* of the string vw. Moreover, w is a *substring* of vwv'. Both prefixes and suffixes of a string are substrings of it. For all strings v, $v\varepsilon = \varepsilon v = v$. Note that the empty string is a suffix, a prefix and a substring of every string. Moreover, $v^0 := \varepsilon$ and $v^i := v^{i-1}v$ for every $i \in \mathbb{N}_+$. Also, note (Rozenberg and Salomaa, 1997) that X^* and X^+ are, respectively, the *free monoid* and the *free semigroup generated by* X (with concatenation as operation). The *length* of a string v, i.e., its number of letters, is denoted by $|v|$, and $|v|_x$ denotes the number of occurrences of a given letter $x \in X$ in the string v. Furthermore, for any $Y \subseteq X$, we set $|v|_Y := \sum_{x \in Y} |v|_x$, i.e., $|v|_Y$ is the length of the string obtained by erasing from v all letters not in Y. The *reverse* of a string $v = v_1v_2 \dots v_n$ is the string $v^R = v_n \dots v_2v_1$ written in reverse order.

Subsets of X^* are called *languages* over X, and subsets of X^+ are *ε-free* languages. We reserve K and L to denote languages. Since any language is a set, the operations of union, intersection, difference and complementation are defined for languages in the standard set-theoretical way. If $K \subseteq X^*$ and $L \subseteq Y^*$, then the *concatenation* of K and L is the language $KL := \{vw \mid v \in K \text{ and } w \in L\}$. Sometimes we want to concatenate an arbitrary number of strings from a language, and hence we need to define the closure of a language. Let $L^0 := \{\varepsilon\}$ and $L^i := L^{i-1}L$ for every $i \in \mathbb{N}_+$. Then, $L^* := \bigcup_{i \geq 0} L^i$ is the *Kleene closure* of L, and $L^+ := \bigcup_{i \geq 1} L^i$ is the *positive closure* of L. We say that a family of languages is *closed under an operation* if the result of applying the operation to any language(s) of the family also belongs to the family. For example, a class of languages is closed under intersection if the intersection of any two languages in the class is also a member of the class.

Definition 2.2.1. A mapping $\mu\colon X^* \to Y^*$ is a (string) *homomorphism* if $\mu(vw) := \mu(v)\mu(w)$ for all $v, w \in X^*$. The mapping $\mu^{-1}\colon Y^* \to \wp(X^*)$, defined by $\mu^{-1}(w) := \{v \in X^* \mid \mu(v) = w\}$ for every $w \in Y^*$, is then called an *inverse homomorphism*. □

In particular, a homomorphism is defined by the image of the letters in its domain, i.e., $\mu(\varepsilon) = \varepsilon$, and if $v = x_1x_2 \dots x_n$ with $x_i \in X$ ($i \in [n]$), then $\mu(v) = \mu(x_1)\mu(x_2) \dots \mu(x_n)$.

If $Y \subseteq X$, we define the *projection* homomorphism $\mathrm{pr}_Y\colon X^* \to Y^*$ by setting $\mathrm{pr}_Y(y) := y$ for all $y \in Y$, and $\mathrm{pr}_Y(x) := \varepsilon$ for all $x \in X \setminus Y$. The symbols μ and ν shall always denote string homomorphisms.

By applying a homomorphism $\mu\colon X^* \to Y^*$ to a language $L \subseteq X^*$ we get another language $\mu(L) := \{\mu(v) \in Y^* \mid v \in L\}$. Also, for $L \subseteq Y^*$, $\mu^{-1}(L) := \{v \in X^* \mid \mu(v) \in L\}$ is the language consisting of those strings which get mapped by μ into a string in L.

Next, we illustrate some of the notions introduced in this section by an example.

Example 2.2.2. English (understood as the set of all possible sentences over the finite vocabulary of the English language), JAVA (understood as the set of all syntactically correct JAVA programs) and $\{0^n1^n \mid n \in \mathbb{N}\}$ are examples of languages. The string homomorphism $\mu\colon \{0,1,2\}^* \to \{x,y\}^*$ is defined by $\mu(0) := x$, $\mu(1) := yy$ and $\mu(2) := \varepsilon$. If $L = \{012\}^*$, then $\mu(L) = \{xyy\}^*$. Also $\nu\colon \{0,1\} \to \{x\}$, $\nu(0) := x$, $\nu(1) := x$ is a string homomorphism, and we have $\nu^{-1}(x) = \{0,1\}$ and $\nu^{-1}(x^*) = \{0,1\}^*$. □

2.3 Specifying languages

Usually, a language is infinite or, even if it is finite, it can contain arbitrarily many strings. Thus, it may be impossible, or at least not practical, to exhaustively enumerate all the strings in the language. Consequently, to study languages mathematically, finitary mechanisms to specify them are necessary.

2.3.1 Recognizers

A *recognizer* is a device that accepts a string as its input and decides, after some computations, whether the string belongs to a given language. If this is the case, it may simultaneously produce output of some form. Thus, the *language accepted* by a recognizer is the set of all strings it accepts.

Moreover, a recognizer is *deterministic* or *nondeterministic* depending on whether, in each configuration, there is at most one possible move or a finite set of possible moves. The general scheme of such devices can be found in Aho and Ullman (1972, Figure 2.1) or Linz (2001, Figure 2.1), for example.

2.3.2 Generating devices

The most common generating device is a formal grammar – a finite mechanism for producing sets of strings over an alphabet of *terminals*. To form such valid strings, it uses another alphabet of special symbols called *nonterminals*, which often represent syntactic categories. This alphabet is disjoint from the terminal alphabet. A distinguished symbol from the nonterminals is chosen as the start of the generation process. Moreover, a finite set of *rules* specifies how a string is transformed into another. The process of determining the way a terminal string is generated by a grammar is called *parsing*. Before proceeding with the formal definition of a generating device, we give an example from linguistics.

Example 2.3.1. A grammar for a natural language, let us say English, correctly decides, using a set of rules, whether a sentence is well formed or not. The terminal alphabet would contain all strings in the English language. Now, a common rule of English grammar is "A sentence (S) can consist of a noun phrase (NP) followed by a verb phrase (VP)". Treating S, NP and VP as nonterminals, we may write the grammatical rule mentioned above as $S \to NP\,VP$. Since we still did not obtain a sentence in English, we have to specify grammatical rules for both NP and VP. Usually, a noun phrase is formed of a determiner (DET) and a noun (NN), and a verb phrase may have a verb (VB) and a noun phrase. Hence, the grammar also has the rules $NP \to DET\,NN$ and $VP \to VB\,NP$. Now, if we choose S as the start symbol and associate the terminals *the* and *a* to DET, *boy* and *girl* to NN and *sees* to VB, we can progressively generate the grammatical English sentences *"the boy sees a girl"* and *"the girl sees the boy"*. □

Now, the precise definition of a grammar follows.

Definition 2.3.2. A *phrase structure grammar* is a system $G = (N, X, P, S)$ specified as follows.

(1) N is a finite set of *nonterminal symbols*.

(2) X is the *terminal alphabet* such that $X \cap N = \emptyset$.

(3) P is a finite subset of $(N \cup X)^* N (N \cup X)^* \times (N \cup X)^*$. An element (α, β) in P will be written $\alpha \to \beta$ and called a *production*. Moreover, α, and β will be referred to as the *left-hand side*, and *right-hand side* of the production, respectively.

(4) $S \in N$ is the distinguished *start symbol*.

If, for any $n \geq 2$, $\alpha \to \beta_1$, $\alpha \to \beta_2$, ..., $\alpha \to \beta_n$ are all in P, then in examples we may simply write $\alpha \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$. □

To generate strings, a grammar starts with the start symbol and then repeatedly uses rules to transform the current string until a string containing terminals only is obtained. The strings thus obtained at each step are called *sentential forms*, and they are defined inductively as follows.

(1) S is a sentential form.

(2) If $\delta\alpha\gamma$ is a sentential form and $\alpha \to \beta$ is in P, then also $\delta\beta\gamma$ is a sentential form.

Moreover, if $\delta\alpha\gamma$ and $\delta\beta\gamma$ are as above, then we write $\delta\alpha\gamma \Rightarrow_G \delta\beta\gamma$ and we say that $\delta\alpha\gamma$ *directly derives* $\delta\beta\gamma$. Furthermore, for any sentential forms δ and γ, $\delta \Rightarrow_G^* \gamma$ means that there is a *derivation*

$$\delta \Rightarrow_G \delta_1 \Rightarrow_G \ldots \Rightarrow_G \delta_{n-1} \Rightarrow_G \gamma$$

of length $n \geq 0$ of γ from δ in G. We say that γ is *derived* from δ (or equivalently, δ *derives* γ) in n derivation steps, and we express it by writing $\delta \Rightarrow_G^n \gamma$. Finally, the *language generated by* G is the set $L(G) := \{v \in X^* \mid S \Rightarrow_G^* v\}$. When the grammar G is clear from the context, we may drop the subscript G from $\Rightarrow_G$, $\Rightarrow_G^n$ and $\Rightarrow_G^*$.

An important concept is expressed when we say that two grammars are "equivalent". Since there are a number of general ideas of equivalence which make sense, the term *weak equivalence* is sometimes used for this concept.

Definition 2.3.3. Two grammars G_1 and G_2 are *(weakly) equivalent* if $L(G_1) = L(G_2)$. □

We use the following conventions to represent the elements involved in a grammar:

(i) x, y and z represent terminals, as do the digits 0, 1 and 2;

(ii) A, B and C are nonterminals, and S is always the start symbol;

(iii) v and w are strings of terminals only;

(iv) α, β, δ and γ represent strings that may contain both nonterminals and terminals (sentential forms).

To clarify the notions introduced so far, we give an example.

Example 2.3.4. The device $G = (\{S, A, B, C, C_1, C_2\}, \{x\}, P, S)$, where the productions

$$\begin{aligned} S &\to C_1CxC_2|x & Cx &\to xxC \\ CC_2 &\to AC_2|B & xA &\to Ax \\ xB &\to Bx & C_1A &\to C_1C \\ C_1B &\to \varepsilon \end{aligned}$$

form P, is a phrase structure grammar. The strings S and C_1xxCC_2 are examples of sentential forms of G, and a derivation in G is

$$S \Rightarrow_G C_1CxC_2 \Rightarrow_G C_1xxCC_2 \Rightarrow_G C_1xxB \Rightarrow_G C_1xBx \Rightarrow_G C_1Bxx \Rightarrow_G xx \ .$$

Using two endmarkers C_1 and C_2, the grammar allows only the construction of strings where the number of x's is a power of 2. Thus, $L(G) = \{x^{2^n} \mid n \in \mathbb{N}\}$. □

Phrase structure grammars can be classified according to the form of their productions. Thus, by imposing restrictions on the elements of P, we may obtain classes of languages with particular properties. Some of such classes are presented next.

Note that each time we introduce a class of languages, we present the corresponding class of generating device (the *descriptive method*) and the corresponding class of recognizers (the *pragmatic method*) specifying it. Sometimes we also give an *algebraic method* to describe languages, which is mostly used to investigate mathematical properties of such language classes.

2.3.3 The Chomsky hierarchy

We now survey some of the best-known types of phrase structure grammars and present recognizers for them.

Definition 2.3.5. A phrase structure grammar $G = (N, X, P, S)$ is said to be

- *context-sensitive* if all productions in P are of the form $\alpha A\beta \to \alpha\gamma\beta$, where $A \in N$ and $\alpha, \beta, \gamma \in (N \cup X)^*$, which means that A can be rewritten as γ only if it occurs in a context of α on the left and β on the right. In addition, P may contain the production $S \to \varepsilon$, and in this case S does not occur on the right-hand side of any production in P.

- *context-free* if each production in P is of the form $A \to \beta$, where $A \in N$ and $\beta \in (N \cup X)^*$.

- *right-linear* if each production in P is of the form $A \to xB$ or $A \to x$, where $A, B \in N$ and $x \in X \cup \{\varepsilon\}$. □

Sentential forms, derivations and generated languages are defined as in Section 2.3.2. Next we shall give some examples.

Example 2.3.6. The fragment grammar of Example 2.3.1 is context-free and generates a few sentences in English that are grammatically correct. On the other hand, the grammar $G' = (\{S, A, B, C\}, \{x, y\}, P, S)$ with P consisting of

$$\begin{array}{ll} S \to xA \mid yB \mid \varepsilon & B \to xC \mid yS \\ A \to yC \mid xS & C \to xB \mid yA \end{array}$$

is right-linear and generates the language

$$L(G') = \{v \in \{x, y\}^* \mid |v|_x \text{ and } |v|_y \text{ are both even}\} \ .$$

The phrase structure grammar $G = (\{S, A, B\}, \{x, y, z\}, P, S)$, where

$$\begin{aligned} S &\to xyz \mid xAyz & yB &\to By \\ Ay &\to yA & xB &\to xx \mid xxA \\ Az &\to Byzz \end{aligned}$$

are the productions in P, is context-sensitive. A derivation in G is

$$\begin{aligned} S \Rightarrow_G xAyz \Rightarrow_G xyAz \Rightarrow_G xyByzz \Rightarrow_G xByyzz \Rightarrow_G xxAyyzz \Rightarrow_G xxyAyzz \\ \Rightarrow_G xxyyAzz \Rightarrow_G xxyyByzzz \Rightarrow_G xxyByyzzz \Rightarrow_G xxByyyzzz \Rightarrow_G xxxyyyzzz \ . \end{aligned}$$

The nonterminals A and B act as messengers as follows. Each time an A is created, it travels to reach the first z, creates another y and z and lets the messenger B travel until it sees an x and then create another x. Thus, the language generated by G is $L(G) = \{x^n y^n z^n \mid n \geq 1\}$ (Linz, 2001, Example 11.2). □

Any of the above families of grammars gives rise to a family of languages.

Definition 2.3.7. A language L is said to be *type 0*, *context-sensitive*, *context-free*, or *regular*, if there exists a phrase structure, context-sensitive, context-free, or right-linear grammar G, respectively, such that $L = L(G)$. □

A great deal is known about these classes of languages and grammars. Part of the material is summarized in Table 2.3.1, where, for the undefined terms, the reader is invited to consult Harrison (1978), Hopcroft et al. (2006), Salomaa (1973), Linz (2001), and Rozenberg and Salomaa (1997), for example.

Grammars	**Languages**	**Notation**	**Recognizers**
Phrase structure (type 0)	Recursively enumerable	RE	Deterministic or nondeterministic Turing machines
Context-sensitive (type 1)	Context-sensitive	CSL	Linear bounded automata
Context-free (type 2)	Context-free	CFL	Pushdown automata
Right-linear (type 3)	Regular	REG	Deterministic or nondeterministic finite automata

Table 2.3.1: Main classes of languages and their specification methods.

In the original terminology (Chomsky, 1959b, a), RE, CSL, CFL, and REG are named type 0 languages, type 1 languages, type 2 languages, and type 3 languages, respectively. It is well known that each language class of type i ($i \in [3]$) is a proper subset of the family of type $i-1$. These relationships are depicted in the HASSE diagram of Figure 2.3.1 to which we will refer to as the *Chomsky hierarchy*. In such a diagram every upwards oriented edge denotes proper inclusion. No edge or chain of edges connecting two nodes denotes their incomparability.

Note that such HASSE diagrams not only establish hierarchies of language classes but also give a classification of recognizers and generating devices according to their power as

Figure 2.3.1: The Chomsky hierarchy.

language specification methods. For example, pushdown automata are more powerful than finite automata.

Unfortunately, for some practical purposes such as a better representation of all syntax of programming or natural languages, each of the four main classes is either too limited or too powerful. Thus, further families of languages together with their generating devices and recognizers were introduced and investigated. Consequently, to study their place in the Chomsky hierarchy becomes natural.

2.3.4 Decision problems

A *decision problem* is a question in some formal system with a `Yes`/`No` answer, depending on the values of some input parameters. A decision problem is *(algorithmically) decidable* if there exists an algorithm such that, given any instance of the problem as input, it outputs `Yes` or `No`, depending on whether the input is true or not, respectively. Otherwise, it is *undecidable*. The most common decidability issues related to languages are the following.

- *Emptiness*: is a given language L empty?
- *Finiteness*: is a given language L a finite set?
- *Membership*: does $v \in L$ hold for a given string v and a given language L?
- *Inclusion*: does $L_1 \subseteq L_2$ hold for two given languages L_1 and L_2?
- *Equality*: does $L_1 = L_2$ hold for two given languages L_1 and L_2?

2.4 Context-free languages

Context-free languages (CFLs) are languages generated by context-free grammars (CFGs) or recognized by pushdown machines – finite automata equipped with an auxiliary memory in the form of a pushdown stack. These specification methods have features that permit the description of nested structures needed in programming and natural languages. For example, a CFG defines the syntactic structure of almost all programming languages. Moreover, the Document Type Definition feature of XML is specified by a CFG which describes the permissible HTML tags and the ways in which these tags may be nested. For practical examples, the interested reader may consult Hopcroft et al. (2001, Section 5.3) or Wintner (2001, Section 3.6). Also, CFGs can be used to model RNA folding and to generate RNA

structures (Searls, 1992). Furthermore, deterministic versions of pushdown machines are widely used in parsing (see Aho et al., 2006, for an overview).

Now, we recall the formal definition of the class of CFLs as well as a few of their properties to be used later on. For details, we refer to Aho and Ullman (1972, Sections 2.4 and 2.6), Hopcroft et al. (2001, Chapters 5 and 7), and Linz (2001, Chapters 5, 6 and 8).

Definition 2.4.1. A *context-free grammar* (CFG) is a phrase structure grammar $CF = (N, X, P, S)$ in which each production in P is of the form $A \to \beta$, where $A \in N$ and $\beta \in (N \cup X)^*$. A language is *context-free* if it is defined by a CFG, and CFL denotes the class of all context-free languages. □

Next, an example is given.

Example 2.4.2. The context-free grammar

$$CF = (\{S\}, \{\texttt{if}, \texttt{else}\}, \{S \to SS \mid \texttt{if}\ S \mid \texttt{if}\ S\ \texttt{else} \mid \varepsilon\}, S)$$

generates the language of all possible sequences of `if` and `if-else` clauses in C. A sample derivation of `if if else` is

$$S \Rightarrow_{CF} SS \Rightarrow_{CF} \texttt{if}\ SS \Rightarrow_{CF} \texttt{if if}\ S\ \texttt{else}\ S \Rightarrow_{CF} \texttt{if if else}\ S \Rightarrow_{CF} \texttt{if if else}\ .\square$$

Note that a derivation may involve sentential forms with more than one nonterminal, and hence there is a choice of the order in which nonterminals are rewritten. For example, always the leftmost nonterminal was replaced at each step of the derivation above. But

$$S \Rightarrow_{CF} SS \Rightarrow_{CF} S\ \texttt{if}\ S\ \texttt{else} \Rightarrow_{CF} \texttt{if}\ S\ \texttt{if}\ S\ \texttt{else} \Rightarrow_{CF} \texttt{if}\ S\ \texttt{if else} \Rightarrow_{CF} \texttt{if if else}$$

is also a valid derivation generating `if if else` and using the same productions. The difference between them is entirely in the order in which productions are applied. A derivation is called *leftmost* if at each step the leftmost nonterminal of the current sentential form is rewritten. It is well known that if $\delta \Rightarrow^* \gamma$, then there is also a leftmost derivation of γ from δ. Analogously, a *rightmost* derivation may be defined.

A convenient way to show the derivation of a terminal string in a CFG is by a hierarchical structure called *parse tree*, *syntax tree* or *derivation tree* (cf. Aho and Ullman (1972, Section 2.4.1) or Linz (2001, pp. 130–133), for example). For example, the derivation tree corresponding to the above derivation of "`if if else`" is illustrated by Figure 2.4.1(b). We omit further details here because in Chapter 4, we systematically present the general notion of tree and formally define derivation trees to suit our needs.

The machines recognizing CFLs are called *pushdown automata*. Informally, such a recognizer is a nondeterministic finite automaton equipped with a pushdown stack as an auxiliary memory. This memory enables the machine to record a potentially unbounded amount of information in a last-in-first-out fashion. The deterministic versions of pushdown automata define an important proper subfamily of CFL: the *deterministic context-free languages*. For details, we refer to Hopcroft et al. (2001, Chapter 6), Linz (2001, Chapter 7), and Aho and Ullman (1972, Section 2.5).

Another proper subfamily of CFL is the class of linear languages defined thus.

Definition 2.4.3. A *linear grammar* is a context-free grammar $LG = (N, X, P, S)$ in which each production is of the form $A \to vBw$ or of the form $A \to v$, where $A, B \in N$ and

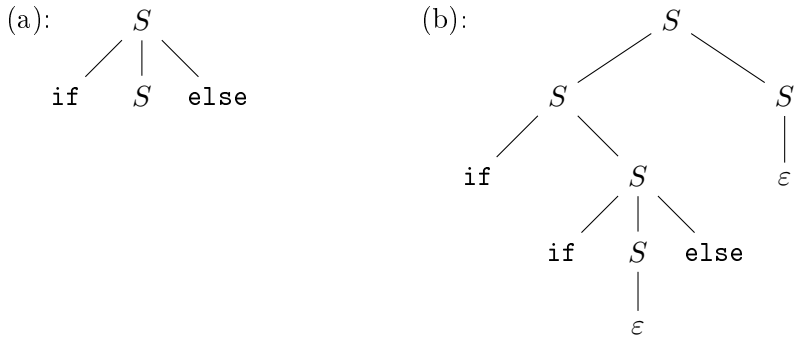


Figure 2.4.1: Derivation tree associated to the production $S \to \texttt{if}\ S\ \texttt{else}$ (a) and to the terminal string `if if else` (b).

$v, w \in X^*$. A language L is *linear* if $L = L(LG)$ for some linear grammar LG. The class of all linear languages is denoted by LIN. □

Also, there is a type of recognizer called *one-turn pushdown automaton* that accepts exactly the linear languages (see Harrison, 1978, Section 5.7).

Next, an example of a linear language is given.

Example 2.4.4. The grammar $LG = (\{S, A, B\}, \{x, y\}, P, S)$, where P has the productions

$$S \to A \mid B \mid \varepsilon,$$
$$A \to xAy \mid xy, \text{ and}$$
$$B \to xByy \mid xyy,$$

is linear and generates the language $L = \{x^n y^n \mid n \in \mathbb{N}\} \cup \{x^n y^{2n} \mid n \in \mathbb{N}\}$. □

It is often practical to modify a given CFG so that it has a particular structure. First of all, there might be certain types of undesirable productions which make parsing less efficient, or unusable symbols that do not affect the generative capacity. More precisely, we would like to construct an equivalent simplified grammar by removing, in the following order,

- ε-productions, i.e., productions of the form $A \to \varepsilon$ with $A \in N$,
- unit productions, i.e., productions of the form $A \to B$ with $A, B \in N$,
- nonterminals that do not generate any strings of terminal symbols, and
- nonterminals or terminals that cannot be reached in a derivation from S.

Definition 2.4.5. A CFG $CF = (N, X, P, S)$ is called

- *ε-free* if either
  - (1) P has no ε-productions, or
  - (2) there is exactly one ε-production $S \to \varepsilon$ and S does not appear on the right-hand side of any production in P;

- *cycle-free* if there is no derivation of the form $A \Rightarrow_{CF}^{+} A$ for any $A \in N$;
- *without useless symbols* if, for every $\delta \in N \cup X$, there is a derivation of the form $S \Rightarrow_{CF}^{*} v_1 \delta v_2 \Rightarrow_{CF}^{*} v_1 v v_2$ with $v, v_1, v_2 \in X^*$;
- *proper* if it is ε-free, cycle-free and without useless symbols. □

Then, we can note.

Theorem 2.4.6. *For any given CFG, there effectively exists an equivalent proper CFG.*

Now, we present a normal form for CFGs that have both practical and theoretical implications (e.g., faster parsing and easier proofs).

Definition 2.4.7. A CFG $CF = (N, X, P, S)$ is in *Chomsky normal form* (CNF) if each of its productions is of the form

(1) $A \to BC$ with $A, B, C \in N$,

(2) $A \to x$ with $A \in N$ and $x \in X$, or

(3) $S \to \varepsilon$.

Moreover, if $S \to \varepsilon$ is in P, then $B, C \in N \setminus \{S\}$ in any rule $A \to BC$ of type (1). □

There is an algorithm to convert a given CFG to one in CNF as certified by the next theorem.

Theorem 2.4.8. *For every CFG, there effectively exists an equivalent grammar in CNF.*

Next, we mention some of the closure properties of CFLs that will be used later.

Theorem 2.4.9. *The class of context-free languages is effectively closed under union, star-closure, concatenation, homomorphism, inverse homomorphism and intersection with regular languages, but it is not closed under intersection and complementation.*

Finally, we turn our attention to the common decision problems for CFLs.

Theorem 2.4.10. *The emptiness, finiteness and membership are decidable for context-free languages but the inclusion and equality are not.*

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

CHAPTER 3

Translations

A translation is a relation between elements of one language (the "source" or "input") and elements of another language (the "target" or "output"), that is to say, a set of pairs of strings. In NLP, a translation from a language, let us say Spanish, to another language such as Romanian, relates the sentences (i.e., strings) with the same meaning from these languages. In computer science, the notion of translation appeared in the same time as programming languages (FORTRAN-1957, ALGOL-1958, COBOL-1959) and their compilers. For example, a compiler can be seen as a device that translates a source program written by a user in a high-level programming language to a target program in a lower-level programming language, which can be recognized and executed by a computer. Therefore, it performs a string translation.

Two of the most investigated classes of translations with numerous applications are *regular translations* and *syntax-directed translations*. In the literature, regular translations are known under different names: regular transductions, finite state transducer mappings, finite transductions, rational translations, rational transductions or rational relations (see, for example, Elgot and Mezei, 1965, Ginsburg, 1966, Aho and Ullman, 1972, Eilenberg, 1974, Choffrut, 1978, Berstel, 1979, Yu, 1997, Mohri, 1996, 1997, Wintner, 2001, Roche and Schabes, 1995, 1997, Jurafsky and Martin, 2009, and the references therein). Detailed presentations of syntax-directed translations are given by Aho and Ullman (1972, 1973), Vere (1970), Aho et al. (2006), Huang (2008), and Crespi-Reghizzi (2009).

In this chapter, we start by fixing in Section 3.1 the general notation and terminology related to the translation theory. Next, in Section 3.2, we survey the usefulness in practice and the basic properties of regular translations (Theorem 3.2.4), as well as two specification methods for them: finite-state transducers (Definition 3.2.1) and string bimorphisms (Theorem 3.2.3). Then, we focus on specification methods for syntax-directed translations such as syntax-directed translation schemata (Section 3.3.1) and pushdown transducers (Section 3.3.2). Moreover, we overview in Section 3.3.3 the basic mathematical properties of this translation class, and we exhibit useful subclasses of syntax-directed translations in Section 3.3.1 and the inclusion relations between them in the HASSE diagram of Figure 3.3.1. We conclude this chapter by presenting several original references related to regular translations and (subclasses of) syntax-directed translations.

3.1 Basic notions

Let X and Y be alphabets. Any relation $\lambda \subseteq X^* \times Y^*$ is called a *translation*. The fact that $(v, w) \in \lambda$ for some $v \in X^*$ and $w \in Y^*$ means that λ *translates* v into w, and w is then called a *translation* of v. The *input alphabet* of λ is X and Y is the *output alphabet*.

Example 3.1.1. Every homomorphism $\mu\colon X^* \to Y^*$ specifies a translation

$$\lambda(\mu) := \{(v, v\mu) \mid v \in X^*\} \ (\subseteq X^* \times Y^*) \ .$$

For example, suppose we have a mathematical text written in English and we want to translate every Greek letter appearing in it into its corresponding English name. Let $X = \text{MATH} \cup \text{ROMAN} \cup \text{GREEK}$ be the input alphabet, where GREEK is the alphabet of Greek letters and MATH that of mathematical symbols. Moreover, let $Y = \text{MATH} \cup \text{ROMAN}$ be the output alphabet. To automatically change every Greek letter in the given text to its corresponding English name, we can use the homomorphism $\mu\colon X^* \to Y^*$ defined by setting $\mu(x) := x$ if $x \in X \setminus \text{GREEK}$, and $\mu(\alpha) :=$ alpha, $\mu(\beta) :=$ beta, ..., $\mu(\Omega) :=$ Omega for each letter in GREEK (see the table of Aho and Ullman, 1972, p. 214 for the complete definition). Then, the text $\alpha X + \beta Y$ would be translated into alpha$X$ + beta$Y$. □

Since translations are binary relations, all the notions and notation introduced in Section 2.1 apply to them, too. Thus, the *converse* of a translation $\lambda \subseteq X^* \times Y^*$ is the translation $\lambda^{-1} := \{(w, v) \mid (v, w) \in \lambda\}$ from Y^* to X^*, and for any $v \in X^*$, $K \subseteq X^*$, $w \in Y^*$ and $L \subseteq Y^*$,

- $v\lambda := \{w \in Y^* \mid (v, w) \in \lambda\}$ is the set of translations of v,
- $K\lambda := \bigcup_{v \in K} v\lambda$ is the set of translations of members of K,
- $w\lambda^{-1} := \{v \in X^* \mid (v, w) \in \lambda\}$ is the pre-image of w, and
- $L\lambda^{-1} := \bigcup_{w \in L} w\lambda^{-1}$ is the pre-image of L.

In particular, the *domain* of λ is the set

$$\text{Dom}(\lambda) := Y^*\lambda^{-1} = \{v \in X^* \mid \exists w \in Y^* \text{ such that } (v, w) \in \lambda\}$$

of all strings over X that have at least one translation, and the *range* of λ is the set

$$\text{Range}(\lambda) := X^*\lambda = \{w \in Y^* \mid \exists v \in X^* \text{ such that } (v, w) \in \lambda\}$$

of all strings over Y that are translations of at least one string in X^*.

Moreover, the *composition* of two translations $\lambda \subseteq X^* \times Y^*$ and $\lambda' \subseteq Y^* \times Z^*$ is the translation

$$\lambda \circ \lambda' := \{(v, w) \mid v \in X^*, w \in Z^*, (\exists v' \in Y^*)\ v\lambda v',\ v'\lambda' w\}\ .$$

The composition operation is extended in a natural way to classes of translations: if $\mathcal{A}$ and $\mathcal{B}$ are classes of translations, then

$$\mathcal{A} \circ \mathcal{B} = \{\lambda \circ \lambda' \mid \lambda \in \mathcal{A}, \lambda' \in \mathcal{B}\}$$

is the class of all translations that are the composition of a translation from $\mathcal{A}$ and a translation from $\mathcal{B}$. For any classes $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ of translations,

- $\mathcal{A} \circ \mathcal{B} \subseteq \mathcal{C}$ means that any composition of an $\mathcal{A}$-translation and a $\mathcal{B}$-translation is a $\mathcal{C}$-translation,
- $\mathcal{C} \subseteq \mathcal{A} \circ \mathcal{B}$ means that any $\mathcal{C}$-translation can be *decomposed* into the product of an $\mathcal{A}$-translation and a $\mathcal{B}$-translation, and
- $\mathcal{C} \circ \mathcal{C} \subseteq \mathcal{C}$ means that $\mathcal{C}$ is *closed under composition*.

Moreover, a class $\mathcal{C}$ of translations *preserves* a class $\mathcal{L}$ of languages if $L\lambda \in \mathcal{L}$ for all $\lambda \in \mathcal{C}$ and $L \in \mathcal{L}$.

Almost any interesting class of translations includes as a subclass the class $\mathcal{ID}_\lambda$ of *identity translations* $\{(v, v) \mid v \in X^*\}$. If $\mathcal{ID}_\lambda \subseteq \mathcal{C}$ for some class $\mathcal{C}$, then $\mathcal{C} \subseteq \mathcal{C} \circ \mathcal{C}$ and, even more, $\mathcal{B} \subseteq \mathcal{C} \circ \mathcal{B}$ and $\mathcal{B} \subseteq \mathcal{B} \circ \mathcal{C}$ for any class $\mathcal{B}$ of translations.

Similarly as in the case of languages (see Section 2.3), translations may be specified by generating devices, called now *synchronous grammars*, which simultaneously generate the pairs in a translation. This is done by associating the nonterminals (cf. Irons, 1961, Barnett and Futrelle, 1962, Čulík, 1965, Aho and Ullman, 1972, Wu, 1997, Saers, 2011, Satta and Peserico, 2005, Shieber, 2004, 2006, Shieber and Schabes, 1990b, a, Eisner, 2003, Abeillé et al., 1990, Schreiber, 1975, 1976, Maletti, 2013, Nederhof and Vogler, 2012, for example) or the nonterminals and terminals (Melamed, 2003, Melamed et al., 2004) on which the grammar is defined, or by linking the productions which guide the generation process (Georgeff, 1981). On the other hand, there are basically two types of machines recognizing translations:

(i) *two-tape recognizers* that get as input both an input string and a suggested translation of it (see Elgot and Mezei, 1965, Ginsburg, 1975, Berstel, 1979, for overviews);

(ii) *finite-state transducers* that produce a translation as an output from the given input string (see, for example, Ginsburg, 1966, Eilenberg, 1974, Aho and Ullman, 1972, Choffrut, 1978, Mohri, 1996, 1997, Yu, 1997, and the references therein).

In this monograph we shall consider and compare some means of defining translations. For any type of translation-defining devices TDD, we let $\lambda[TDD]$ denote the class of translations definable by a device belonging to TDD. Moreover, the symbol λ is reserved to always denote a translation.

3.2 Regular translations

One of the simplest, but yet very useful, type of translation is the *regular translation*. In this section, we have chosen the terminology and notation of Aho and Ullman (1972) and Yu (1997).

A translation is *regular* if it is computed by a finite-state transducer. Informally, a finite-state transducer is just a finite automaton with an associated write-only output tape. During each move, a string of symbols, possibly ε, is read and the machine emits a string of output symbols, possibly ε, on the output tape. In fact, one can assume without loss of generality that, during each move, the read input and the emitted output are just a single symbol or ε (Yu, 1997, Theorem 2.17). This is formalized as follows.

Definition 3.2.1. A *finite-state transducer* (FST) is a device $FT = (Q, X, Y, \kappa, q_0, F)$, where

(1) Q is the finite set of *states*,

(2) X is the *output alphabet* and Y is the *output alphabet*,

(3) $\kappa\colon Q \times (X \cup \{\varepsilon\}) \to \wp(Q \times (Y \cup \{\varepsilon\}))$ is the *transition mapping*,

(4) $q_0 \in Q$ is the *initial state*, and

(5) $F \subseteq Q$ is the set of *final states*. □

A *configuration* of FT is a triple (q, v, w), where $q \in Q$ is the current state, $v \in X^*$ is the input string remaining to be read and $w \in Y^*$ is the output string that has been already emitted. The moves of FT are determined by the transition mapping and described by the *next-configuration relation* $\vdash_{FT} \subseteq (Q \times X^* \times Y^*) \times (Q \times X^* \times Y^*)$, defined as follows. For all $q, q' \in Q$, $x \in X \cup \{\varepsilon\}$, $v \in X^*$, $y \in Y \cup \{\varepsilon\}$ and $w \in Y^*$, we write $(q, xv, w) \vdash_{FT} (q', v, wy)$ if $(q', y) \in \kappa(q, x)$. We say that $w \in Y^*$ is an *output*, or a *translation*, of $v \in X^*$ if $(q_0, v, \varepsilon) \vdash_{FT}^* (q, \varepsilon, w)$ for some $q \in F$. Thus, the *translation computed by* FT is the relation

$$\lambda(FT) := \{(v, w) \mid (q_0, v, \varepsilon) \vdash_{FT}^* (q, \varepsilon, w) \text{ for some } q \in F\} \ (\subseteq X^* \times Y^*) \ .$$

Note that an input string may have multiple translations. The class of translations computable by FSTs is called the class of *regular translations*, and it is denoted by $\lambda[FST]$.

The FST FT is *deterministic* if, for all $q \in Q$, either $\kappa(q, \varepsilon) = \emptyset$ and $|\kappa(q, x)| \leq 1$ for every $x \in X$, or $|\kappa(q, \varepsilon)| = 1$ and $\kappa(q, x) = \emptyset$ for every $x \in X$, but still several translations can be defined for a single input. To obtain exactly one translation for each input string, it is enough to require that no ε-moves can be made in a final state as suggested by Aho and Ullman (1972, p. 227), although other simple modifications of the definition of a deterministic FST can be found in the literature (Schützenberger, 1977, Ginsburg, 1966, Eilenberg, 1974, Berstel, 1979, Choffrut, 1978, Mohri, 1996, 1997). Contrary to the case of finite automata, not every FST can be determinized (Mohri, 1997, Choffrut, 1978).

Next, we give examples of regular translations (see also Wintner, 2001, 2002, Koskenniemi, 1983, Karp et al., 1992).

Example 3.2.2. The FST $FT = (\{q_0, q_1, \ldots, q_7\}, \text{ROMAN}, \text{ROMAN}, \kappa, q_0, \{q_3, q_7\})$ with κ defined by setting $\kappa(q_0, m) := (q_1, m)$, $\kappa(q_1, a) := (q_2, e)$, $\kappa(q_1, o) := (q_4, i)$, $\kappa(q_2, n) := (q_3, n)$, $\kappa(q_4, u) := (q_5, c)$, $\kappa(q_5, s) := (q_6, e)$ and $\kappa(q_6, e) := (q_7, \varepsilon)$ computes the regular translation that maps *man* and *mouse* to their plural *men* and *mice*, respectively. Moreover, it is deterministic. Such an FST can be easily extended to an FST that maps every English noun in singular to its plural form (Wintner, 2001, Example 2.19). Another simple and useful FST is a part-of-speech tagger that translates every string in some natural language into its corresponding tag (from the output alphabet of part-of-speech tags), for example, *girl* into *NN* and *see* into *VB*. Furthermore, this can be extended to a morphological analyzer that associates to every string in some natural language its internal structure of morphemes. For example, *girl* is translated into $girl - NN - singular$, and the output of *see* is $see - VB - present$. □

Nivat (1968, Section I.3) gives a more algebraic representation of regular translations by means of regular languages and string homomorphisms (cf. also Berstel, 1979, Theorems 3.2 and 4.1).

Theorem 3.2.3 (Nivat's Characterization)**.** *A translation $\lambda \subseteq X^* \times Y^*$ is regular if and only if there exist an alphabet Z, two homomorphisms $\mu\colon Z^* \to X^*$, $\nu\colon Z^* \to Y^*$ and a regular language $L \subseteq Z^*$ such that $\lambda = \mu^{-1} \circ \mathrm{id}_L \circ \nu$.*

A triple $SB = (\mu, L, \nu)$ with $L \subseteq Z^*$ (*center language*), $\mu\colon Z^* \to X^*$ (*input homomorphism*) and $\nu\colon Z^* \to Y^*$ (*output homomorphism*) is called a *string bimorphism*. The relation

$$\lambda(SB) := \mu^{-1} \circ \mathrm{id}_L \circ \nu = \{(v\mu, v\nu) \mid v \in L\} \ (\subseteq X^* \times Y^*)$$

is the *translation defined by SB*. In other words Nivat's theorem says that a translation is regular if and only if there is a string bimorphism with a regular center language defining it. This characterization offers elegant proofs of properties of regular translations such as the ones gather in Theorem 3.2.4 below (see Nivat (1968, Chapter III), Ginsburg (1966, Chapter 4) and Berstel (1979, Chapter III)). Further applications of this result may be found in the next section.

Theorem 3.2.4. *The following assertions are true.*

(i) *The domain and range of any regular translation are regular languages.*

(ii) *The class of regular translations is closed under union, star-closure, concatenation, inverse and composition, but it is not closed under intersection or complementation.*

(iii) *The class of regular translations preserves regular languages and context-free languages.*

(iv) *For any regular translation, the finiteness, the emptiness and the membership are decidable, but inclusion and equivalence are not.*

Not only that regular translations have good properties and a well-developed theory, but they are also efficient (see Jurafsky and Martin, 2009, Roche and Schabes, 1997, Mohri, 1997, 1996, for example) and easy to work with since they are implemented by toolkits like Carmel (Graehl, 1997), OpenFst (Allauzen et al., 2007) and XFST (Karttunen et al., 1997, Beesley and Karttunen, 2003). Thus, regular translations and FSTs (sometimes enriched with weights) are suitable to describe basic word transformations encountered in various real-world applications:

- in compiler design as models for lexical analyzers (Aho et al., 2006, Crespi-Reghizzi, 2009, Aho and Ullman, 1972, Section 3.3);
- in image processing as tools for image manipulation – filters, edge detectors, wavelet transform, etc. (Culik II and Kari, 1997);
- in computational biology as models for sequence analysis – genome alignment, finding frequent nucleotide patterns, reconstruction of DNA sequences, searching DNA databases, etc. (Cortes and Mohri, 2005);
- in natural language processing, especially in text processing – spelling correction, searching patterns in long texts, indexation, compression, part-of-speech tagging, morphological analysis, etc., and speech processing – large-vocabulary speech recognition, speech synthesis, etc. (Jurafsky and Martin, 2009, Beesley and Karttunen, 2003, Roche and Schabes, 1997, Mohri, 1996, 1997, Kaplan and Kay, 1994, Wintner, 2001, Koskenniemi, 1983, Karttunen and Wittenburg, 1983).

Unfortunately, they are limited, for example, in translations between natural languages and in the construction of other parts of a compiler, where a basic knowledge about the structure of the input and reordering of its parts are needed. Consequently, more powerful translation devices are called for, and some of these we study in the upcoming section.

3.3 Syntax-directed translations

Another well-known class of translations is that of syntax-directed translations. A syntax-directed translation originates in the idea to separate the syntactic description of a programming language from its semantics, and to use CFGs for the former (Irons, 1961, Barnett and Futrelle, 1962). Therefore, the core principle of a syntax-directed translation is that the meaning of an input is related to its syntactic structure (i.e., parse tree). Thus, syntax-directed translations have more representational power than regular translations (similarly as CFLs do over regular languages). They are suitable for applications:

- in programming-language compilation (see Aho and Ullman, 1972, 1973, Vere, 1970, Aho et al., 2006, Crespi-Reghizzi, 2009, and the references therein) as, for example, the design of compiler generators – programs that automate large parts of the work required to write a compiler;
- in natural language processing for semantic interpretation (De Mori et al., 1982) and, especially, for machine translation (see Yamada and Knight, 2001, Wu, 1997, Chiang, 2007, Huang, 2008, Chiang and Knight, 2006, for example).

Hence, the practical importance of syntax-directed translations motivated deeper and continuous theoretical investigations of specification methods, mathematical foundations (e.g., closure properties and decomposition results), and useful subclasses and the relations between them. Consequently, in what follows, we exhibit:

(i) specification methods of syntax-directed translations such as syntax-directed translation schemata (Section 3.3.1) and pushdown transducers (Section 3.3.2),

(ii) subclasses of syntax-directed translations (Section 3.3.1) and the inclusion relations between them (Section 3.3.4 and the Hasse diagram of Figure 3.3.1), and

(iii) basic mathematical properties of syntax-directed translations such as normal forms, generative capacity and composition and decomposition results (Section 3.3.3).

In Section 3.3.3 we also present an algebraic characterization, the string bimorphism, which naturally yields various subclasses of syntax-directed translations and offers elegant proofs of their properties. Moreover, in Section 6.2, we introduce tree bimorphisms that specify all syntax-directed translations. Such a new characterization will be further used, for example, to improve the mathematical foundations of syntax-directed translations by proving other closure properties (union, closure under composition, etc.).

3.3.1 Syntax-directed translation generating devices

First, we present a generating device defining a syntax-directed translation – a synchronous grammar called syntax-directed translation schema (Irons, 1961, Aho and Ullman, 1972, Čulík, 1965) and also known as syntax-directed transduction (Lewis II and Stearns, 1968). It consists of an input CFG and an output CFG with a common set of nonterminals. In every synchronous production the number of nonterminals occurring in the right-hand side of the production of the input grammar is the same as the number of nonterminals occurring in the corresponding right-hand side of the production in the output grammar. Moreover, a pairing is made by associating occurrences of the same input/output nonterminals. In

other words, a syntax-directed translation schema may be seen as a CFG with translation elements attached to each production. Whenever a production is used in a derivation of an input string, the associated translation element generates a part of an output string. The precise definition follows.

Definition 3.3.1. A *syntax-directed translation schema* (SDTS) is a device $SD = (N, X, Y, P, S)$ specified as follows.

(1) N is the alphabet of *nonterminal symbols* such that $N \cap (X \cup Y) = \emptyset$.

(2) X and Y are the *input* and the *output alphabet*.

(3) $S \in N$ is the *start symbol*.

(4) P is a finite set of *productions* of the form

$$A; A \to v_0 A_1 v_1 \dots v_{m-1} A_m v_m; w_0 A_{\sigma(1)} w_1 \dots w_{m-1} A_{\sigma(m)} w_m \ (\sigma) \tag{3.3.1}$$

where $m \geq 0$, $A, A_1, \dots, A_m \in N$, σ is a permutation on $[m]$, and for every $0 \leq i \leq m$, $v_i \in X^*$ and $w_i \in Y^*$. Note that $\sigma(i) = j$ means that the i^{th} nonterminal in β corresponds to the j^{th} nonterminal in α. □

When $\alpha \in X^*$ and $\beta \in Y^*$ in a production $A; A \to \alpha; \beta\,(\sigma)$ in P, we may omit the empty permutation σ and write the production simply as $A; A \to \alpha; \beta$. The *input grammar* of SD is the CFG $SD^{\text{in}} = (N, X, P^{\text{in}}, S)$, where $P^{\text{in}} := \{A \to \alpha \mid A; A \to \alpha; \beta\,(\sigma) \in P \text{ for some } \beta \text{ and } \sigma\}$. Similarly, the CFG $SD^{\text{out}} = (N, Y, P^{\text{out}}, S)$, where $P^{\text{out}} := \{A \to \beta \mid A; A \to \alpha; \beta\,(\sigma) \in P \text{ for some } \alpha \text{ and } \sigma\}$, is called the *output grammar* of SD.

To present the semantics of an SDTS SD, we use the notion of associated nonterminals. Whenever we apply a production in a derivation, we have to apply it to two "associated" nonterminals. This notion will be formalized later in Section 5.3.1 for a more general case (cf. Aho and Ullman, 1969a, p. 321). The *translation forms* of SD, which are elements of $(N \cup X)^* \times (N \cup Y)^*$, are defined inductively as follows.

(1) (S, S) is a translation form, and the two Ss are said to be *associated*.

(2) If $(\gamma A \delta, \gamma' A \delta')$ is a translation form in which the two explicit instances of A are associated and $A; A \to \alpha; \beta\,(\sigma)$ is a production in P, then $(\gamma\alpha\delta, \gamma'\beta\delta')$ is a translation form. The nonterminals of α and β are associated in $(\gamma\alpha\delta, \gamma'\beta\delta')$ the same way as they are associated in the production. The nonterminals of γ and δ are associated with those of γ' and δ' in the new translation form exactly as in the original one.

If $(\gamma A \delta, \gamma' A \delta')$ and $(\gamma\alpha\delta, \gamma'\beta\delta')$ are as above, then we write

$$(\gamma A \delta, \gamma' A \delta') \Rightarrow_{SD} (\gamma\alpha\delta, \gamma'\beta\delta').$$

This is a *leftmost derivation step* if the explicit instance of A is the leftmost occurrence of any nonterminal symbol in $\gamma A \delta$. Furthermore, for any translation forms (γ, δ) and (γ', δ'), $(\gamma, \delta) \Rightarrow^*_{SD} (\gamma', \delta')$ means that, for some $n \in \mathbb{N}$, there exists a *derivation*

$$(\gamma, \delta) \Rightarrow_{SD} (\gamma_1, \delta_1) \Rightarrow_{SD} \dots \Rightarrow_{SD} (\gamma_{n-1}, \delta_{n-1}) \Rightarrow_{SD} (\gamma', \delta')$$

of (γ', δ') from (γ, δ) in SD. A derivation is *leftmost* if every step in it is leftmost. The *translation defined* by SD is the relation

$$\lambda(SD) := \{(v, w) \mid (S, S) \Rightarrow_{SD}^{*} (v, w)\} \ (\subseteq X^* \times Y^*) \ .$$

Let $\lambda[SDTS]$ denote the class of translations definable by SDTSs. A translation is *syntax-directed* (SDT) if it is definable by an SDTS. Moreover, two SDTSs are *equivalent* if they define the same translation. In what follows, we might denote the class of all syntax-directed translations by $\mathcal{SDT}$.

We illustrate the notions introduced so far by giving an example of an SDTS that translates a fragment of English into Japanese (Yamada and Knight, 2001, Figure 1). Tîrnăucă (2008, Section 5) gives as an example SDTSs that model a fragment of a Romanian-English and an English-Spanish translation.

Example 3.3.2. The system $SD = (N, \text{ROMAN}, \text{ROMAN}, P, S, S)$, where the nonterminals S, VB, VB_1, VP, NN, TO, DET and PRP are in N and P consists of the productions

$$\begin{aligned}
S; S &\to PRP\, VB_1\, VP; PRP\, VP\, VB_1\ (1\,3\,2) & VB_1; VB_1 &\to \textit{adores}; \textit{daisuki desu}\\
VP; VP &\to VB\, TO; TO\, VB\, \textit{ga}\ (2\,1) & VB; VB &\to \textit{listening}; \textit{kiku no}\\
TO; TO &\to TO\, NN; NN\, TO\ (2\,1) & TO; TO &\to \textit{to}; \textit{wo}\\
PRP; PRP &\to \textit{he}; \textit{kare ha} & NN; NN &\to \textit{music}; \textit{ongaku}\ ,
\end{aligned}$$

is an SDTS. A derivation in SD is

$$\begin{aligned}
(S, S) &\Rightarrow_{SD} (PRP\, VB_1\, VP, PRP\, VP\, VB_1) \Rightarrow_{SD} (\textit{he}\, VB_1\, VP, \textit{kare ha}\, VP\, VB_1)\\
&\Rightarrow_{SD} (\textit{he adores}\, VP, \textit{kare ha}\, VP\ \textit{daisuki desu})\\
&\Rightarrow_{SD} (\textit{he adores}\, VB\ TO, \textit{kare ha}\, TO\, VB\ \textit{ga daisuki desu})\\
&\Rightarrow_{SD} (\textit{he adores listening}\, TO, \textit{kare ha}\, TO\ \textit{kiku no ga daisuki desu})\\
&\Rightarrow_{SD} (\textit{he adores listening}\, TO\, NN, \textit{kare ha}\, NN\, TO\ \textit{kiku no ga daisuki desu})\\
&\Rightarrow_{SD}^{2} (\textit{he adores listening to music}, \textit{kare ha ongaku wo kiku no ga daisuki desu})
\end{aligned}$$

translating the English text *"he adores listening to music"* into its corresponding Japanese sentence *"kare ha ongaku wo kiku no ga daisuki desu"*. In this example, we may observe some of the features of SDTSs that may be appealing, up to some extent, not only for the design of compilers but also for syntax-based translation of natural languages: they model simple reordering of parts of sentences required by languages with different grammatical structure (e.g., swapping of grammatical categories), they do insertions of extra strings in a derivation step to specify different syntactic cases, and they perform rough word-for-word translations between strings of both languages acting like a dictionary. □

By restricting the productions of SDTSs or by giving them more freedom (e.g., in the associating process), we may obtain other useful classes of translation defining devices and translations (Aho and Ullman, 1972, 1973, Aho et al., 2006, Wu, 1997, Saers, 2011, Satta and Peserico, 2005, Satta, 2007). These we describe next. Firstly, we focus on the restricted versions of SDTSs.

Definition 3.3.3. An SDTS $SD = (N, X, Y, P, S)$ is said to be

- *of order* k (k-SDTS, $k \geq 0$) if in no production $A; A \to \alpha; \beta\,(\sigma)$ in P, the number of nonterminals in α exceeds k,
- *simple* (sSDTS) if in each production $A; A \to \alpha; \beta\,(\sigma)$ in P, σ is the identity permutation,
- an *inversion transduction grammar* (ITG) if in each production $A; A \to \alpha; \beta\,(\sigma)$ in P, σ is the identity permutation or the inverse permutation,
- *right-linear* (rSDTS) if each production in P is of the form $A; A \to vB; wB\,(1)$ or of the form $A; A \to v; w$, where $A, B \in N$, $v \in X^*$ and $w \in Y^*$, or
- *linear* (lSDTS) if each production in P is of the form $A; A \to vBv'; wBw'\,(1)$ or of the form $A; A \to v; w$, where $A, B \in N$, $v, v' \in X^*$ and $w, w' \in Y^*$.

The classes of translations defined by SDTSs of order k, simple SDTSs, ITGs, right-linear SDTSs, and linear SDTSs are denoted by $\lambda[k\text{-}SDTS]$, $\lambda[sSDTS]$, $\lambda[ITG]$, $\lambda[rSDTS]$, and $\lambda[lSDTS]$, respectively. □

To clarify the above definition, we give some examples.

Example 3.3.4. The SDTS $SD_1 = (\{A, S\}, \{0, 1, \sharp\}, \{0, 1, \sharp\}, P, S)$, where P consists of

$$S; S \to A\sharp A; A\sharp A\,(2\,1) \qquad A; A \to 1A; 1A\,(1)$$
$$A; A \to 0A; 0A\,(1) \qquad A; A \to \varepsilon; \varepsilon\ ,$$

is an ITG of order 2. Obviously, $\lambda(SD_1) = \{(v\sharp w, w\sharp v) \mid v, w \in \{0,1\}^*\}$. The SDTS

$$SD_2 = (\{S\}, \{0,1\}, \{0,1\}, \{S; S \to \varepsilon; \varepsilon, S; S \to 0S; S0\,(1), S; S \to 1S; S1\,(1)\}, S)$$

is linear (and hence of order 1) and defines the translation $\lambda(SD_2) = \{(v, v^R) \mid v \in \{0,1\}^*\}$. The SDTS of Example 3.3.2 has order 3, and is neither simple nor an ITG. The SDTS

$$SD_3 = (\{S\}, X, X, \{S; S \to +SS; SS+\ (1\,2), S; S \to *SS; SS*\ (1\,2), S; S \to x; x\}, S)$$

where $X = \{+, *, x\}$, is simple (of order 2) and translates every prefix Polish arithmetic expression over X into its corresponding postfix Polish expression. The system

$$SD_4 = (\{S\}, \{0,1\}, \{0,1\}, \{S; S \to 0S; 1S\,(1), S; S \to 1S; 0S\,(1), S; S \to \varepsilon; \varepsilon\}, S)$$

is a right-linear SDTS that translates every bit string into its bitwise complement, e.g., 1001 into 0110. □

Next we study synchronous context-free grammars, which are generalized SDTSs in which nonterminals associated in a production can be distinct. This decoupling of nonterminals may be essential to capture the syntactic divergences between languages (Huang et al., 2009, p. 565) and it allows more general parse tree transformations (Satta, 2007) as we will formally show in Section 6.2.2. Also, it has been claimed that this stronger expressivity may be very convenient when proving formal properties of the model (Satta, 2007). Let us note that in the NLP community, the term synchronous context-free grammar often refers to the syntax-directed translation schemata considered in Definition 3.3.1 (see

Chiang, 2006, 2007, Zhang et al., 2006, Zhang and Gildea, 2007, for example). However, we prefer the traditional names and to consider the two formalisms separately. The formal definition of a synchronous context-free grammar is as follows (cf. Satta and Peserico, 2005).

Definition 3.3.5. A *synchronous context-free grammar* (SCFG) is a construct $SC = (N, X, Y, P, S, S')$, where

(1) N, X and Y are as in Definition 3.3.1,

(2) $S, S' \in N$ are the *start symbols*, and

(3) P is a finite set of *productions* of the form

$$A; B \to v_0 A_1 v_1 \dots v_{m-1} A_m v_m; w_0 B_1 w_1 \dots w_{m-1} B_m w_m \, (\sigma) \tag{3.3.2}$$

where $m \geq 0$, $A, A_1, \dots, A_m, B, B_1, \dots, B_m \in N$, σ is a permutation on $[m]$, and for every $0 \leq i \leq m$, $v_i \in X^*$ and $w_i \in Y^*$. Note that $\sigma(i) = j$ has the same meaning as for SDTSs. □

The input grammar, output grammar, translation forms, derivation and leftmost derivation are defined for SCFGs in an analogous way as for SDTSs (cf. Satta, 2007, Tîrnăucă, 2011, for details). Thus, the *translation defined* by SC is the relation

$$\lambda(SC) := \{(v, w) \mid (S, S') \Rightarrow^*_{SC} (v, w)\} \; (\subseteq X^* \times Y^*) \; .$$

The class of translations definable by SCFGs is denoted by $\lambda[SCFG]$.

An example of an SCFG is given next.

Example 3.3.6. The device

$$SC = (\{S, A\}, \{x\}, \{0, 1\}, \{S; S \to S; A \, (1), S; S \to x; 0, S; A \to x; 1\}, S, S)$$

is an SCFG. There are exactly two successful derivations in SC:

$$(S, S) \Rightarrow_{SC} (x, 0) \qquad \text{and} \qquad (S, S) \Rightarrow_{SC} (S, A) \Rightarrow_{SC} (x, 1) \; ,$$

and therefore, $\lambda(SC) = \{(x, 0), (x, 1)\}$.

Note that even if it is obvious how to construct an SDTS defining the same translation, there is no SDTS that can capture exactly the syntactic divergences represented by the two pairs of syntactic trees of SC. This is suggested by the fact that the same nonterminal may appear in a different number of occurrences in the pairs of parse trees, which is not the case for an SDTS. More precisely, the following SCFG-rule (Huang et al., 2009)

$$VP \; ; VP \to VB \; NN \; ; VBZ \; NNS \, (1 \, 2)$$

illustrates that Chinese does not have a plural noun (*VBZ*) or third-person-singular verb (*NNS*). The difference between the syntactic trees of SDTSs and SCFGs will be formally shown with the help of tree bimorphisms in Section 6.2.2. □

3.3.2 Recognizers of syntax-directed translations

In this section we turn our attention to machines specifying various forms of syntax-directed translations: *nondeterministic pushdown transducers*, which are in fact pushdown automata

with an attached write-only output tape, and *deterministic pushdown transducers*. Aho and Ullman (1972) shall be checked for details.

Lewis II and Stearns (1968, Theorem 3) showed that simple SDTSs and pushdown transducers define the same translation class. Aho and Ullman (1969b) proved that deterministic pushdown transducers are less powerful translation devices that nondeterministic pushdown transducers. Therefore, we get the following result, where $\lambda[PDT]$ and $\lambda[DPDT]$ denote the class of all translations definable by nondeterministic pushdown transducers and deterministic pushdown transducers, respectively,.

Theorem 3.3.7. $\lambda[DPDT] \subset \lambda[PDT] = \lambda[sSDTS]$.

To define all SDTs, one may add to a pushdown transducer a fixed number $k \geq 0$ of storage registers associated to each stack symbol (Aho and Ullman, 1969b). A storage register holds a string of output symbols. When the machine pops the topmost symbol of the stack, the contents of its registers are transmitted to the below stack symbol. Such a device is called *k-register pushdown transducer*, and let $\lambda[kPDT]$ denote the class of translations defined by all k-register pushdown transducers. Aho and Ullman (1969b, Theorem 4.1 and 4.2) proved the following.

Theorem 3.3.8. $\lambda[kPDT] = \lambda[k\text{-}SDTS]$, *and the equality is effective.*

3.3.3 Properties of syntax-directed translations

In this section we present some of the general properties of syntax-directed translations (cf. Aho and Ullman, 1972, Vere, 1970). We start by noting the generative capacity of an SDTS (Aho and Ullman, 1972, Exercise 3.1.7).

Proposition 3.3.9. *The domain and range of any SDTS of order $k \geq 2$ are CFLs. If the SDTS is of order 1, then its domain and range are linear languages.*

Proof. Let $SD = (N, X, Y, P, S)$ be any SDTS. Firstly, we show that $\mathrm{Dom}(\lambda(SD)) = L(SD^{\mathrm{in}})$. If $v \in \mathrm{Dom}(\lambda(SD))$, then there is $w \in Y^*$ such that $(v, w) \in \lambda(SD)$, which means that $(S, S) \Rightarrow^*_{SD} (v, w)$, and hence $S \Rightarrow^*_{SD^{\mathrm{in}}} w$. So, $\mathrm{Dom}(\lambda(SD)) \subseteq L(SD^{\mathrm{in}})$.

Conversely, if $v \in L(SD^{\mathrm{in}})$, then $S \Rightarrow^*_{SD^{\mathrm{in}}} v$, and hence there is a derivation

$$S \Rightarrow_{SD^{\mathrm{in}}} \delta_1 \Rightarrow_{SD^{\mathrm{in}}} \ldots \Rightarrow_{SD^{\mathrm{in}}} \delta_{n-1} \Rightarrow_{SD^{\mathrm{in}}} v$$

of v from S in SD^{in}. By the definition of SD^{in}, each production $A \to \alpha$ in P^{in} is constructed from at least one production $A; A \to \alpha; \beta\,(\sigma)$ in P for some β and σ. Hence, there is at least one corresponding derivation

$$(S, S) \Rightarrow_{SD} (\delta_1, \gamma_1) \Rightarrow_{SD} \ldots \Rightarrow_{SD} (\delta_{n-1}, \gamma_{n-1}) \Rightarrow_{SD} (v, w)$$

in SD for some $\gamma_i \in (N \cup Y)^* (i \in [n-1])$ and $w \in Y^*$ (since v does not contain any nonterminals because in any translation form of an SDTS, the input and output parts contain the same nonterminals and the same number of each). So, there exists $w \in Y^*$ (not necessarily unique) such that $(S, S) \Rightarrow^*_{SD} (v, w)$, and consequently $v \in \mathrm{Dom}(\lambda(SD))$. Hence, $L(SD^{\mathrm{in}}) \subseteq \mathrm{Dom}(\lambda(SD))$.

Since SD^{in} is a CFG, this shows that $\mathrm{Dom}(\lambda(SD))$ is a CFL, and if SD is of order 1, the grammar SD^{in} is linear.

For ranges the proof can be done analogously. □

On the other hand, the following property of SDTSs shows a generative limitation of these devices (Aho and Ullman, 1969b, Theorem 4).

Theorem 3.3.10. *If $\lambda \in \lambda[SDTS]$, then there is $k \in \mathbb{N}$ such that, for all $v \in \mathrm{Dom}(\lambda)$ with $v \neq \varepsilon$, there exists w such that $(v, w) \in \lambda$ and $|v| \leq k|w|$.*

The above result can be used, for example, to show that certain translations (e.g., $\{(x^n, x^{n^2}) \mid n \geq 1\}$) cannot be defined by any SDTS.

As in the case of CFGs (see Theorem 2.4.6), there may be redundant productions and nonterminals in an SDTS that can be removed in an effective way (Aho and Ullman, 1972, Lemmata 3.6 and 3.9).

Theorem 3.3.11. *If $\lambda \in \lambda[k\text{-}SDTS]$ ($k \geq 2$), then there effectively exists a k-SDTS $SD = (N, X, Y, P, S)$ such that*

(1) *there is no production of the form $A; A \to B; B$ in P (i.e., no unit productions),*

(2) *there is no production of the form $A; A \to \varepsilon; \varepsilon$ in P with $A \neq S$, and if $S; S \to \varepsilon; \varepsilon$ is in P, then S does not occur in the right-hand side of any production,*

(3) *there are no useless nonterminals (i.e., for any $A \in N$, there is a derivation $(S, S) \Rightarrow_{SD}^* (\delta_1 A \delta_2, \gamma_1 A \gamma_2) \Rightarrow_{SD}^* (v, w)$ for some $\delta_1, \delta_2 \in (N \cup X)^*$, $\gamma_1, \gamma_2 \in (N \cup Y)^*$, $v \in X^*$ and $w \in Y^*$), and*

(4) $\lambda(SD) = \lambda$.

Any SDTS as above is called *proper*. In addition, it is useful, for both practical and theoretical considerations, to also put its productions in a normal form. For different types of SDTSs the following normal forms are available.

Theorem 3.3.12. *The following hold.*

(i) *Any right-linear SDTS is effectively equivalent to a proper right-linear SDTS with every production of the form $A; A \to xB; yB\,(1)$, or of the form $A; A \to x; y$, where $A, B \in N$, $x \in X \cup \{\varepsilon\}$ and $y \in Y \cup \{\varepsilon\}$.*

(ii) *Any ITG is effectively equivalent to an ITG in which each production is of one of the forms*

(1) *$A; A \to BC; BC\,(1\,2)$ with $A, B, C \in N$,*

(2) *$A; A \to BC; CB\,(2\,1)$ with $A, B, C \in N$,*

(3) *$A; A \to x; y$ with $x \in X \cup \{\varepsilon\}$ and $y \in Y \cup \{\varepsilon\}$, but x and y are not both ε, or*

(4) *$S; S \to \varepsilon; \varepsilon$, and then $B, C \in N \setminus \{S\}$ in any production of type (1) or (2).*

(iii) *Any simple SDTS is effectively equivalent to a simple SDTS where each production is of the form (1), (3), or (4) as in (ii).*

(iv) *Any SDTS of order $k \geq 2$ is effectively equivalent to a proper SDTS with each of its productions of the form*

(1) *$A; A \to A_1 \dots A_m; A_{\sigma(1)} \dots A_{\sigma(m)}\,(\sigma)$ with $A_1, \dots, A_m \in N$, or of the form*

(2) $A; A \to v; w$ *with* $v \in X^*$ *and* $w \in Y^*$.

We say that such an SDTS is in weak normal form.

(v) *Any SDTS is effectively equivalent to a proper SDTS with each of its productions of the form*

(1) $A; A \to A_1 \dots A_m; A_{\sigma(1)} \dots A_{\sigma(m)}\,(\sigma)$ *with* $A_1, \dots, A_m \in N$, *or of the form*

(2) $A; A \to x; y$ *with* $x \in X \cup \{\varepsilon\}$ *and* $y \in Y \cup \{\varepsilon\}$.

We say that the SDTS is in strong normal form.

(vi) *Any linear SDTS is effectively equivalent to a linear SDTS with every production of the form* $A; A \to \varepsilon; \varepsilon$ *or of the form* $A; A \to xBx'; yBy'$ *with* $A, B \in N$, $x, x' \in X \cup \{\varepsilon\}$, $y, y' \in Y \cup \{\varepsilon\}$, *and* x, x', y *and* y' *are not all* ε.

Proof. For (i), let $SD = (N, X, Y, P, S)$ be any right-linear SDTS. From this we construct a right-linear SDTS $SD' = (N', X, Y, P', S)$ as follows. Let $A; A \to vB; wB$ be any production in P with $v = x_1 \dots x_n$ $(x_i \in X)$ and $w = y_1 \dots y_m$ $(y_i \in Y)$. If $1 \leq n < m$ (the case $n \geq m \geq 1$ is treated analogously), we add the productions

$$\begin{aligned} A; A &\to x_1 B_1; y_1 B_1\,(1) \\ B_i; B_i &\to x_{i+1} B_{i+1}; y_{i+1} B_{i+1}\,(1) \quad (i \in [n-1]) \\ B_j; B_j &\to B_{j+1}; y_{j+1} B_{j+1}\,(1) \quad (n \leq j \leq m-2) \\ B_{m-1}; B_{m-1} &\to B; y_m B\,(1) \end{aligned}$$

to P'. Moreover, we add A, B and B_l $(l \in [m-1])$ to N'. Furthermore, for every production $A; A \to v; w$ in P with $v = x_1 \dots x_n$ $(x_i \in X)$, $w = y_1 \dots y_m$ $(y_i \in Y)$ and $1 \leq n < m$ (the case $n \geq m \geq 1$ is treated analogously), the productions

$$\begin{aligned} A; A &\to x_1 B_1; y_1 B_1\,(1) \\ B_i; B_i &\to x_{i+1} B_{i+1}; y_{i+1} B_{i+1}\,(1) \quad (i \in [n-1]) \\ B_j; B_j &\to B_{j+1}; y_{j+1} B_{j+1}\,(1) \quad (n \leq j \leq m-2) \\ B_{m-1}; B_{m-1} &\to \varepsilon; y_m \end{aligned}$$

are added to P', and $A, B_1, \dots, B_{m-1}$ to N'. If $n \in \{0, 1\}$ and $m \in \{0, 1\}$, the production is already in the desired normal form. Obviously every production in P has the same effect on the generative process as the correspondent productions newly introduced in P'. Hence, $\lambda(SD) = \lambda(SD')$, and SD' is in the required normal form.

Statement (ii) was shown by Wu (1997, Theorem 2). Using the string bimorphism characterization of simple SDTSs (Theorem 3.3.15), (iii) was proved by Aho and Ullman (1969a, Corrolary 1) or by Aho and Ullman (1972, Theorem 3.6). The proof of (iv) is due to Aho and Ullman (1969b, Lemmata 3.1 and 3.2) or to Aho and Ullman (1972, Lemmata 3.7 and 3.9).

To prove (v), for any given k-SDTS $SD = (N, X, Y, P, S)$ $(k \geq 2)$, we construct the SDTS $SD' = (N', X, Y, P', S)$ (cf. Maletti and Tîrnăucă, 2009), where

- $N' := N \cup \{\underline{x} \mid x \in X\} \cup \{\overline{y} \mid y \in Y\}$ with $\underline{x}$ and $\overline{y}$ being new nonterminals,

- for every $x \in X$ and $y \in Y$, the following two productions are in P'

$$\underline{x} \to x; \varepsilon \qquad \text{and} \qquad \overline{y} \to \varepsilon; y \ ,$$

- and for every typical SDTS-production

$$A; A \to v_0 A_1 v_1 \dots v_{m-1} A_m v_m; w_0 A_{\sigma(1)} w_1 \dots w_{m-1} A_{\sigma(m)} w_m \ (\sigma)$$

of the form (3.3.1) in P with $v_i = x_{i1} \dots x_{ik_i}$ ($k_i \geq 0$, $x_{ij} \in X$) and $w_i = y_{i1} \dots y_{il_i}$ ($l_i \geq 0$, $y_{ij} \in Y$), the following production

$$A; A \to \underline{v_0}\overline{w_0} A_1 \underline{v_1}\overline{w_1} \dots A_m \underline{v_m}\overline{w_m}; \underline{v_0}\overline{w_0} A_{\sigma(1)} \underline{v_1}\overline{w_1} \dots A_{\sigma(m)} \underline{v_m}\overline{w_m} \ (\sigma)$$

is in P', where for every $0 \leq i \leq m$ we define

$$\underline{x_{i1} \dots x_{ik_i}} := \underline{x_{i1}} \dots \underline{x_{ik_i}} \qquad \text{and} \qquad \overline{y_{i1} \dots y_{il_i}} := \overline{y_{i1}} \dots \overline{y_{il_i}} \ .$$

- The set P' does not contain any further productions.

Obviously, SD' is in weak normal form. Finally, it is easy to see that the new productions together produce exactly the effect of the original productions, and hence $\lambda(SD') = \lambda(SD)$.

Statement (vi) is proved by Saers (2011, Theorem 5.1). □

Note that in general not every SDTS can be given in a Chomsky-like normal form as remarked by Aho and Ullman (1972), Huang et al. (2009) or Wu (1997). This will be formally shown in next section by Theorem 3.3.18. On the other hand, restricted versions of SDTSs such as ITGs admit a CNF. Moreover, Huang et al. (2009) present conditions for an SDTS of order k to posses a CNF.

We continue by presenting a more algebraic characterization of syntax-directed translations – the string bimorphism (Aho and Ullman, 1969b).

Definition 3.3.13. A language $L \subseteq Z^*$ is said to

(i) *characterize* a translation $\lambda \subseteq X^* \times Y^*$ if there is a string bimorphism $SB = (\mu, L, \nu)$ with $\mu\colon Z^* \to X^*$ and $\nu\colon Z^* \to Y^*$, such that $\lambda = \lambda(SB)$, and to

(ii) *strongly characterize* a translation $\lambda \subseteq X^* \times Y^*$ if there is a string bimorphism $SB = (\mu, L, \nu)$ such that $\lambda = \lambda(SB)$, and $\mu\colon Z^* \to X^*$ and $\nu\colon Z^* \to Y^*$ are thus:

- $Z = X \cup Y'$ with $X \cap Y' = \emptyset$,
- $\mu(z) := z$ if $z \in X$, and $\mu(z) := \varepsilon$ if $z \in Y'$, and
- $\nu(z) := \varepsilon$ if $z \in X$, and $\nu|_{Y'}$ is a bijection between Y' and Y. □

To clarify the definition, we present Examples 3.14-3.15 of Aho and Ullman (1972).

Example 3.3.14. The language 0^+ characterizes, but does not strongly characterize, the translation $\lambda = \{(x^n, x^n) \mid n \geq 1\}$ ($\mu(0) = \nu(0) := x$). Moreover, λ is strongly characterized by the language $\{x^n y^n \mid n \geq 1\} \subseteq \{x, y\}^*$ ($Z_1 = \{x\}$, $Z_2 = \{y\}$, $\mu(x) := x$, $\mu(y) := \varepsilon$, $\nu(x) := \varepsilon$ and $\nu(y) := y$). □

Now using Definition 3.3.13, we get the following specification methods for various types of syntax-directed translations.

Theorem 3.3.15. *The following statements hold.*

(i) *A translation is regular, if and only if it is strongly characterized by a regular language, if and only if it is characterized by a regular language.*

(ii) *A syntax-directed translation is simple, if and only if it is strongly characterized by a CFL, if and only if it is characterized by a CFL.*

(iii) *A syntax-directed translation is linear, if and only if it is strongly characterized by a linear language, if and only if it is characterized by a linear language.*

Proof. Statement (i) was proved independently by Nivat (1968, Section I.3) and Aho and Ullman (1972, Theorem 3.3) (see also Theorem 3.2.3). Item (ii) is due to Aho and Ullman (1969a, Theorem 1) and Aho and Ullman (1972, Theorem 3.4). Next we exemplify the constructions of Aho and Ullman by showing statement (iii).

First, let $\lambda \subseteq X^* \times Y^*$ be a translation characterized by a linear language. By Definition 3.3.13(i), there exist an alphabet Z and a string bimorphism $SB = (\mu, L, \nu)$ with $\mu\colon Z^* \to X^*$, $\nu\colon Z^* \to Y^*$ and $L \subseteq Z^*, L \in LIN$, such that $\lambda = \lambda(SB)$. Moreover, because $L \in LIN$, there is a linear grammar $LG = (N, Z, P, S)$ such that $L = L(LG)$. Now, we construct a linear SDTS $SD = (N, X, Y, P', S)$, where

(1) for every production $A \to v$ in P with $A \in N$ and $v \in Z^*$, we put $A;A \to \mu(v);\nu(v)$ into P';

(2) for every production $A \to vBw$ in P with $A, B \in N$ and $v, w \in Z^*$, we have $A\,;A \to \mu(v)B\mu(w)\,;\nu(v)B\nu(w)\,(1)$ in P';.

Obviously, SD is a linear SDTS. To prove that $\lambda = \lambda(SB) = \lambda(SD)$, it is enough to show by length of the derivation that for every $A \in N$,

$$(A, A) \Rightarrow_{SD}^{n} (v, w) \quad \Leftrightarrow \qquad \exists v' \in Z^*, A \Rightarrow_{LG}^{n} v', v = \mu(v'), \text{ and } w = \nu(v') \ .$$

Assuming this has be done, we get that λ is a linear translation.

Next, let $\lambda \subseteq X^* \times Y^*$ be a linear translation. Then, there is a linear SDTS $SD = (N, X, Y, P, S)$ such that $\lambda(SD) = \lambda$. Let $Z := X \cup Y'$, where $Y' := \{y' \mid y \in Y\}$ is an alphabet of new symbols. Moreover, we define $\mu\colon Z^* \to X^*$ by setting

$$\mu(z) := z \text{ if } z \in X \quad \text{ and } \quad \mu(z) := \varepsilon \text{ if } z \in Y' \ ,$$

$\nu\colon Z^* \to Y^*$ by

$$\nu(z) := \varepsilon \text{ if } z \in X \quad \text{ and } \quad \nu(z) := y \text{ if } z \in Y' \ ,$$

and $\phi\colon Y^* \to Y'^*$ by $\phi(y) := y'$. Furthermore, we construct a linear grammar $LG = (N, Z, P', S)$, where P' is obtained from P as follows.

(1) For every production $A\,;A \to v\,;w$ in P with $v \in X^*$ and $w \in Y^*$, we add the production $A \to v\phi(w)$ in P'.

(2) For every production $A\,;A \to vBv'\,;wBw'\,(1)$ in P with $A, B \in N$, $v, v' \in X^*$ and $w, w' \in Y^*$, we add the production $A \to v\phi(w)Bv'\phi(w')$ in P'.

Thus, we defined the string bimorphism $SB = (\mu, L(LG), \nu)$.

Now, if one shows by length of the derivation that for every $A \in N$

$$(A, A) \Rightarrow_{SD}^{n} (v, w) \quad \Leftrightarrow \quad \exists v' \in Z^*, A \Rightarrow_{LG}^{n} v', v = \mu(v'), \text{ and } w = \nu(v') \ ,$$

then it easily follows that $\lambda(SD) = \lambda(SB)$. Consequently, the linear translation λ is strongly characterized by the linear language $L(LG)$.

Since it is obvious from Definition 3.3.13 that if a linear language characterize a translation then it also strongly characterize that translation, we may conclude the proof. □

Corollary 3.3.16. *There exist linear SDTs which are not regular and translations defined by ITGs which are not simple SDTs.*

Proof. By Theorem 3.3.15(i), the translation defined by the linear SDTS SD_2 of Example 3.3.4 is not regular (Aho and Ullman, 1972, Example 3.16). Also, by Theorem 3.3.15(ii), the translation defined by the ITG SD_1 of Example 3.3.4 is not a simple SDT (Aho and Ullman, 1969a, Theorem 2). □

Finally, we present a few composition results for various types of SDTSs.

Proposition 3.3.17. *The following hold.*

(i) $\lambda[FST] \circ \lambda[SDTS] \subseteq \lambda[SDTS]$.

(ii) $\lambda[SDTS] \circ \lambda[FST] \subseteq \lambda[SDTS]$.

(iii) $\lambda[FST] \circ \lambda[sSDTS] \subseteq \lambda[sSDTS]$.

(iv) $\lambda[sSDTS] \circ \lambda[FST] \subseteq \lambda[sSDTS]$.

Proof. Aho and Ullman (1968, Section VII) and Aho et al. (1969) showed statements (i)-(iv) by using automaton definitions of these classes of translations. However, for statements (iii) and (iv) we give an alternative elegant proof using the string bimorphism characterization of SDTs given by Theorem 3.3.15.

For (iii), let $\lambda_1 \in \lambda[FST]$ and $\lambda_2 \in \lambda[sSDTS]$. By Theorem 3.3.15(i-ii), there exist the string bimorphisms $SB_1 = (\mu_1, L_1, \nu_1)$ and $SB_2 = (\mu_2, L_2, \nu_2)$ with $L_1 \in REG$ and $L_2 \in CFL$ such that $\lambda(SB_1) = \lambda_1$ and $\lambda(SB_2) = \lambda_2$. Moreover, let $SB = (\mu_1, L, \nu_2)$ be the string bimorphism with $L := \mu_2^{-1}(\nu_1(L_1)) \cap L_2$. By Theorems 4.14 and 4.16 of Hopcroft et al. (2001) and Theorem 2.4.9, L is a CFL, and obviously $\lambda(SB) = \lambda_1 \circ \lambda_2$. Therefore, $\lambda_1 \circ \lambda_2 \in \lambda[sSDTS]$ by Theorem 3.3.15(ii).

For (iv), let $\lambda_1 \in \lambda[sSDTS]$ and $\lambda_2 \in \lambda[FST]$. By Theorem 3.3.15(i-ii), there exist string bimorphisms $SB_1 = (\mu_1, L_1, \nu_1)$ and $SB_2 = (\mu_2, L_2, \nu_2)$ with $L_1 \in CFL$ and $L_2 \in REG$ such that $\lambda(SB_1) = \lambda_1$ and $\lambda(SB_2) = \lambda_2$. Moreover, let $SB = (\mu_1, L, \nu_2)$ be the string bimorphism with $L := \mu_2^{-1}(\nu_1(L_1)) \cap L_2$. Using again Theorem 2.4.9, we get $L \in CFL$. Therefore, $\lambda_1 \circ \lambda_2 = \lambda(SB) \in \lambda[sSDTS]$ by Theorem 3.3.15(ii). □

3.3.4 Hierarchies of syntax-directed translations

We start with an ascending hierarchy that classifies SDTSs according to their order (see Aho and Ullman (1972, Section 3.2.3) or Aho and Ullman (1969b, Section III)). This can be used to show certain connections between different types of syntax-directed translation defining devices.

Theorem 3.3.18. *$\lambda[1SDTS] \subset \lambda[2SDTS] = \lambda[3SDTS]$, and for all $k \geq 3$, it holds that λ[k-SDTS] $\subset \lambda$[(k+1)-SDTS].*

Next, we give another specification method of regular translations: right-linear SDTSs.

Theorem 3.3.19. *$\lambda[FST] = \lambda[rSDTS]$, and the equality is effective.*

Proof. Firstly, given any FST $FT = (Q, X, Y, \kappa, q_0, F)$, we build the right-linear SDTS $SD = (Q, X, Y, P, \{q_0\})$ with P constructed as follows. For every $q \in F$, $q; q \to \varepsilon; \varepsilon$ is in P. Moreover, for all $q, q' \in Q$, $x \in X \cup \{\varepsilon\}$ and $y \in Y \cup \{\varepsilon\}$ such that $(q', y) \in \kappa(q, x)$, add $q; q \to xq'; yq'$ (1) to P. Next, we show by induction on $n \geq 0$ that for all $q, q' \in Q$, $v \in X^*$ and $w \in Y^*$,

$$(q, v, \varepsilon) \vdash_{FT}^{*} (q', \varepsilon, w) \text{ in } n \text{ moves} \quad \text{iff } (q, q) \Rightarrow_{SD}^{n} (vq', wq') \ .$$

To prove the "*only if*" direction we proceed by induction on the number of moves of FT.

(1) If $(q, v, \varepsilon) \vdash_{FT} (q', \varepsilon, w)$, then $v \in X \cup \{\varepsilon\}$, $w \in Y \cup \{\varepsilon\}$ and $(q', w) \in \kappa(q, v)$. So, $q; q \to vq'; wq'$ (1) is in P, and hence $(q, q) \Rightarrow_{SD} (vq', wq')$.

(2) If $(q, v, \varepsilon) \vdash_{FT}^{*} (q', \varepsilon, w)$ in $n \geq 2$ moves, then there exist $q'' \in Q$, $x \in X \cup \{\varepsilon\}$, $v' \in X^*$, $y \in Y \cup \{\varepsilon\}$ and $w' \in Y^*$ such that $(q'', y) \in \kappa(q, x)$ and $(q'', v', \varepsilon) \vdash_{FT}^{*} (q', \varepsilon, w')$ in $n - 1$ moves with $v = xv'$ and $w = yw'$. Thus, $q; q \to xq''; yq''$ (1) is in P, and from the induction assumption, we have $(q'', q'') \Rightarrow_{SD}^{n-1} (v'q', w'q')$. Finally, we get $(q, q) \Rightarrow_{SD}^{n} (vq', wq')$.

For the "*if*" direction, we proceed by induction on the length of derivation witnessing $(q, q) \Rightarrow_{SD}^{n} (vq', wq')$.

(1) If $(q, q) \Rightarrow_{SD} (vq', wq')$, then $q; q \to vq'; wq'$ (1) with $v \in X \cup \{\varepsilon\}$ and $w \in Y \cup \{\varepsilon\}$ is in P. Thus, $(q', w) \in \kappa(q, v)$, and hence $(q, v, \varepsilon) \vdash_{FT} (q', \varepsilon, w)$.

(2) If $(q, q) \Rightarrow_{SD}^{n} (vq', wq')$ for some $n \geq 2$, then there exist $q'' \in Q$, $x \in X \cup \{\varepsilon\}$, $v' \in X^*$, $y \in Y \cup \{\varepsilon\}$ and $w' \in Y^*$ such that $q; q \to xq''; yq''$ (1) is in P and $(q'', q'') \Rightarrow_{SD}^{n-1} (v'q', w'q')$ with $v = xv'$ and $w = yw'$. By the induction hypothesis, we get $(q'', v', \varepsilon) \vdash_{FT}^{*} (q', \varepsilon, w')$ in $n - 1$ moves, and since $(q'', y) \in \kappa(q, x)$, we finally obtain $(q, v, \varepsilon) \vdash_{FT}^{*} (q', \varepsilon, w)$ in exactly n moves.

Now, we can argue as follows:

$$\begin{aligned}(v, w) \in \lambda(FT) &\Leftrightarrow (q_0, v, \varepsilon) \vdash_{FT}^{*} (q, \varepsilon, w) \text{ for some } q \in F \\ &\Leftrightarrow (q_0, q_0) \Rightarrow_{SD}^{*} (vq, wq) \text{ and } q; q \to \varepsilon; \varepsilon \in P \\ &\Leftrightarrow (q_0, q_0) \Rightarrow_{SD}^{*} (v, w) \Leftrightarrow (v, w) \in \lambda(SD) \ .\end{aligned}$$

Hence, $\lambda[FST] \subseteq \lambda[rSDTS]$. For the converse inclusion (cf. Aho and Ullman, 1972, Lemma 3.4), let $SD = (N, X, Y, P, S)$ be any right-linear SDTS in normal form by Theorem 3.3.12(i). We construct the FST $FT = (N \cup \{\star\}, X, Y, \kappa, S, \{\star\})$ with $\star$ a new symbol, and

- for every $A; A \to xB; yB$ (1) in P, $(B, y) \in \kappa(A, x)$;
- for every $A; A \to x; y$ in P, $(\star, y) \in \kappa(A, x)$.

Next, one can prove by induction on $n \geq 0$ that for all $A \in N$, $v \in X^*$ and $w \in Y^*$,

$$(S, v, \varepsilon) \vdash_{FT}^{*} (A, \varepsilon, w) \text{ in } n \text{ moves } \Leftrightarrow (S, S) \Rightarrow_{SD}^{n} (vA, wA) \ .$$

Assuming this has be done similarly as above, it follows that $(v, w) \in \lambda(SD)$ if and only if $(v, w) \in \lambda(FT)$. Consequently, $\lambda[rSDTS] \subseteq \lambda[FST]$, which concludes the proof. □

Now, we investigate the connections between the various types of SDTSs presented in Definition 3.3.3.

Proposition 3.3.20. $\lambda[rSDTS] \subset \lambda[lSDTS] \subset \lambda[sSDTS] \subset \lambda[ITG] \subset \lambda[SDTS]$.

Proof. From Definition 3.3.3 it is clear that

$$\lambda[rSDTS] \subseteq \lambda[lSDTS] \subseteq \lambda[sSDTS] \subseteq \lambda[ITG] \subseteq \lambda[SDTS] \ .$$

By Corollary 3.3.16 and Theorem 3.3.19, we get $\lambda[rSDTS] \subset \lambda[lSDTS]$ and $\lambda[sSDTS] \subset \lambda[ITG]$. Every ITG is of order 2 by Theorem 3.3.12(ii), and since $\lambda[2SDTS] \subset \lambda[4SDTS]$ by Theorem 3.3.18, we have $\lambda[ITG] \subset \lambda[SDTS]$. Clearly, every linear SDTS is of order 1 and simple, and every simple SDTS is of order 2 by Theorem 3.3.12(iii). But $\lambda[1SDTS] \subset \lambda[2SDTS]$ by Theorem 3.3.18, and hence $\lambda[lSDTS] \subset \lambda[sSDTS]$. □

In the sequel we show that by allowing different nonterminals to be associated in an SDTS we do not increase the generative capacity. The construction of a compound nonterminal alphabet was suggested by Huang et al. (2009, p. 565) and by Zhang et al. (2006, p. 258), and the result was mentioned and used in several places in the literature, at least by Satta and Peserico (2005), Chiang and Knight (2006), Satta (2007), and Maletti and Tîrnăucă (2010). Here, we just check the formal details.

Proposition 3.3.21. $\lambda[SCFG] = \lambda[SDTS]$, *and the equality is effective.*

Proof. Since any SDTS is an SCFG with $A = B$ and $A_{\sigma(i)} = B_i$ (for all $i \in [m]$) in each production of the form (3.3.2), the inclusion $\lambda[SDTS] \subseteq \lambda[SCFG]$ is obvious.

To prove the converse, we associate with any given SCFG $SC = (N, X, Y, P, S, S')$ an SDTS $SD = (N \times N, X, Y, P', (S, S'))$ with productions in P' obtained from the productions in P as follows. If $A; B \to \alpha; \beta\,(\sigma)$ is a production of the form (3.3.2) in P, then the production

$$(A, B); (A, B) \to \alpha'; \beta'\,(\sigma) \ ,$$

where

$$\alpha' = v_0(A_1, B_{\sigma^{-1}(1)})v_1 \ldots v_{m-1}(A_m, B_{\sigma^{-1}(m)})v_m$$

and

$$\beta' = w_0(A_{\sigma(1)}, B_1)w_1 \ldots w_{m-1}(A_{\sigma(m)}, B_m)w_m \ ,$$

is included into P'. The linking permutation of this production is σ, too. For example, if

$$A; B \to xxAyCAy; BzAzC\,(2\ 3\ 1)$$

is a SCFG-production, then

$$(A, B); (A, B) \to xx(A, C)y(C, B)(A, A)y; (C, B)z(A, A)z(A, C)\,(2\ 3\ 1)$$

is the corresponding SDTS-production.

It remains to show $\lambda(SC) = \lambda(SD)$. Let $\mu\colon ((N \times N) \cup X)^* \to (N \cup X)^*$ and $\nu\colon ((N \times N) \cup Y)^* \to (N \cup Y)^*$ be the homomorphisms such that $x\mu := x$, $y\nu := y$, $(A,B)\mu := A$ and $(A,B)\nu := B$ for any $x \in X$, $y \in Y$ and $(A,B) \in N \times N$. It is then clear that if $(A,B);(A,B) \to \alpha';\beta'\,(\sigma)$ is a production in P' obtained from the SCFG-production $A;B \to \alpha;\beta\,(\sigma)$ in P, then $\mu(\alpha') = \alpha$ and $\nu(\beta') = \beta$. First we prove that for any derivation

$$(A,B) \Rightarrow_{SC} (\delta_1,\delta_1') \Rightarrow_{SC} \ldots \Rightarrow_{SC} (\delta_n,\delta_n') \tag{3.3.3}$$

in SC, there is a derivation

$$((A,B),(A,B)) \Rightarrow_{SD} (\gamma_1,\gamma_1') \Rightarrow_{SD} \ldots \Rightarrow_{SD} (\gamma_n,\gamma_n') \tag{3.3.4}$$

in SD such that $\delta_i = \gamma_i\mu$ and $\delta_i' = \gamma_i'\nu$ for every $i \in [n]$. We proceed by induction on the length of the derivation (3.3.3).

(1) If $n = 1$, then $(A,B) \Rightarrow_{SC} (\delta_1,\delta_1')$ and $A;B \to \delta_1;\delta_1'\,(\sigma)$ is a production in P with $\delta_1 \in (N \cup X)^*$ and $\delta_1' \in (N \cup Y)^*$. Consequently, $(A,B);(A,B) \to \gamma_1;\gamma_1'\,(\sigma)$ is the corresponding SDTS-production in P' with $\delta_1 = \gamma_1\mu$ and $\delta_1' = \gamma_1'\nu$, and hence $((A,B),(A,B)) \Rightarrow_{SD} (\gamma_1,\gamma_1')$ such that $\gamma_1\mu = \delta_1$ and $\gamma_1'\nu = \delta_1'$.

(2) Let (3.3.3) be a derivation of length $n \geq 2$, and suppose the assertion holds for all shorter derivations. Thus, there is a derivation

$$(A,B) \Rightarrow_{SC} (\delta_1,\delta_1') \Rightarrow_{SC} (\delta_2,\delta_2') \Rightarrow_{SC} \ldots \Rightarrow_{SC} (\delta_{n-1},\delta_{n-1}')$$

of length $n-1$ in SC, and a production $A;B \to \delta;\delta'\,(\sigma)$ of the form (3.3.2) in P such that $\delta_{n-1} = \alpha A\beta$, $\delta_{n-1}' = \alpha' B\beta'$, $\delta_n = \alpha\delta\beta$, $\delta_n' = \alpha'\delta'\beta'$, and the A and the B singled out are associated in the pair $(\delta_{n-1}',\delta_{n-1}')$. So, $(A,B);(A,B) \to \gamma;\gamma'\,(\sigma)$ is the corresponding SDTD-production in P' with $\delta = \gamma\mu$ and $\delta' = \gamma'\nu$, and this means that $(\gamma_{n-1},\gamma_{n-1}') \Rightarrow_{SD} (\gamma_n,\gamma_n')$. Obviously, $\delta_n = \gamma_n\mu$ and $\delta_n' = \gamma_n'\nu$. Furthermore, by the induction assumption, we have the derivation

$$((A,B),(A,B)) \Rightarrow_{SD} (\gamma_1,\gamma_1') \Rightarrow_{SD} (\gamma_2,\gamma_2') \Rightarrow_{SD} \ldots \Rightarrow_{SD} (\gamma_{n-1},\gamma_{n-1}')$$

of length $n-1$ in SD such that $\delta_i = \gamma_i\mu$ and $\delta_i' = \gamma_i'\nu$ for every $i \in [n-1]$, which concludes the induction step.

In particular, if $(\delta_n,\delta_n') \in \lambda(SC)$, then $\delta_n \in X^*$ and $\delta_n' \in Y^*$, and hence $\delta_n = \gamma_n\mu = \gamma_n$ and $\delta_n' = \gamma_n'\nu = \gamma_n'$, which implies $(\delta_n,\delta_n') \in \lambda(SD)$. So, $\lambda(SC) \subseteq \lambda(SD)$.

For $\lambda(SD) \subseteq \lambda(SC)$, it suffices to verify by induction on the length of the derivation that for every derivation (3.3.4) there is a derivation (3.3.3) such that $\delta_i = \gamma_i\mu$ and $\delta_i' = \gamma_i'\nu$ for every $i \in [n]$. Assuming this has be done similarly as above, if $(\gamma_n,\gamma_n') \in \lambda(SD)$, then $\gamma_n \in X^*$ and $\gamma_n' \in Y^*$, and hence $\gamma_n\mu = \gamma_n$ and $\gamma_n'\nu = \gamma_n'$. So, $(\gamma_n,\gamma_n') \in \lambda(SC)$, which concludes the proof. □

We conclude by gathering all the results of this section in the following.

Theorem 3.3.22. *The inclusion relations between the classes $\lambda[FST]$, $\lambda[ITG]$, $\lambda[SDTS]$, $\lambda[SCFG]$, $\lambda[lSDTS]$ and $\lambda[sSDTS]$ are given by the* Hasse *diagram of Figure 3.3.1.*

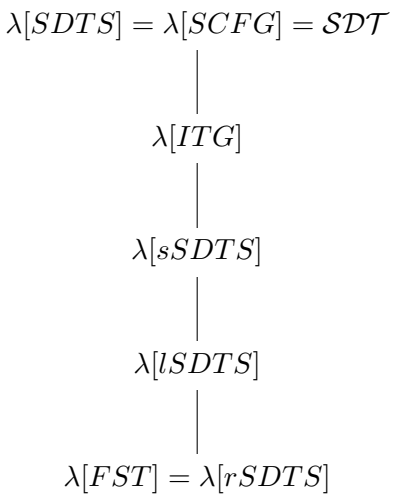


Figure 3.3.1: HASSE diagram of various types of syntax-directed translations.

Proof. It follows from Theorem 3.3.19 and Propositions 3.3.20 and 3.3.21. □

Later, in Section 6.2.2, we introduce another specification method for syntax-directed translations which involves the parse trees of such synchronous grammars, and which will permit the improvement of the mathematical foundations of SDTSs and related formalisms.

CHAPTER 4

Trees and Tree Languages

This chapter is a short introduction to tree languages and tree automata. The emphasis is on finite tree automata and regular tree languages as these certainly form the core of the whole monograph. Special attention is also paid to the connections between regular tree languages and context-free languages. We followed the notations, terminology and presentation of Steinby (2005, 2004) and Engelfriet (1975c), but also Gécseg and Steinby (1984), Gécseg and Steinby (1997), and Comon et al. (2007) have been consulted.

4.1 Trees and contexts

The trees considered here are finite (finitely many nodes and branches), labeled (the nodes are labeled with symbols from some alphabet), rooted (there is one node, the root, with no branches entering it), directed (the branches are going downwards) and ordered (the branches leaving any given node are ordered from left to right). Such trees are commonly and conveniently defined as terms (cf. Thatcher and Wright, 1968, Doner, 1970, Burris and Sankappanavar, 1981, Gécseg and Steinby, 1984, Gécseg and Steinby, 1997, Baader and Nipkow, 1998, for example).

A *ranked alphabet* (Σ, rk) consists of an alphabet Σ and a mapping $\mathrm{rk}\colon \Sigma \to \mathbb{N}$ which assigns to each symbol in Σ a non-negative integer *rank*. Usually we leave the mapping rk implicit. For any $m \in \mathbb{N}$, let $\Sigma_m := \{f \in \Sigma \mid \mathrm{rk}(f) = m\}$. Symbols of ranks 0, 1 and 2 are called *constants*, and *unary* and *binary* symbols, respectively. We may write $\Sigma = \{f_1/m_1, \ldots, f_k/m_k\}$ when Σ consists of the symbols $f_1, \ldots, f_k$ with the respective ranks $m_1, \ldots, m_k$. In what follows Σ, Ω and Γ are always ranked alphabets. The letters e, f, g and h are reserved for symbols in the ranked alphabets. If U is a set of symbols, then

$$\Sigma(U) := \{f(u_1, \ldots, u_m) \mid m \geq 0, f \in \Sigma_m \text{ and } u_1, \ldots, u_m \in U\} \ .$$

Besides ranked alphabets, we use also ordinary alphabets X, Y and Z for labeling leaves of trees. In this context, they are called *leaf alphabets*, and they are assumed to be disjoint from the ranked alphabets.

Definition 4.1.1. The set $T_\Sigma(X)$ of Σ-*terms over* X is the smallest set $\mathcal{T}$ such that

(1) $X \cup \Sigma_0 \subseteq \mathcal{T}$, and

(2) $f(t_1, \ldots, t_m) \in \mathcal{T}$ whenever $m \geq 1$, $f \in \Sigma_m$ and $t_1, \ldots, t_m \in \mathcal{T}$.

When regarded as representations of labeled trees, we call them ΣX-*trees*. □

Any $d \in X \cup \Sigma_0$ represents a tree with only one node which is labeled with d. Similarly, $f(t_1, \ldots, t_m)$ is interpreted as a tree formed by adjoining the m trees represented by $t_1, \ldots, t_m$ to a new root labeled with f. Note that any $t \in T_\Sigma(X)$ is represented as a

Figure 4.1.1: The ΣX-context $f(g(x), \xi)$ (on the right) and the ΣX-tree represented by $f(g(x, f(e, x)))$ (on the left). The alphabets are given in Example 4.1.3.

special string over $\Sigma \cup X \cup Z$, where Z consists of the parentheses (and) and the comma (see Aho and Ullman (1972, Section 0.5.7) or (Engelfriet, 1975c, Section 2), for example). Also observe that $T_\Sigma(X)$ is a CFL over $\Sigma \cup X \cup Z$. For every $t \in T_\Sigma(X)$ and $f \in \Sigma$, $|t|_f$ is the number of nodes in t labeled with the symbol f. Analogously, $|t|_x$ is the number of leaves in t labeled with the letter x. A tree $t \in T_\Sigma(X)$ is *linear* (respectively, *non-deleting*) in $Y \subseteq X$ if $|t|_y \leq 1$ (respectively, $|t|_y \geq 1$) for every $y \in Y$. For every $t \in T_\Sigma(X)$ and $g \in \Sigma_1$, we let $g^0(t) := t$ and $g^{n+1}(t) := g(g^n(t))$ for all $n \in \mathbb{N}$. If $X = \emptyset$ in Definition 4.1.1, then $T_\Sigma(X)$ becomes the set T_Σ of Σ-*trees*. We will also speak generally about *trees* without specifying the alphabets. The letters r, s, t and u are reserved for trees.

Remark 4.1.2. As the definition of the set $T_\Sigma(X)$ is *inductive*, notions related to ΣX-trees can be defined *recursively* and statements about them can be proved by *tree induction*. For example, the *yield* $\mathrm{yd}(t)$, the *height* $\mathrm{hg}(t)$, the set $\mathrm{sub}(t)$ of *subtrees*, the (label of the) *root* $\mathrm{root}(t)$, and the set $\mathrm{fork}(t)$ of *forks* of a ΣX-tree t are defined thus:

(1) $\mathrm{yd}(x) := x$, $\mathrm{hg}(x) := 0$, $\mathrm{sub}(x) := \{x\}$, $\mathrm{root}(x) := x$ and $\mathrm{fork}(x) := \emptyset$ for $x \in X$;

(2) $\mathrm{yd}(e) := \varepsilon$, $\mathrm{hg}(e) := 0$, $\mathrm{sub}(e) := \{e\}$, $\mathrm{root}(e) := e$ and $\mathrm{fork}(e) := \emptyset$ for $e \in \Sigma_0$;

(3) $\mathrm{yd}(t) := \mathrm{yd}(t_1)\mathrm{yd}(t_2)\ldots\mathrm{yd}(t_m)$, $\mathrm{hg}(t) := \max\{\mathrm{hg}(t_1), \ldots, \mathrm{hg}(t_m)\} + 1$, $\mathrm{sub}(t) := \{t\} \cup \mathrm{sub}(t_1) \cup \ldots \cup \mathrm{sub}(t_m)$, $\mathrm{root}(t) := f$ and $\mathrm{fork}(t) := \mathrm{fork}(t_1) \cup \ldots \cup \mathrm{fork}(t_m) \cup \{f(\mathrm{root}(t_1), \ldots, \mathrm{root}(t_m))\}$ for $t = f(t_1, \ldots, t_m)$ $(m \geq 1)$.

The (finite) set of all possible forks of ΣX-trees is denoted by $\mathrm{fork}(\Sigma, X)$. To prove by tree induction that a statement $\mathcal{S}(t)$ holds for all ΣX-trees t, one shows that

(1) $\mathcal{S}(t)$ holds for every $t \in X \cup \Sigma_0$, and that

(2) if $t = f(t_1, \ldots, t_m)$, then $\mathcal{S}(t)$ follows from the assumption that $\mathcal{S}(t_1), \ldots, \mathcal{S}(t_m)$ all hold (the *inductive assumption*). □

For any $D \subseteq X \cup \Sigma_0$, the *generalized yield-mapping* $\mathrm{yd}_D \colon T_\Sigma(X) \to D^*$ is defined inductively by $\mathrm{yd}_D(d) := d$ for every $d \in D$, $\mathrm{yd}_D(d) := \varepsilon$ for every $d \in (X \cup \Sigma_0) \setminus D$, and $\mathrm{yd}_D(f(t_1, \ldots, t_m)) := \mathrm{yd}_D(t_1) \ldots \mathrm{yd}_D(t_m)$ for every $f \in \Sigma_m$ $(m \geq 1)$ and $t_1, \ldots, t_m \in T_\Sigma(X)$. In particular, $\mathrm{yd}_X = \mathrm{yd}$.

Let ξ be a special symbol of rank 0 not in Σ or X. A $\Sigma(X \cup \{\xi\})$-tree in which ξ appears exactly once is called a ΣX-*context*. The set of all ΣX-contexts is denoted by $C_\Sigma(X)$. If $c, c' \in C_\Sigma(X)$, then $c'(c)$ is the ΣX-context obtained from c' by replacing the ξ in it with c. Similarly, if $t \in T_\Sigma(X)$ and $c \in C_\Sigma(X)$, then $c(t)$ is the ΣX-tree obtained when the ξ in c is replaced with t. Moreover, for any $c \in C_\Sigma(X)$, $c^0 := \xi$ and $c^{n+1} := c(c^n)$ for every $n \in \mathbb{N}$. Again by setting $X = \emptyset$, $C_\Sigma(X)$ becomes the set C_Σ of Σ-*contexts*.

Example 4.1.3. Let $\Sigma = \{f/2, g/1, e/0\}$ and $X = \{x\}$. Then, $t = f(g(x), f(e, x))$ is a ΣX-tree, and $c = f(g(x), \xi)$ is a ΣX-context, both being depicted in Figure 4.1.1. Then, t is non-deleting but not linear in X, $|t|_x = 2$, $|t|_g = 1$, $\mathrm{root}(t) = f$, $\mathrm{hg}(t) = 2$, $\mathrm{sub}(t) = \{t, g(x), f(e, x), e, x\}$ and $\mathrm{fork}(t) = \{f(g, f), g(x), f(e, x)\}$. Note that $\mathrm{fork}(\Sigma, X)$ has 20 elements. Now, $\mathrm{yd}_X(t) = \mathrm{yd}(t) = xx$, and if $D = \{e, x\}$, then $\mathrm{yd}_D(t) = xex$. Moreover, $c(f(e, x)) = t$, and, if $f(e, \xi)$ is another ΣX-context, $c(f(e, \xi)) = f(g(x), f(e, \xi))$.

Besides the term representation, there is another customary method to formally define trees that makes use of the Dewey notation (Gécseg and Steinby, 1984, Gécseg and Steinby, 1997). It uses a set, called domain, of sequences of non-negative integers plus the empty sequence ε that in a natural way represent the 'nodes' of a tree. The integers appearing in such a sequence are separated by a dot (cf. Gorn, 1967). However, there are simple translations between the two representations of trees.

Any ΣX-tree defined as a term according to Definition 4.1.1 can be redefined as a tree which uses the Dewey notation as follows. Firstly, we formally define the *domain* $\mathrm{dom}(t)$ of a ΣX-tree t by setting:

(1) $\mathrm{dom}(t) := \{\varepsilon\}$ for $t \in X \cup \Sigma_0$;

(2) $\mathrm{dom}(t) := \{\varepsilon\} \cup \{i.\omega \mid i \in [m], \omega \in \mathrm{dom}(t_i)\}$ for $t = f(t_1, \ldots, t_m)$ $(m \geq 1)$.

We omit $.\varepsilon$ at the end of a sequence in $\mathrm{dom}(t)$. The symbol ω is reserved to always denote an element of a domain of a tree.

Now, we define the mapping $\hat{t}\colon \mathrm{dom}(t) \to X \cup \Sigma$ thus:

(1) if $t \in X \cup \Sigma_0$, then $\hat{t}(\varepsilon) := t$;

(2) if $t = f(t_1, \ldots, t_m)$, then $\hat{t}(\varepsilon) := f$ and $\hat{t}(i.\omega) := \hat{t}_i(\omega)$ for every $i \in [m]$, $\omega \in \mathrm{dom}(t_i)$.

For example, the ΣX-tree $f(g(x), f(e, x))$ of Figure 4.1.1 can be redefined as the mapping $\hat{t}\colon \{\varepsilon, 1, 2, 1.1, 2.1, 2.2\} \to X \cup \Sigma$, $\varepsilon \mapsto f$, $1 \mapsto g$, $2 \mapsto f$, $1.1 \mapsto x$, $2.1 \mapsto e$, $2.2 \mapsto x$.

Conversely, let V be the set of all finite sequences of positive integers separated by dots. In particular, it includes the empty sequence ε. The prefix relation $\leq$ in V is defined thus: $\omega \leq \omega'$ if and only if $\omega.\omega'' = \omega'$ for some $\omega'' \in V$. A finite subset W of V is a (finite) *tree domain* if it satisfies the following two conditions:

(1) if $\omega \leq \omega'$ and $\omega' \in W$, then $\omega \in W$;

(2) if $\omega.j \in W$, $i, j \in \mathbb{N}^*$, and $i < j$, then $\omega.i \in W$.

A ΣX-tree can now be defined as a mapping $t\colon W \to \Sigma \cup X$, where W is a tree domain and for each $\omega \in W$,

(1) $t(\omega) \in \Sigma_0 \cup X$ if $\nexists i \in \mathbb{N}^+$ such that $\omega.i \in W$, and

(2) $t(\omega) \in \Sigma_m$, where $m = \max\{i \in \mathbb{N}^+ \mid i.\omega \in W\}$, otherwise.

We mostly use the term representation of trees, but in some cases it is useful if we employ the elements of $\mathrm{dom}(t)$ for referring to them as *nodes* of t or *positions* in t. For example, the *label* $t(\omega)$ of a ΣX-tree t at position $\omega \in \mathrm{dom}(t)$, the *subtree* $t|_\omega$ of t *rooted at node* ω and the *replacement* $t[u]_\omega$ of the subtree of t rooted at node ω by the ΣX-tree u are easily defined. Moreover, for every $D \subseteq X \cup \Sigma$, let $\mathrm{dom}_D(t) := \{\omega \in \mathrm{dom}(t) \mid t(\omega) \in D\}$. Then,

the set of *branches* from the root to the leaves of a ΣX-tree t is $\mathrm{br}(t) := \mathrm{dom}_{X\cup\Sigma_0}(t)$. Other basic notions and operations regarding trees are equally easy to define (cf. Remark 4.1.2). As an example, $\mathrm{sub}(t) := \{t|_\omega \mid \omega \in \mathrm{dom}(t)\}$ for any $t \in T_\Sigma(X)$. Finally, we say that two trees s and t, possibly over different alphabets, have the *same shape* if $\mathrm{dom}(s) = \mathrm{dom}(t)$.

Example 4.1.4. Consider the ΣX-tree $t = f(g(x), f(e, x))$ of Figure 4.1.1. Then, $\mathrm{dom}(t) = \{\varepsilon, 1, 2, 1.1, 2.1, 2.2\}$, the label at position 2.2 is $t(2.2) = x$, the subtree rooted at 1 is $t|_1 = g(x)$, and $t[e]_2 = f(g(x), e)$. Finally, $\mathrm{dom}_{\{f\}}(t) = \{\varepsilon, 2\}$ and $\mathrm{br}(t) = \{1.1, 2.1, 2.2\}$. □

Now, let $\Xi = \{\xi_1, \xi_2, \ldots\}$ be a countably infinite set of symbols called *variables* disjoint from any leaf alphabet or ranked alphabet considered here. For any $n \in \mathbb{N}$, let $\Xi_n = \{\xi_1, \ldots, \xi_n\}$. These variables indicate places into which trees may be substituted. For any Σ, X and $n \in \mathbb{N}$,

$$\widetilde{T}_\Sigma(X \cup \Xi_n) := \{t \in T_\Sigma(X \cup \Xi_n) \mid \mathrm{yd}_{\Xi_n}(t) = \xi_1\xi_2\ldots\xi_n\} \ ,$$

and

$$C^n_\Sigma(X) := \{t \in T_\Sigma(X \cup \Xi_n) \mid |t|_{\xi_i} = 1 \text{ for all } i \in [n]\} \ .$$

In other words, $C^n_\Sigma(X)$ contains all those trees of $T_\Sigma(X\cup\Xi_n)$ in which each variable $\xi_1, \ldots, \xi_n$ occurs exactly once. If $t \in T_\Sigma(X \cup \Xi_n)$ and $t_1, \ldots, t_n \in T_\Sigma(Y)$ for some X, Y and $n \in \mathbb{N}$, then $t[t_1, \ldots, t_n]$ denotes the $\Sigma(X \cup Y)$-tree obtained from t by simultaneously replacing in it each occurrence of ξ_i with t_i for all $i \in [n]$. Now, if $x \in X$ and $n = |t|_x$, $t[x \leftarrow (t_1, \ldots, t_n)]$ denotes the result of replacing, for every $i \in [n]$, the i-th (with respect to the lexicographic order on the positions) occurrence of x in the ΣX-tree t by t_i. Moreover, $\mathrm{var}(t)$ is the set of all variables $\xi_i \in \Xi_n$ appearing in $t \in T_\Sigma(X \cup \Xi_n)$.

A ΣX-*substitution* is a finite set of pairs (ξ_i, s), where $\xi_i \in \Xi$, $s \in T_\Sigma(X)$ and no variable ξ_i appears in two different pairs. If $t \in T_\Sigma(X \cup \Xi)$ and $\theta = \{(\xi_{i_1}, s_1), \ldots, (\xi_{i_m}, s_m)\}$ is a ΣX-substitution, then $\theta(t)$ is the $\Sigma(X\cup\Xi)$-tree obtained by replacing the occurrences of the variables $\xi_{i_1}, \ldots, \xi_{i_m}$ in t by the corresponding trees $s_1, \ldots, s_m$. If $\mathrm{var}(t)$ is a subset of the set of variables occurring in the substitution θ, then $\theta(t)$ is a ΣX-tree and $\theta(t) = t[s_1, \ldots, s_m]$. Moreover, for any ΣX-substitution θ, a ΣX-tree t is said to be an *instance* of a $\Sigma(X\cup\Xi)$-tree u with variables if $t = \theta(u)$. For example, if $\theta = \{(\xi_1, f(x, x)), (\xi_3, x)\}$ and $t = f(\xi_1, f(c, \xi_1))$, then $\theta(t) = f(f(x, x), f(c, f(x, x)))$. Thus, the ΣX-tree $f(f(x, x), f(c, f(x, x)))$ is an instance of t. The symbol θ will always denote a substitution.

4.2 Tree homomorphisms

Now we introduce mappings that may transform trees quite radically.

Definition 4.2.1. A *tree homomorphism* $\varphi\colon T_\Sigma(X) \to T_\Omega(Y)$ is determined by a mapping $\varphi_X\colon X \to T_\Omega(Y)$ and mappings $\varphi_m\colon \Sigma_m \to T_\Omega(Y \cup \Xi_m)$, for all $m \geq 0$ such that $\Sigma_m \neq \emptyset$, as follows:

(1) $x\varphi := \varphi_X(x)$ for $x \in X$,

(2) $e\varphi := \varphi_0(e)$ for $e \in \Sigma_0$, and

(3) $t\varphi := \varphi_m(f)[t_1\varphi, \ldots, t_m\varphi]$ for $t = f(t_1, \ldots, t_m)$ $(m \geq 1)$. □

In other words, a tree homomorphism φ recursively processes a tree t starting from its root and going towards the leaves by applying the mapping corresponding to the current examined symbol: if it is a symbol f of rank $m \geq 1$, then it replaces f in t with the corresponding tree $\varphi_m(f)$ in which variables may appear as leaf symbols (given by the mappings of type (3)). Next, each such variable ξ_i ($i \in [m]$) will be replaced by the corresponding subtree $t_i\varphi$ ($i \in [m]$) of t. Observe that this is done inductively using the family of mappings $(\varphi_m)_{m\geq 1}$ and repeating the same procedure as above. Moreover, for each $i \in [m]$, ξ_i is a place holder in $\varphi_m(f)$ for $t_i\varphi$, and a presence of a ξ_i can be seen as a call for computing $t_i\varphi$. The tree homomorphism recursively continues this way until it reaches the leaves of t: each constant e is replaced by the tree $\varphi_0(e)$ in $T_\Omega(Y)$ (given by the mapping of type (2)), and each x in X is substituted by the corresponding tree $\varphi_X(x)$ in $T_\Omega(Y)$ (given by the mapping of type (1)). Note that the processing of $t\varphi$ can be done also in a 'bottom-up' fashion: starting at the leaves and going upwards to the root.

Depending on the restrictions imposed on each tree $\varphi_m(f)$ ($m \geq 1$) and implicitly on its variables, we may find several types of tree homomorphisms in the literature (cf. Thatcher, 1973, Engelfriet, 1975a, c, Arnold and Dauchet, 1978a, 1982, Gécseg and Steinby, 1984, Gécseg and Steinby, 1997, Steinby, 1986, Bozapalidis, 1992, Comon et al., 2007, Steinby and Tîrnăucă, 2007, 2009, for example). In particular, a tree homomorphism $\varphi\colon T_\Sigma(X) \to T_\Omega(Y)$ is said to be:

- *linear* if for all $m \geq 1$ and $f \in \Sigma_m$, $\varphi_m(f)$ is linear in Ξ_m; otherwise, it is called *non-linear*;

- *non-deleting* if for all $m \geq 1$ and $f \in \Sigma_m$, $\varphi_m(f)$ is non-deleting in Ξ_m; otherwise, it is called *deleting*;

- *strict* if no $\varphi_m(f)$, with $f \in \Sigma_m$ and $m \geq 1$, is of the form ξ_i ($i \in [m]$);

- *normalized* if for all $m \geq 1$ and $f \in \Sigma_m$, we have $\varphi_m(f) \in \widetilde{T}_\Sigma(X \cup \Xi_n)$ for some $n \in \mathbb{N}$;

- *alphabetic* if $\varphi_X(x) \in Y$ for every $x \in X$, $\varphi_0(e) \in \Omega_0$ for every $e \in \Sigma_0$, and for all $m \geq 1$ and $f \in \Sigma_m$,
$$\varphi_m(f) = g(\xi_1, \ldots, \xi_m)$$
for some $g \in \Omega_m$;

- *permuting* if $\varphi_X(x) \in Y$ for every $x \in X$, $\varphi_0(e) \in \Omega_0$ for every $e \in \Sigma_0$, and for all $m \geq 1$ and $f \in \Sigma_m$,
$$\varphi_m(f) = g(\xi_{\sigma(1)}, \ldots, \xi_{\sigma(m)})$$
for some permutation σ of $[m]$ and some $g \in \Omega_m$;

- *quasi-alphabetic* if $\varphi_X(x) \in Y$ for every $x \in X$, and for all $m \geq 0$ and $f \in \Sigma_m$,
$$\varphi_m(f) = g(y_{01}, \ldots, y_{0k_0}, \xi_{\sigma(1)}, y_{11}, \ldots, y_{m-1k_{m-1}}, \xi_{\sigma(m)}, y_{m1}, \ldots, y_{mk_m}),$$
where σ is a permutation of $[m]$, $k_i \in \mathbb{N}$ and $y_{i1}, \ldots, y_{ik_i} \in Y$ for every $0 \leq i \leq m$, and $g \in \Omega_{m'}$ for $m' = m + k_0 + k_1 + \ldots + k_m$;

- *symbol-to-symbol* if $\varphi_X(x) \in Y$ for every $x \in X$, $\varphi_0(e) \in \Omega_0$ for every $e \in \Sigma_0$, and for all $m \geq 1$ and $f \in \Sigma_m$,
$$\varphi_m(f) = g(\xi_{i_1}, \ldots, \xi_{i_k})$$
for some $k \geq 1$, $g \in \Omega_k$ and $1 \leq i_1, \ldots, i_k \leq m$;

- *fine* if $\varphi_X(x) \in Y$ for every $x \in X$, $\varphi_0(e) \in \Omega_0$ for every $e \in \Sigma_0$, and for all $m \geq 1$ and $f \in \Sigma_m$, either $\varphi_m(f) = \xi_i$ for some $i \in [m]$ or
$$\varphi_m(f) = g(\xi_{i_1}, \ldots, \xi_{i_k})$$
for some $k \in [m]$, $g \in \Omega_k$ and $1 \leq i_1, \ldots, i_k \leq m$ ($i_j \neq i_l$ for every $j, l \in [k]$ and $j \neq k$).

If $X = \emptyset$, i.e., for tree homomorphisms of the form $\varphi\colon T_\Sigma \to T_\Omega(Y)$, the mapping φ_X and any conditions concerning it can be omitted.

We denote by lH, nH, sH, aH, pH, qH, ssH, and fH the classes of all linear, non-deleting, strict, alphabetic, permuting, quasi-alphabetic, symbol-to-symbol, and fine tree homomorphisms, respectively. Further subclasses of tree homomorphisms can be obtained by combining any of these restrictions. For example, lnH is the class of all linear non-deleting tree homomorphisms. We reserve υ, φ and ψ to always denote tree homomorphisms.

Now let us informally describe the features of the various types defined above, as well as the connections and differences between them. Linear means that in each $\varphi_m(f)$ ($m \geq 1$, $f \in \Sigma_m$), no ξ_i ($i \in [m]$) appears more than once, i.e., no subtree is copied during the application of the mappings φ_m. For a non-deleting tree homomorphism, every ξ_i ($i \in [m]$) appears at least once in each $\varphi_m(f)$ ($m \geq 1$, $f \in \Sigma_m$), i.e., no subtree is erased during the application of the mappings φ_m. A tree homomorphism is strict if no $\varphi_m(f)$ ($m \geq 1$, $f \in \Sigma_m$) is equal to one of its direct subtrees. An alphabetic tree homomorphism just relabels the nodes of a tree since after applying the mappings the resulting tree has the same shape as the processed tree; it is linear, non-deleting and normalized. A permuting tree homomorphism not only relabels the nodes of a tree, but may also reorder its subtrees. A quasi-alphabetic tree homomorphism also relabels the nodes of a tree and may reorder subtrees but in addition, letters from the leaf alphabet Y may appear as direct subtrees of each node. Moreover, a constant is mapped to another constant or to a tree of height 1. In a symbol-to-symbol tree homomorphism the nodes of a tree are relabeled and its subtrees may be reordered, deleted or copied. A fine tree homomorphism does not allow copying, and it may relabel the nodes of a tree, replace a subtree by one of its subtrees or reorder and delete subtrees of a tree.

Example 4.2.2. Let $\Sigma = \{f/3, g/2, e/0\}$, $\Omega = \{h/5, f/3, g/2, e/0\}$, $X = \{x\}$, and $Y = \{0, 1\}$. Consider the six tree homomorphisms $\varphi, \psi, \upsilon, \varphi', \psi', \upsilon'\colon T_\Sigma(X) \to T_\Omega(Y)$ defined by setting

$\varphi_3(f) := g(\xi_2, \xi_1)$	$\varphi_2(g) := g(\xi_1, \xi_1)$	$\varphi_0(e) := e$	$\varphi_X(x) := 0$
$\psi_3(f) := \xi_2$	$\psi_2(g) := g(\xi_2, \xi_1)$	$\psi_0(e) := e$	$\psi_X(x) := 1$
$\upsilon_3(f) := f(\xi_3, \xi_1, \xi_2)$	$\upsilon_2(g) := g(\xi_1, \xi_2)$	$\upsilon_0(e) := e$	$\upsilon_X(x) := 1$
$\varphi'_3(f) := f(\xi_1, \xi_2, g'(e, \xi_3))$	$\varphi'_2(g) := g(\xi_1, \xi_2)$	$\varphi'_0(e) := e$	$\varphi'_X(x) := 0$
$\psi'_3(f) := h(\xi_3, 0, \xi_1, \xi_2, 1)$	$\psi'_2(g) := g(\xi_1, \xi_2)$	$\psi'_0(e) := f(1, 0, 1)$	$\psi'_X(x) := 1$
$\upsilon'_3(f) := f(\xi_1, \xi_2, \xi_3)$	$\upsilon'_2(g) := g(\xi_1, \xi_2)$	$\upsilon'(e) := e$	$\upsilon'_X(x) := 0$.

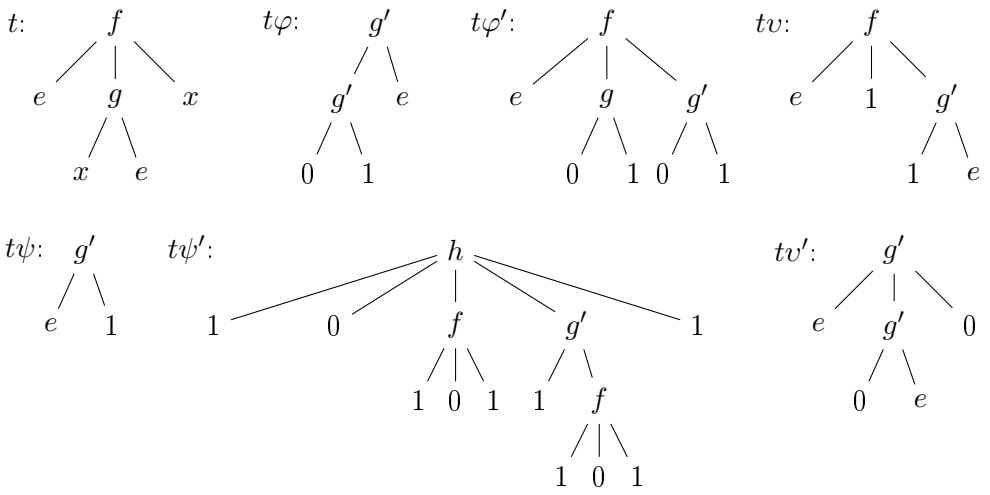


Figure 4.2.1: Application of various tree homomorphisms.

Then, φ is symbol-to-symbol, non-linear, deleting and not normalized, ψ is fine, not normalized, linear nor deleting, υ is permuting but not normalized, φ' is linear, non-deleting and normalized, ψ' is quasi-alphabetic and not normalized, and υ' is alphabetic and hence normalized. In addition, φ, υ, φ', ψ' and υ' are strict but ψ is not. For the ΣX-tree $t = f(e, g(x, e), x)$, we get

$$\begin{array}{lll} t\varphi = g'(g'(0,0),e) & t\varphi' = f(e, g'(0,e), g'(e,0)) & t\upsilon = f(1, e, g'(1,e)) \\ t\psi = g'(e,1) & t\psi' = h(1,0,f(1,0,1), g'(1,f(1,0,1)),1) & t\upsilon' = f(e, g'(0,e), 0) \ . \end{array}$$

The processed tree t and the obtained trees are displayed in Figure 4.2.1. □

Note that any normalized tree homomorphism is linear but not necessarily strict or non-deleting. Any fine tree homomorphism is linear, it may not be strict and may be deleting. Every symbol-to-symbol tree homomorphism is strict but may be non-linear and deleting. On the other hand, every quasi-alphabetic tree homomorphism is strict, linear, non-deleting and symbol-to-symbol. Clearly, any permuting tree homomorphism is quasi-alphabetic. Finally, an alphabetic tree homomorphism means a normalized permuting tree homomorphism. These observations immediately yield the following proposition.

Proposition 4.2.3. *Let* $\varphi\colon T_\Sigma(X) \to T_\Omega(Y)$ *be a tree homomorphism and* $t \in T_\Sigma(X)$.

(i) *If* φ *is quasi-alphabetic, then* $\mathrm{hg}(t) \le \mathrm{hg}(t\varphi) \le \mathrm{hg}(t) + 1$.

(ii) *If* φ *is symbol-to-symbol, then* $\mathrm{hg}(t\varphi) \le \mathrm{hg}(t)$.

(iii) *If* φ *is permuting, then* $\mathrm{hg}(t\varphi) = \mathrm{hg}(t)$.

Also, we note the following facts to be used later.

Remark 4.2.4. If $\varphi \in \mathrm{pH}$, $\psi \in \mathrm{qH}$ and $\upsilon \in \mathrm{aH}$, then $\varphi \circ \psi, \upsilon \circ \psi \in \mathrm{qH}$. □

4.3 Tree languages

Any subset of $T_\Sigma(X)$ is called a ΣX-*tree language*. We reserve the letters T and U to always denote tree languages. Such a ΣX-tree language T is *almost variable-free* if $T \subseteq T_\Sigma \cup X$. Moreover, if the leaf alphabet X is empty, then $T \subseteq T_\Sigma$ is a Σ-tree language, and we call it *variable-free*. We may also speak about *tree languages* without specifying the alphabets. If Fam is any family of tree languages, we denote the sets of ΣX-tree languages and Σ-tree languages in Fam by $\mathrm{Fam}_\Sigma(X)$ and Fam_Σ, respectively. Moreover, $\mathrm{Fam}^{\mathrm{vf}}$ ($\mathrm{Fam}^{\mathrm{avf}}$, respectively) denotes the family of all variable-free (almost-variable free, respectively) tree languages in Fam.

Now we turn our attention to operations on tree languages. Since any tree language is just a set, the operations of union, intersection, Cartesian product, difference and complementation are defined for tree languages in the usual set-theoretic way (see Section 2.1).

Next, we present operations on trees that may be regarded as generalizations of the concatenation of languages. First, for all $T, T_1, \ldots, T_n \subseteq T_\Sigma(X \cup \Xi_n)$,

$$T[T_1, \ldots, T_n] := \{t[t_1, \ldots, t_n] \mid t \in T, t_1 \in T_1, \ldots, \ldots t_n \in T_n\} \ .$$

For any $f \in \Sigma_m$ ($m \geq 1$), the *f-concatenation* of m tree languages $T_1, \ldots, T_m \subseteq T_\Sigma(X \cup \Xi_n)$ is the set $f(T_1, \ldots, T_m) := \{f(t_1, \ldots, t_m) \mid t_1 \in T_1, \ldots, t_m \in T_m\}$. For any given $x \in X$, the *x-product* $T \bullet_x U$ of two tree languages $T, U \subseteq T_\Sigma(X)$ is

$$T \bullet_x U := \{t[x \leftarrow (u_1, \ldots, u_n)] \mid t \in T, n = |t|_x, \text{ and } u_1, \ldots, u_n \in U\} \ .$$

In other words, the x-product of T and U is the set of all trees obtained from any $t \in T$ by replacing each leaf labeled with x with a tree in U. Note that different leaves labeled with x may be substituted by different trees in U. Furthermore, the *x-quotient* of T by U is

$$T /_x U := \{t \in T_\Sigma(X) \mid (\{t\} \bullet_x U) \cap T \neq \emptyset\} \ .$$

Hence, $T /_x U$ is the set of all trees that can be transformed into a tree in T by replacing every leaf labeled with x with a tree in U. Now, the *x-iteration* of $T (\subseteq T_\Sigma(X))$ is the ΣX-tree language $T^{*x} := \bigcup_{j \geq 0} T^{j,x}$, where $T^{0,x} := \{x\}$ and $T^{j,x} := (T \bullet_x T^{j-1,x}) \cup T^{j-1,x}$ for all $j \geq 1$. Informally, T^{*x} contains x and all trees obtained by replacing in some $t \in T$ every occurrence of x with some tree that is already in T^{*x}. Let us illustrate the previous notions by an example.

Example 4.3.1. Let $\Sigma = \{f/3, g/2, e/0\}$ and $X = \{x\}$, and consider the tree $t = f(g(\xi_2), x, f(e, x, \xi_1)) \in T_\Sigma(X \cup \Xi_2)$ and two arbitrary trees $t_1, t_2 \in T_\Sigma(X)$. Then

$$\begin{aligned} t[t_1, t_2] &= f(g(t_2), x, f(e, x, t_1)) \ , \\ t[x \leftarrow (t_1, t_2)] &= f(g(\xi_2), t_1, f(e, t_2, \xi_1)) \ , \text{ and} \\ f(\{e, t_1\}, \{\xi_2\}, \{t_2\}) &= \{f(e, \xi_2, t_2), f(t_1, \xi_2, t_2)\} \ . \end{aligned}$$

Now let $T = \{x, f(x, e, f(e, x, e))\}$ and $U = \{t_1, t_2\}$. Obviously, $T \bullet_x U$ equals

$$U \cup \{f(t_1, e, f(e, t_1, e)), f(t_1, e, f(e, t_2, e)), f(t_2, e, f(e, t_1, x)), f(t_2, e, f(e, t_2, e))\} \ ,$$

and if $t_1 = x$ and $t_2 = e$, then

$$T /_x U = \{f(x,x,f(x,x,x)), f(x,x,f(x,x,e)), f(x,x,f(e,x,x)), f(x,x,f(e,x,e))\} \\ \cup \{f(x,e,f(x,x,x)), f(x,e,f(x,x,e)), f(x,e,f(e,x,x))\} \cup T \ . \qquad \square$$

For any $T \subseteq T_\Sigma(X)$, we call $\mathrm{yd}(T) := \{\mathrm{yd}(t) \mid t \in T\}$ $(\subseteq X^*)$ the *yield language* of T, and let $\mathrm{yd}^{-1}(L) := \{t \in T_\Sigma(X) \mid \mathrm{yd}(t) \in L\}$ for any $L \subseteq X^*$. Moreover, any ΣX-context c defines a unary operation $c\colon T_\Sigma(X) \to T_\Sigma(X)$, $t \mapsto c(t)$, which generates two tree language operations. More precisely, for any $T \subseteq T_\Sigma(X)$,

$$c(T) := \{c(t) \mid t \in T\} \quad \text{and} \quad c^{-1}(T) := \{t \in T_\Sigma(X) \mid c(t) \in T\} \ .$$

To work with (infinite sets of) trees, one needs methods to specify tree languages. Again, as in the string case, grammars and recognizers are the most common ones, but rather than working with strings, they generate and accept trees. We proceed by presenting the most well-known classes of tree languages, how to specify them and the relation between the various classes introduced.

4.4 Recognizable tree languages

As shown already by Thatcher and Wright (1968), Doner (1970) and Magidor and Moran (1969), for example, the recognizable (or regular) tree languages are obtained in numerous ways (cf. Gécseg and Steinby, 1984, Gécseg and Steinby, 1997, Engelfriet, 1975c, Comon et al., 2007, for expositions and further references). Next, we exhibit the recognizers and the generating devices defining recognizable tree languages, as well as some of their properties. Furthermore, we present subclasses with practical applications and several connections between tree languages and string languages.

4.4.1 Tree recognizers

Since trees are hierarchical structures, they can be processed either starting from the root and moving towards the leaves (*top-down*), or starting from the leaves and going up towards the root (*bottom-up*). In both cases the control may be nondeterministic or deterministic. We shall see that one type of recognizer accepts a smaller class than recognizable tree languages, the other three being equivalent. We start with nondeterministic top-down recognizers first studied by Magidor and Moran (1969).

Definition 4.4.1. A *nondeterministic top-down* (ndT) ΣX*-recognizer* is a system $TR = (Q, \Sigma, X, P, I)$, specified as follows.

(1) Q is a unary ranked alphabet of *states* such that $Q \cap (\Sigma \cup X) = \emptyset$.

(2) Σ and X are the *input alphabets*.

(3) $I \subseteq Q$ is the set of *initial states*.

(4) P is a finite set of *rules*, each of one of the following two types:

(NDT1) $q(d) \to d$, where $d \in X \cup \Sigma_0$ and $q \in Q$;

(NDT2) $q(f(\xi_1,\ldots,\xi_m)) \to f(q_1(\xi_1),\ldots,q_m(\xi_m))$, where $m \geq 1$, $f \in \Sigma_m$ and q, $q_1,\ldots,q_m \in Q$. □

Since we defined trees as terms, we give the semantics of an ndT as a term rewriting system. So, for any $s,t \in T_{\Sigma\cup Q}(X)$, $s \Rightarrow_{TR} t$ means that t is obtained from s by replacing an occurrence of

(1) a subtree $q(d)$ of s by d, where $q(d) \to d$ is a rule of type (NDT1) in P, or

(2) a subtree $q(f(t_1,\ldots,t_m))$ of s by the tree $f(q_1(t_1),\ldots,q_m(t_m))$ by using a type (NDT2) rule $q(f(\xi_1,\ldots,\xi_m)) \to f(q_1(\xi_1),\ldots,q_m(\xi_m))$ appearing in P.

Then, the *tree language accepted* by TR is the set

$$T(TR) := \{t \in T_\Sigma(X) \mid q(t) \Rightarrow_{TR}^* t \text{ for some } q \in I\} \ .$$

Thus, TR accepts a ΣX-tree t if and only if it is possible to choose an initial state in which TR starts at the root of t and then successively apply transition rules for each node of t in such a way every leaf is reached in a state q that agrees with the label d of the leaf, i.e., P contains the rule $q(d) \to d$. A ΣX-tree language R is called *recognizable*, or *regular*, if $R = T(TR)$ for some ndT ΣX-recognizer TR. Let Rec denote the family of all recognizable tree languages. Hence, by our general convention, $\mathrm{Rec}_\Sigma(X)$, and Rec_Σ denote the sets of all recognizable ΣX-tree languages, and all recognizable Σ-tree languages, respectively.

An *ndT Σ-recognizer* $TR = (Q,\Sigma,P,I)$ of a Σ-tree language $T \subseteq T_\Sigma$ is obtained by an obvious modification of Definition 4.4.1, and let $\mathrm{Rec}^{\mathrm{vf}}$ be the family of recognizable tree languages without a leaf alphabet.

Example 4.4.2. The system $TR = (\{q_f,q_g,q_x\},\{f/2,g/1\},\{x\},P,\{q_f\})$, where P is consisting of the rules

$$\begin{aligned} q_f(f(\xi_1,\xi_2)) &\to f(q_g(\xi_1),q_g(\xi_2)) \\ q_g(g(\xi_1)) &\to g(q_g(\xi_1)) \\ q_g(g(\xi_1)) &\to g(q_x(\xi_1)) \\ q_x(x) &\to x \ , \end{aligned}$$

is an ndT recognizer, and $T(TR) = \{f(g^n(x),g^m(x)) \mid n,m \geq 1)$. A sample computation in TR is

$$\begin{aligned} q_f(f(g(x),g(g(x)))) &\Rightarrow_{TR} f(q_g(g(x)),q_g(g(g(x)))) \Rightarrow_{TR}^2 f(g(q_x(x)),g(q_g(g(x)))) \Rightarrow_{TR} \\ &\Rightarrow_{TR} f(g(q_x(x)),g(g(q_x(x)))) \Rightarrow_{TR}^2 f(g(x),g(g(x))) \ , \end{aligned}$$

which shows that the tree $f(g(x),g(g(x)))$ is accepted by the machine. □

If a nondeterministic top-down tree recognizer has exactly one initial state and, in the accepting process, at most one rule can be applied at each node of the tree, then it becomes a deterministic top-down recognizer. In this case, the recognition power is weaker because the machine has to make the decision of acceptance separately at each leaf without any information about the tree outside the path leading from the root to that leaf.

Definition 4.4.3. A *deterministic top-down* (dT) ΣX*-recognizer* is an ndT ΣX-recognizer $TR = (Q, \Sigma, X, P, I)$ with exactly one initial state, at most one rule of type (NDT1) for each pair $(d, q) \in (X \cup \Sigma_0) \times Q$, and at most one rule of type (NDT2) for any $m \geq 1$, $f \in \Sigma_m$ and $q \in Q$. □

If $I = \{q_0\}$, we write $TR = (Q, \Sigma, X, P, q_0)$. A ΣX-tree language is *deterministic recognizable* (dT-recognizable) if it is recognized by a dT ΣX-recognizer. The family of all dT-recognizable tree languages is denoted by DRec. An example is given next.

Example 4.4.4. If $\Sigma = \{f/2, g/1\}$ and $X = \{x\}$, then $TR = (\{q_0, q_1\}, \Sigma, X, P, q_0)$ with P consisting of the following rules

$$\begin{aligned} q_0(f(\xi_1, \xi_2)) &\to f(q_1(\xi_1), q_1(\xi_2)) \\ q_0(g(\xi_1)) &\to g(q_1(\xi_1)) \\ q_1(f(\xi_1, \xi_2)) &\to f(q_0(\xi_1), q_0(\xi_2)) \\ q_1(g(\xi_1)) &\to g(q_0(\xi_1)) \\ q_0(x) &\to x \end{aligned}$$

is a deterministic top-down ΣX-recognizer accepting the language of all ΣX-trees in which each path is of even length. □

We also note the following (see Magidor and Moran, 1969, Gécseg and Steinby, 1984, Gécseg and Steinby, 1997, Engelfriet, 1975c, for example).

Theorem 4.4.5. DRec $\subset$ Rec.

Proof. It is clear from the definitions that any dT ΣX-recognizer is also an ndT ΣX-recognizer. Let $\Sigma = \{f/2\}$ and $X = \{x, y\}$. Obviously, the finite set $\{f(x, y), f(y, x)\}$ is in $\mathrm{Rec}_\Sigma(X)$. If there is a dT ΣX-recognizer that accepts the trees $f(x, y)$ and $f(y, x)$, then it must also accept the trees $f(x, x)$ and $f(y, y)$. Consequently, we get that $\{f(x, y), f(y, x)\} \notin \mathrm{DRec}_\Sigma(X)$, and hence DRec $\subset$ Rec. □

Let us now consider the bottom-up generalization of the finite automaton. The precise definition follows.

Definition 4.4.6. A *nondeterministic bottom-up* (ndB) ΣX*-recognizer* is a system $BR = (Q, \Sigma, X, P, F)$, where

(1) Q, Σ and X have the same meaning as in Definition 4.4.1,

(2) $F \subseteq Q$ is the set of *final states*, and

(3) P is a finite set of *rules*, each of one of the following two types:

(NDB1) $d \to q(d)$ with $d \in X \cup \Sigma_0$ and $q \in Q$;

(NDB2) $f(q_1(\xi_1), \ldots, q_m(\xi_m)) \to q(f(\xi_1, \ldots, \xi_m))$ with $m \geq 1$, $f \in \Sigma_m$ and q, $q_1, \ldots, q_m \in Q$. □

The next-configuration relation $\Rightarrow_{BR}$ is defined for any ndB BR as a term rewriting system as follows. For any $s, t \in T_{\Sigma \cup Q}(X)$, $s \Rightarrow_{BR} t$ means that t is obtained from s by replacing an occurrence of

(1) d in s by the tree $q(d)$, where $q(d) \to d$ is a rule of type (NDB1) in P, or

(2) a subtree $f(q_1(t_1), \ldots, q_m(t_m))$ of s by the tree $q(f(t_1, \ldots, t_m))$ by using a type (NDB2) rule $f(q_1(\xi_1), \ldots, q_m(\xi_m)) \to q(f(\xi_1, \ldots, \xi_m))$ appearing in P.

Thus, the *tree language accepted by BR* is the set

$$T(BR) := \{t \in T_\Sigma(X) \mid t \Rightarrow_{BR}^* q(t) \text{ for some } q \in F\} \ .$$

Moreover, BR is *deterministic* if there are no two transition rules in P with the same left-hand side. Next we shall give an example.

Example 4.4.7. The device $BR = (\{q_f, q_g, q_x\}, \{f/2, g/1\}, \{x\}, P, \{q_f\})$, where P consists of the rules

$$\begin{aligned} f(q_g(\xi_1), q_g(\xi_2)) &\to q_f(f(\xi_1, \xi_2)) \\ g(q_g(\xi_1)) &\to q_g(g(\xi_1) \\ g(q_x(\xi_1)) &\to q_g(g(\xi_1) \\ x &\to q_x(x) \end{aligned}$$

is a bottom-up tree recognizer recognizing the tree language $\{f(g^n(x), g^m(x)) \mid n, m \geq 1)$. The computation of $f(g(x), g(g(x)))$ in TR is

$$\begin{aligned} f(g(x), g(g(x))) \Rightarrow_{BR}^2 f(g(q_x((x))), g(g(q_x(x)))) \Rightarrow_{BR}^2 f(q_g(g(x)), g(q_g(g(x)))) \Rightarrow_{TR} \\ \Rightarrow_{BR} f(q_g(g(x)), q_g(g(g(x)))) \Rightarrow_{TR} q_f(f(g(x), g(g(x)))) \ . \end{aligned}$$

Note that actually BR is deterministic. □

Obviously, any deterministic ΣX-bottom-up tree recognizer is by definition an ndB that contains exactly one rule for each possible left-hand side. On the other hand, any ndB ΣX-recognizer can be made deterministic by the usual subset construction for finite automata (see Engelfriet (1975c, Theorem 3.8) or Gécseg and Steinby (1984, Theorem 2.6), for example). Thus, deterministic and nondeterministic bottom-up tree recognizers define the same class of tree languages. Furthermore, any ndB ΣX-recognizer defined as a term rewriting system becomes an equivalent ndT ΣX-recognizer of the kind introduced in Definition 4.4.1 when all rules are reversed and final states are turned into initial states (see Example 4.4.7). Moreover, the converse transformation yields an equivalent ndB ΣX-recognizer for any given ndT ΣX-recognizer (Engelfriet, 1975c, Theorem 3.17). Hence, ndB tree recognizers recognize exactly the recognizable tree languages.

In what follows we also speak generally about tree recognizers without specifying the alphabets.

4.4.2 Regular tree grammars

Another way to define the recognizable tree languages is by regular tree grammars, first studied by Brainerd (1969) (in a more general form). These generating devices are the natural generalization from strings to trees of regular grammars of the Chomsky hierarchy. Formally, they are defined as follows.

Definition 4.4.8. A *regular ΣX-tree grammar* (RTG) is a device $RT = (N, \Sigma, X, P, S)$ specified as follows.

(1) N is a finite non-empty set of *nonterminal symbols* such that $N \cap (\Sigma \cup X) = \emptyset$.

(2) Σ is a ranked alphabet and X is a leaf alphabet that together form the *terminal alphabet* of RT.

(3) P is a finite set of *productions* of the form $A \to r$, where $A \in N$ and $r \in T_\Sigma(X \cup N)$.

(4) $S \in N$ is the distinguished *start symbol*. □

For any $s, t \in T_\Sigma(X \cup N)$, $s \Rightarrow_{RT} t$ means that there exist a $(\Sigma \cup N)X$-context c and a production $A \to r$ in P such that $s = c(A)$ and $t = c(r)$ (i.e., t can be obtained from s by replacing one occurrence of A by r). The derivation relation $\Rightarrow^*_{RT}$ and the n-step derivation relation $\Rightarrow^n_{RT}$ are defined as usual (see Section 2.3.2). The *ΣX-tree language generated* by RT is the set

$$T(RT) := \{t \in T_\Sigma(X) \mid S \Rightarrow^*_{RT} t\} \ .$$

Note that any regular ΣX-tree grammar may be viewed as a CFG with the terminal alphabet $\Sigma \cup X \cup Z$, where Z consists of the parentheses (and) and the comma. Thus, the tree languages generated by RTGs are special CFLs when trees are treated as strings. An example is given next.

Example 4.4.9. Let $\Sigma = \{f/3, g/2\}$ and $X = \{x\}$. The system

$$RT = (\{S, B\}, \Sigma, X, \{S \to f(x, S, B), S \to g(x, B), B \to y\}, S)$$

is an RTG generating the ΣX-tree language

$$T(RT) = \{t \in T_\Sigma(X) \mid t = f(x, \xi, y)^n(g(x, y)), n \in \mathbb{N}\} \ .$$

A sample derivation in RT is

$$\begin{aligned} S &\Rightarrow_{RT} f(x, S, B) \Rightarrow_{RT} f(x, f(x, S, B), B) \Rightarrow_{RT} f(x, f(x, g(x, B), B), B) \\ &\Rightarrow^3_{RT} f(x, f(x, g(x, y), y), y) \ . \end{aligned}$$

Note that the yield language of $T(RT)$ is the CFL $\{x^n y^n \mid n \geq 1\}$.

A more complicated example of an RTG that models linguistic phenomena can be found in Graehl et al. (2008, Figure 4), for example. Also, May (2010, Section 2.2) shall be consulted. □

We recall the following (cf. Gécseg and Steinby, 1984, Lemma II.3.4).

Theorem 4.4.10. *Every regular ΣX-tree grammar is effectively equivalent to a regular ΣX-tree grammar $RT = (N, \Sigma, X, P, S)$ in which each production is of the form*

(1) *$A \to d$, where $A \in N$ and $d \in \Sigma_0 \cup X$, or of the form*

(2) *$A \to f(A_1, \ldots, A_m)$, where $m > 0$, $f \in \Sigma_m$ and $A, A_1, \ldots, A_m \in N$.*

Such a tree grammar is said to be in normal form.

Example 4.4.11. Let $\Sigma = \{f/2, g/1\}$ and $X = \{x, y\}$. Then $RT = (\{S, A\}, \Sigma, X, P, S)$ with P consisting of the productions

$$S \to f(A, A) \mid g(A), \text{ and}$$
$$A \to f(S, S) \mid g(S) \mid x \mid y$$

is a ΣX-tree grammar in normal form generating all ΣX-trees of odd height.

On the other hand, the RTG RT of Example 4.4.9 is equivalent to the RTG in normal form $(\{S, A, B\}, \Sigma, X, \{S \to f(A, S, B), S \to g(A, B), A \to x, B \to y\}, S)$. □

Any regular ΣX-tree grammar in normal form can easily be converted to an equivalent ndT ΣX-tree recognizer by turning nonterminals into states and taking as rules the productions, and conversely. Hence it is clear that the regular tree grammars generate exactly the recognizable tree languages (see Gécseg and Steinby (1984, Theorem II.3.6) or Gécseg and Steinby (1997, Proposition 6.2), for example).

4.4.2.1 Tree substitution grammars

We now consider a restricted type of regular tree grammar with applications in linguistics, namely tree-substitution grammars (Schabes, 1990, Frank, 2000, Eisner, 2003, Shieber, 2004), by giving a new formal definition that will be further used and explained in Section 5.3.1. In the formulation of Shieber (2004), a tree-substitution grammar is essentially a set P of trees in $T_\Sigma(N)$, where Σ is the ranked alphabet of terminal symbols and $N := \{f_\downarrow \mid f \in \Sigma \setminus \Sigma_0\}$. Each derivation begins with a tree in P whose root is labeled with the symbol in Σ that has been chosen as the start symbol. The idea is that if a leaf is labeled with a symbol $f_\downarrow$ in N, then we may substitute for that leaf any tree in P in which the root is labeled by f. Derivations are defined via derivation trees, but we may define them equivalently in the usual way by regarding the symbols $f_\downarrow \in N$ as nonterminals and then stipulating that if $f_\downarrow$ is the left-hand side of a production, then the right-hand side of the production is a tree $r \in P$ such that $\mathrm{root}(r) = f$. The possibility of having multiple copies of trees in P and the ordering of the N-labeled leaves in the tree in P postulated by Shieber (2004) do not have any effect on the generative power, and their intended uses will be achieved otherwise when we define the synchronous version of tree-substitution grammars. On the other hand, the set of productions should obviously be finite, and hence we arrive at the following definition.

Definition 4.4.12. A *tree-substitution grammar* (TSG) is a system $TS = (N, \Sigma, X, P, S)$ specified as follows.

(1) Σ and X are the *terminal alphabets*.

(2) $N := \{f_\downarrow \mid f \in \Sigma \setminus \Sigma_0\}$ is the set of *nonterminal symbols* such that $N \cap (\Sigma \cup X) = \emptyset$.

(3) P is a finite set of *productions* of the form $f_\downarrow \to r$, where $f_\downarrow \in N$, $r \in T_\Sigma(X \cup N)$ and $\mathrm{root}(r) = f$.

(4) $S \in N$ is the *start symbol* (and hence $S = f_\downarrow$ for some $f \in \Sigma \setminus \Sigma_0$). □

The one-step derivation relation $\Rightarrow_{TS}$ is defined as usual: if $s, t \in T_\Sigma(X \cup N)$, then $s \Rightarrow_{TS} t$ if and only if there is a context $c \in C_\Sigma(X \cup N)$ and a rule $f_\downarrow \to r$ in P such

that $s = c(f_\downarrow)$ and $t = c(r)$. Then, the ΣX-tree language *generated* by TS is the set $T(TS) := \{t \in T_\Sigma(X) \mid S \Rightarrow_{TS}^* t\}$. Furthermore, let $T[TSG]$ denote the class of all the tree languages generated by TSGs.

As Frank (2000) observed, TSGs are adequate systems to produce appropriate structural descriptions for clausal complementation and to generate sentences containing adjunction structures such as adverbial modifiers and relative clauses. Next, we give an example of such a tree grammar.

Example 4.4.13. The system

$$TS = (\{S_\downarrow\}, \{S/2\}, \{x, y\}, \{S_\downarrow \to S(x, y), S_\downarrow \to S(x, S(S_\downarrow, y))\}, S_\downarrow)$$

is a TSG. A sample derivation in TS is

$$\begin{aligned} S_\downarrow &\Rightarrow_{TS} S(x, S(S_\downarrow, y)) \Rightarrow_{TS} S(x, S(S(x, S(S_\downarrow, y)), y)) \\ &\Rightarrow_{TS} S(x, S(S(x, S(S(x, y), y)), y)) \ , \end{aligned}$$

and TS generates the tree language $T(TS) = \{S(x, S(\xi, y))^n(S(x, y)) \mid n \in \mathbb{N}\}$. □

It is immediately clear from the definition that every TSG is a special regular tree grammar and hence that $T[TSG] \subseteq$ Rec. However, it is also obvious that not every recognizable tree language can be generated by a TSG. First of all, the root of any tree in the tree language generated by a TSG must be labeled by the symbol that corresponds to the start symbol. Secondly, the number of nonterminals is limited by the size of the ranked alphabet. Moreover, the dT-recognizable tree language $\{x\}$ cannot be generated by any TSG. On the other hand, the TSG $(\{f_\downarrow\}, \{f/2\}, \{x, y\}, \{f_\downarrow \to f(x, y), f_\downarrow \to f(y, x)\}, f_\downarrow)$ generates the tree language $\{f(x, y), f(y, x)\}$, which is not dT-recognizable by Theorem 4.4.5. Hence, we have.

Theorem 4.4.14. $T[TSG] \subset$ Rec *and* $T[TSG] \parallel$ DRec.

On the other hand, we may note the following fact (cf. Schabes, 1990).

Proposition 4.4.15. *Every context-free language is the yield of a tree language generated by a tree-substitution grammar.*

Proof. If $L \subseteq X^*$ is a CFL, it is generated by a CFG $CF = (N, X, P, S)$ in CNF (cf. Theorem 2.4.8). Assuming first that $L \subseteq X^+$, we define the TSG $TS = (N', \Sigma, X, P', S')$, where $\Sigma := \{e/0\} \cup \{A/2 \mid A \in N\}$, $N' := \{A_\downarrow \mid A \in N\}$, $S' := S_\downarrow$, and

$$P' := \{A_\downarrow \to A(B_\downarrow, C_\downarrow) \mid A \to BC \in P\} \cup \{A_\downarrow \to A(x, e) \mid A \to x \in P\} \ .$$

Clearly, TS generates exactly the usual derivation trees of CF (cf. Sudkamp, 1997, for example) in which inner nodes are labeled with nonterminal symbols (now binary symbols in Σ), but here any leaf labeled with a terminal symbol $x \in X$ is replaced with a subtree $A(x, e)$. Hence, $\mathrm{yd}(T(TS)) = L(CF) = L$. If $\varepsilon \in L$, the construction is modified by adding the production $S_\downarrow \to S(e, e)$ to P'. Note that S does not appear on the right-hand side of any production in P because CF is in CNF. □

The closure properties of $T[TSG]$ are discussed by Maletti (2014). Thus, we find out that this tree language class is not closed under union, intersection, complement and alphabetic tree homomorphism.

4.4.3 Properties of recognizable tree languages

First of all, there is a tool that generalizes the Pumping Lemma of regular languages (Hopcroft et al., 2001, Theorem 4.1) from strings to trees. It may be used to show that certain tree languages are not recognizable (see Engelfriet (1975c, Theorem 3.71), Gécseg and Steinby (1984, Lemma II.10.1) or Gécseg and Steinby (1997, Proposition 5.2), for example.).

Theorem 4.4.16 (Pumping Lemma)**.** *For any $R \in \mathrm{Rec}_\Sigma(X)$, there is $n \in \mathbb{N}_+$ such that if $r \in R$ and $\mathrm{hg}(r) \geq n$, then for some $s \in T_\Sigma(X)$ and $c, c' \in C_\Sigma(X)$, $r = c'(c(s))$, $\mathrm{hg}(c) \geq 1$, and $c'(c^k(s)) \in R$ for all $k \in \mathbb{N}$.*

Now, we mention some relevant closure properties and decidability results of recognizable tree languages (cf. Gécseg and Steinby (1984, Section II.4), and Engelfriet (1975c, Theorems 3.74 and 3.75), for example).

Theorem 4.4.17. *The following hold.*

(i) Rec *is closed under union, intersection, and complement.*

(ii) Rec *is not closed under arbitrary tree homomorphisms.*

(iii) Rec *is closed under linear tree homomorphism and arbitrary inverse tree homomorphism.*

(iv) Rec *is closed under f-concatenation, x-product, x-quotient and x-iteration.*

(v) *If $c \in C_\Sigma(X)$ and $T \in \mathrm{Rec}_\Sigma(X)$, then $c(T)$ and $c^{-1}(T)$ are also in $\mathrm{Rec}_\Sigma(X)$.*

(vi) *The emptiness, the finiteness, the membership, the inclusion and the equivalence are decidable for recognizable tree languages.*

4.5 Local tree languages

Another important class of recognizable tree languages is defined as follows.

Definition 4.5.1. For any $D \subseteq \Sigma \cup X$ and $E \subseteq \mathrm{fork}(\Sigma, X)$, let

$$L(D, E) := \{t \in T_\Sigma(X) \mid \mathrm{root}(t) \in D, \mathrm{fork}(t) \subseteq E\} \ .$$

A ΣX-tree language T is *local* if $T = L(D, E)$ for some D and E. □

Let Loc be the family of local tree languages. Note that, for any given Σ and X, $\mathrm{Loc}_\Sigma(X)$ is a finite set of tree languages. An example is provided next.

Example 4.5.2. Let $\Sigma = \{f/2, g/1, e/0\}$, $X = \{x\}$, $E = \{f(g, e), g(f), g(x)\}$ and $D = \{f, x\}$. Then,

$$L(D, E) = \{x\} \cup \{f(g(\xi), e)^n(g(x)) \mid n \geq 1\} \ .$$

A tree in $L(D, E)$ is $f(g(f(g(x), e)), e)$. □

We also observe the following.

Theorem 4.5.3. Loc $\parallel T[TSG]$, Loc $\subset$ Rec *and* Loc $\parallel$ DRec.

Proof. It is easy to construct a ΣX-recognizer for any given $L(D,E)$ with $D \subseteq \Sigma \cup X$ and $E \subseteq \mathrm{fork}(\Sigma, X)$ (Gécseg and Steinby, 1984, Theorem II.9.4), and consequently $\mathrm{Loc} \subseteq \mathrm{Rec}$. Moreover, it is clear that there exist one-tree tree languages that are dT-recognizable but not local, and therefore the inclusion is strict.

On the other hand, $\{f(x,y), f(y,x)\}$ is local ($D = \{f\}$ and $E = \{f(x,y), f(y,x)\}$) but not dT-recognizable (see the proof of Theorem 4.4.5), and hence $\mathrm{Loc} \parallel \mathrm{DRec}$.

Finally, it is obvious that the local tree language of Example 4.5.2 cannot be generated by a TSG (the label of the root of any tree is either x or g and hence cannot be labeled by the start symbol). Moreover, the tree language $g(g(x))$, which is not local, is generated by the TSG $TS = (\{g_\downarrow\}, \{g/1\}, \{x\}, \{g_\downarrow \to g(g(x))\})$. So, $\mathrm{Loc} \parallel T[TSG]$. □

On the other hand, we have.

Remark 4.5.4. Any dT-recognizable local ΣX-tree language with one root symbol is in $T[TSG]$. □

It is well known that every recognizable tree language is obtained by an alphabetic tree homomorphism from a local tree language (cf. Doner, 1970, Thatcher, 1967, for example). However, if we use the proof presented by Gécseg and Steinby (1984, Section II.9), this fact can be given in a slightly stronger form.

Proposition 4.5.5. *For every recognizable ΣX-tree language T, we may define a ranked alphabet Ω, a leaf alphabet Y, a local dT-recognizable ΩY-tree language U and an alphabetic tree homomorphism $\varphi\colon T_\Omega(Y) \to T_\Sigma(X)$ such that $T = U\varphi$. Moreover, if $T \subseteq T_\Sigma$, then U can be chosen to be in $\mathrm{Loc}_\Omega \cap \mathrm{DRec}_\Omega$.*

Proof. Assume $RT = (N, \Sigma, X, P, S)$ is a regular ΣX-tree grammar in normal form that generates T. Let $Y := \{[A \to x] \mid A \to x \in P, x \in X\}$, and let Ω be the ranked alphabet such that $\Omega_0 := \{[A \to e] \mid A \to e \in P, e \in \Sigma_0\}$ and for each $m \geq 1$,

$$\Omega_m := \{[A \to f(A_1, \ldots, A_m)] \mid A \to f(A_1, \ldots, A_m) \in P\} \ .$$

Now, let $U := L(D,E)$ be the local ΩY-tree language, where $D := \{[S \to r] \mid S \to r \in P\}$ and

$$E := \{[A \to f(A_1, \ldots, A_m)]([A_1 \to r_1], \ldots, [A_m \to r_m]) \mid m \geq 1, \\ A \to f(A_1, \ldots, A_m), A_1 \to r_1, \ldots, A_m \to r_m \in P\} \ .$$

Next, let $\varphi : T_\Omega(Y) \to T_\Sigma(X)$ be the alphabetic tree homomorphism defined by the mappings $\varphi_Y\colon Y \to T_\Sigma(X), [A \to x] \mapsto x$, $\varphi_0\colon \Omega_0 \to T_\Sigma(X), [A \to e] \mapsto e$, and

$$\varphi_m : \Omega_m \to T_\Sigma(X \cup \Xi_m), [A \to f(A_1, \ldots, A_m)] \mapsto f(\xi_1, \ldots, \xi_m) \ ,$$

for every $m \geq 1$. Then $T = U\varphi$ (Gécseg and Steinby, 1984, Theorem 9.5).

To show that U is also dT-recognizable, we define a dT recognizer $TR = (Q, \Omega, Y, P', S)$ as follows. Let $Q = Q_1 := N \cup \{\star\}$, where $\star$ is a new symbol. The set P' of transition rules is defined by the following clauses:

(1) For any $q \in Q$ and $[A \to d] \in Y \cup \Omega_0$, the rule $q([A \to d]) \to [A \to d]$ is in P' if and only if $q = A$.

(2) Consider now any pair $q \in Q$ and $[A \to f(A_1 \dots A_m)] \in \Omega_m$, where $m \geq 1$. If $q \neq A$, then the rule for this pair is

$$q([A \to f(A_1 \dots A_m)](\xi_1, \dots, \xi_m)) \to [A \to f(A_1 \dots A_m)](\star(\xi_1), \dots, \star(\xi_m)) \ .$$

If $q = A$, then the rule is

$$q([A \to f(A_1 \dots A_m)](\xi_1, \dots, \xi_m)) \to [A \to f(A_1 \dots A_m)](A_1(\xi_1), \dots, A_m(\xi_m)) \ .$$

It is not hard to see that $T(RT) = L(D, E)$, and hence $U \in \mathrm{Loc}_\Omega(Y) \cap \mathrm{DRec}_\Omega(Y)$.

Finally, we note if X is empty, then so is Y. □

4.6 Grammars and production trees

Each ΩY-tree s in the set U defined in the proof of Proposition 4.5.5 represents a derivation in RT of the ΣX-tree $s\varphi \in T$ by displaying the structure of the derivation as well as the productions used in it. We call these trees *production trees* of RT and denote their set U by $P(RT)$.

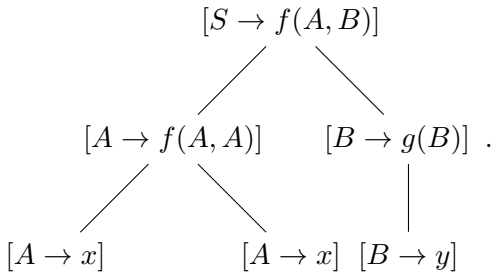
Example 4.6.1. Let us consider the regular ΣX-tree grammar $RT = (N, \Sigma, X, P, S)$, where $N = \{S, A, B\}$, $\Sigma = \{f/2, g/1\}$, $X = \{x, y\}$, and

$$P \ = \ \{S \to f(A, B), A \to f(A, A), A \to x, B \to g(B), B \to y\} \ .$$

The production tree of the derivation

$$S \Rightarrow_{RT} f(A, B) \Rightarrow^2_{RT} f(f(A, A), g(B)) \Rightarrow^3_{RT} f((x, x), g(y)) \ ,$$

is



□

Similar production trees will be used also for representing derivations in other kinds of generating devices. In the derivation trees of a CFG the inner nodes are labeled by nonterminals, but the number of branches going out of a node depends on the length of the right-hand side of the corresponding production rather than the symbol itself (see Section 2.4 and Aho and Ullman, 1972, Section 2.4.1, for example). This contradicts the uniqueness of the rank of each symbol in a ranked alphabet, so we consider production trees in which inner nodes are labeled by productions instead of nonterminals.

Much of the importance of tree automata in language theory derives from the following well-known fact (see Mezei and Wright, 1967, Thatcher, 1967, Doner, 1970, Engelfriet, 1975c, Gécseg and Steinby, 1984, Gécseg and Steinby, 1997, for example).

Theorem 4.6.2. *A language is context-free if and only if it is the yield language of a recognizable tree language.*

This result can be made more precise: the yield-language of any recognizable tree language is context-free, but all CFLs are obtained already from a proper subclass of Rec. Indeed, by using production trees similar to those defined above for regular tree grammars, the class of tree languages needed can be narrowed down to $\mathrm{Loc} \cap \mathrm{DRec}$. Following Gécseg and Steinby (1984), we define the production trees of a CFG $CF = (N, X, P, S)$ as follows.

Let Σ^{CF} be the ranked alphabet such that

$$\Sigma^{CF}_m := \{[A \to \alpha] \mid A \to \alpha \in P,\ |\alpha| = m\}$$

for each $m \geq 0$. The sets $P(CF, d)$ of $\Sigma^{CF}X$-trees, where $d \in N \cup X$, are defined inductively as follows:

(1) $P(CF, x) := \{x\}$ for $x \in X$;

(2) for each $A \in N$ such that $A \to \varepsilon \in P$, let $[A \to \varepsilon] \in P(CF, A)$;

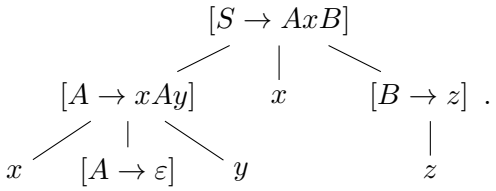
(3) if $A \to d_1 \dots d_m \in P$, where $m \geq 1$, $d_1, \dots, d_m \in N \cup X$ and $t_1 \in P(CF, d_1), \dots, t_m \in P(CF, d_m)$, then $[A \to d_1 \dots d_m](t_1, \dots, t_m) \in P(CF, A)$.

The set of *production trees* of CF is the Σ^{CF}-tree language $P(CF) := P(CF, S)$. Each production tree $t \in P(CF)$ represents a (unique leftmost) derivation of the string $\mathrm{yd}(t) \in L(CF)$. On the other hand, for each $v \in L(CF)$ and each leftmost derivation of v, there is a production tree $t \in P(CF)$ such that $v = \mathrm{yd}(t)$.

Example 4.6.3. Let $CF = (\{S, A, B\}, \{x, y, z\}, P, S)$ be the CFG with the production set $P = \{S \to AxB,\ A \to xAy,\ A \to \varepsilon,\ B \to z\}$. The leftmost derivation

$$S \Rightarrow_{CF} AxB \Rightarrow_{CF} xAyxB \Rightarrow_{CF} xyxB \Rightarrow_{CF} xyxz$$

is represented by the production tree



The yield of this tree is $xyxz$ – as it should. □

Similarly as for the production trees of a RTG (Proposition 4.5.5), we can show that $P(CF)$ is both local and dT-recognizable. The above observations can be summed up as follows (cf. Gécseg and Steinby, 1984, Section II.2).

Proposition 4.6.4. *For any CFG $CF = (N, X, P, S)$, the set $P(CF)$ of production trees is a local and dT-recognizable $\Sigma^{CF}X$-tree language such that* $\mathrm{yd}(P(CF)) = L(CF)$.

4.7 Context-free tree languages

In Section 4.4.2, regular grammars were generalized from strings to trees by allowing, in the right-hand sides of each production, trees in which any nonterminal may occur as a constant labeling the leaves. If we permit nonterminals to label any node in the tree of the right-hand side of any production, CFGs get a natural counterpart in tree language theory formally defined as follows (Rounds, 1970b, 1969).

Definition 4.7.1. A *context-free tree grammar* (CFTG) is a system $CT = (N, \Sigma, X, P, S)$ specified as follows.

(1) N is a ranked alphabet of *nonterminals* such that $N \cap (X \cup \Sigma) = \emptyset$.

(2) Σ and X are a ranked alphabet and a leaf alphabet, respectively, of *terminal symbols*.

(3) P is a finite set of *productions* of the form $f(\xi_1, \ldots, \xi_m) \to r$, where $m \in \mathbb{N}$, $f \in N_m$ and $r \in T_{N \cup \Sigma}(X \cup \Xi_m)$.

(4) $S \in N_0$ is the *start symbol*. □

The direct derivation relation $\Rightarrow_{CT}$, the derivation relation $\Rightarrow^*_{CT}$ and the n-step derivation relation $\Rightarrow^n_{CT}$ are defined as for a term rewriting system with the productions as rewrite rules, similarly as it was done for RTGs in Section 4.4.2. The ΣX-*tree language generated by* CT is the set $T(CT) := \{t \in T_\Sigma(X) \mid S \Rightarrow^*_{CT} t\}$. A tree language is *context-free* if it is generated by a CFTG, and let $CFTL$ denote the class of all context-free tree languages (CFTL). Gécseg and Steinby (1997, Section 15) and Stamer (2009, Proposition 2.15) give more information about their formal properties.

An example shall clarify the definition.

Example 4.7.2. The system

$$CT = (\{A/1, S/0\}, \{f/2\}, \{x\}, \{S \to A(x), A(\xi_1) \to \xi_1, A(\xi_1) \to A(f(\xi_1, \xi_1))\}, S)$$

is a CFTG generating the ΣX-tree language T of all balanced binary trees

$$\{x, f(x,x), f(f(x,x), f(x,x)), f(f(f(x,x), f(x,x)), f(f(x,x), f(x,x))), \ldots\} \ ,$$

which is not recognizable by Theorem 4.4.16. A derivation for $f(f(x,x), f(x,x))$ is

$$S \Rightarrow_{CT} A(x) \Rightarrow_{CT} A(f(x,x)) \Rightarrow_{CT} A(f(f(x,x), f(x,x))) \Rightarrow_{CT} f(f(x,x), f(x,x)) \ .$$

Moreover, the yield language of T is the language $\{x^{2^n} \mid n \in \mathbb{N}\}$. □

Hence, it is obvious that.

Theorem 4.7.3. Rec $\subset$ *CFTL*.

Another way to specify CFTLs is by *nondeterministic pushdown tree recognizers*, which generalize the usual pushdown automata by allowing trees instead of strings in both the input and stack. More precisely, it was shown that both nondeterministic bottom-up pushdown tree recognizers (Schimpf and Gallier, 1985, Theorem 5.3.1) and nondeterministic

top-down pushdown tree recognizers (Guessarian, 1983, Theorem 1) effectively accept exactly the class $CFTL$. For details we also refer the reader to Coquidé et al. (1994), and Gécseg and Steinby (1997).

Finally, note that a language is the yield of a CFTL if and only if it is an indexed language (Rounds, 1969, 1970b, Gécseg and Steinby, 1997, Proposition 15.2). For details on this language class, we refer to Aho (1967, 1968, 1969). Its applications to compilers and linguistics were also studied by Hayashi (1973), Gazdar (1988), Duske and Parchmann (1984), Partee et al. (1990), and Vijay-Shanker and Weir (1994).

4.8 A hierarchy of families of tree languages

The results of Remark 4.5.4 and Theorems 4.4.5, 4.5.3, 4.4.14 and 4.7.3 can be gathered in the Hasse diagram of Figure 4.8.1, which represents a hierarchy of well-known families of tree languages.

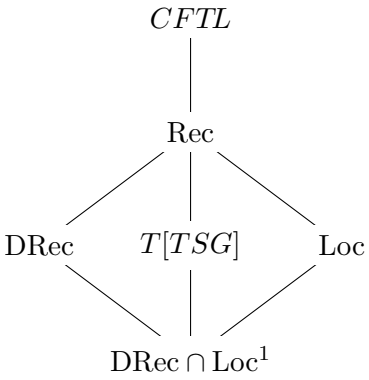


Figure 4.8.1: Hasse diagram representing an hierarchy of well-known families of tree languages, where $\mathrm{DRec} \cap \mathrm{Loc}^1$ denotes the class of all dT-recognizable local tree languages with exactly one root symbol.

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

Chapter 5
Tree Transformations

A tree transformation is any relation between two sets of trees and, by using the yield mapping of Section 4.1, any tree transformation defines a translation as well. For example, $\tau = \{(t_e, t_s) \mid t_e \in T_\Sigma(X), t_s \in T_\Omega(Y)\}$ represents a tree transformation that relates a parse tree t_e of an English sentence with the parse tree t_s of its Spanish translation. This naturally gives rise to a translation $\lambda(\tau) = \{(\mathrm{yd}(t_e), \mathrm{yd}(t_s)) \mid (t_e, t_f) \in \tau\}$ by taking the yields of the paired parse trees. Also, any syntax-directed translation can be viewed as a three-step process, one of which is a tree transformation τ: derivation trees of the input grammar of an SDTS are transformed to derivation trees of the output grammar of the SDTS, i.e., for any given input string $v \in X^*$, one constructs a derivation tree $s \in T_\Sigma(X)$ such that $\mathrm{yd}(s) = v$, takes a transform $t \in T_\Omega(Y)$ of s such that $(s, t) \in \tau$, and outputs the string $w = \mathrm{yd}(s) \in Y^*$ as a translation for v (cf. Čulík, 1965). Note that here (v, w) is an element in the translation defined by τ. For example, this makes it possible to design parsing algorithms that check whether a program is syntactically correct. If the program is correct, a derivation tree is returned as a convenient structural representation of the program. Then the tree transformation (i.e., syntax-directed translation device) turns this parse tree into a representation of the compiled program (for example, a program in machine code or pseudocode), which further exemplifies the natural use of syntax-directed translations in program compilation.

Since the 1960s and the 1970s when the theory of program schemata, syntax-directed translations, attributed grammars and semantic interpretation emerged, many researchers from both formal languages and NLP communities, mostly independently, defined and studied formal models that define tree transformation classes, especially *tree transducers* (see Gécseg and Steinby, 1984, Gécseg and Steinby, 1997, Comon et al., 2007, Maletti, 2010a, for complete expositions) and *synchronous tree grammars* (see Chiang and Knight, 2006, Chiang, 2006, Razmara, 2011, for good introductions). A quite comprehensive list of tree transducers and synchronous tree grammars found in the literature is exhibited in Section 8.3.4.

After fixing in Section 5.1 the general terminology and notation concerning tree transformations, we survey from the literature the main tree transducer types: *top-down tree transducers* (Section 5.2.1), *bottom-up tree transducers* (Section 5.2.2) and *extended top-down tree transducers* (Section 5.2.4), together with their formal properties (Table 5.2.1) and the inclusion relations between them (Figure 5.2.3). In Section 5.3, we present two formal grammars that generate pairs of trees, widely used to model linguistic phenomena encountered in machine translation, language interpretation and natural language generation: *synchronous tree-substitution grammars* (Section 5.3.1) and *generalized synchronous tree-substitution grammars* (Section 5.3.2), also known in the literature as synchronous tree-substitution grammars with states. Using our new formal definition, we get new characterizations of syntax-directed translations (Proposition 5.3.3 and Corollary 5.3.9). In addition, we give a direct and effective characterization in terms of generative devices of

(linear non-deleting) extended top-down tree transducers, which was differently proved by Maletti (2008). On the other hand, in Section 8.2.1, we propose for further research several theoretical topics related to generalized synchronous tree-substitution grammars.

5.1 Basic notions

Let Σ and Ω be ranked alphabets, and X and Y leaf alphabets. Relations of the form $\tau \subseteq T_\Sigma(X) \times T_\Omega(Y)$ are called *tree transformations*. The fact that $(s,t) \in \tau$ for some $s \in T_\Sigma(X)$ and $t \in T_\Omega(Y)$ means that τ may transform s to t, and t is then called a *transform* of s. The *input alphabets* of τ are Σ and X, and Ω and Y are the *output alphabets* of τ. Moreover, any such tree transformation τ naturally gives rise to a translation

$$\lambda(\tau) := \{(\mathrm{yd}(s), \mathrm{yd}(t)) \mid (s,t) \in \tau\} \ (\subseteq X^* \times Y^*) \ .$$

Hence, any tree transformation defining device also defines a translation. The class of tree transformations definable by devices of a given type TTD will be denoted by $\tau[TTD]$, and $\lambda[TTD]$ is its corresponding translation class. In what follows, the symbol τ always denotes a tree transformation. Next, we shall give some examples.

Example 5.1.1. Any collection of pairs of syntax trees of natural language sentences, where ranked alphabets code the grammatical categories such as noun or verb phrase and leaf alphabets usual natural languages alphabets and vocabularies like English, Kanji or Romanian, is a tree transformation.

Example 5.1.2. Any tree homomorphism $\varphi\colon T_\Sigma(X) \to T_\Omega(Y)$ defines a tree transformation

$$\tau(\varphi) := \{(t, t\varphi) \mid t \in T_\Sigma(X)\} \ (\subseteq T_\Sigma(X) \times T_\Omega(Y)) \ .$$

For example, if $\Sigma = \{f/2\}$ and $X = \{x\}$, then $\tau(\varphi) = \{(t,x) \mid t \in T_\Sigma(X)\}$ is the tree transformation defined by the tree homomorphism $\varphi\colon T_\Sigma(X) \to T_\Sigma(X)$ specified by $\varphi_2(f) := \xi_1$ and $\varphi_X(x) := x$. □

Since tree transformations are binary relations, all the general definitions and properties presented in Section 2.1 apply directly to them, too. Thus, the *converse*, or *inverse*, of a tree transformation $\tau \subseteq T_\Sigma(X) \times T_\Omega(Y)$ is the tree transformation

$$\tau^{-1} := \{(t,s) \mid (s,t) \in \tau\}$$

from $T_\Omega(Y)$ to $T_\Sigma(X)$, and for any $t \in T_\Sigma(X)$, $T \subseteq T_\Sigma(X)$, $u \in T_\Omega(Y)$ and $U \subseteq T_\Omega(Y)$,

- $t\tau := \{u \in T_\Omega(Y) \mid (t,u) \in \tau\}$ is the set of transforms of t,
- $T\tau := \bigcup_{t \in T} t\tau$ is the set of transforms of members of T,
- $u\tau^{-1} := \{t \in T_\Sigma(X) \mid (t,u) \in \tau\}$ is the pre-image of u, and
- $U\tau^{-1} := \bigcup_{u \in U} u\tau^{-1}$ is the pre-image of U.

In particular, the *domain* of τ is the set

$$\mathrm{Dom}(\tau) := T_\Omega(Y)\tau^{-1} = \{s \in T_\Sigma(X) \mid \exists t \in T_\Omega(Y) \text{ such that } (s,t) \in \tau\}$$

of all ΣX-trees that have at least one transform, and the *range* of τ is the set

$$\text{Range}(\tau) := T_\Sigma(X)\tau = \{t \in T_\Omega(Y) \mid \exists s \in T_\Sigma(X) \text{ such that } (s,t) \in \tau\}$$

of all ΩY-trees that are transforms of at least one ΣX-tree. Also, if $\tau \subseteq T_\Sigma(X) \times T_\Gamma(Z)$ and $\tau' \subseteq T_\Gamma(Z) \times T_\Omega(Y)$ are two tree transformations such that the output alphabets of τ are the input alphabets of τ', then their *composition* is the tree transformation

$$\tau \circ \tau' := \{(s,t) \mid s \in T_\Sigma(X), t \in T_\Omega(Y), \exists r \in T_\Gamma(Z) \text{ such that } (s,r) \in \tau \text{ and } (r,t) \in \tau'\}$$

from $T_\Sigma(X)$ to $T_\Omega(Y)$. The composition operation is extended in a natural way to classes of tree transformations: if $\mathcal{A}$ and $\mathcal{B}$ are classes of tree transformations, then

$$\mathcal{A} \circ \mathcal{B} = \{\tau \circ \tau' \mid \tau \in \mathcal{A}, \tau' \in \mathcal{B}\}$$

is the class of all tree transformations that are the composition of a tree transformation from $\mathcal{A}$ and a tree transformation from $\mathcal{B}$. Therefore, for any classes $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ of tree transformations,

- $\mathcal{A} \circ \mathcal{B} \subseteq \mathcal{C}$ means that any composition of an $\mathcal{A}$-transformation and a $\mathcal{B}$-transformation is a $\mathcal{C}$-transformation,
- $\mathcal{C} \subseteq \mathcal{A} \circ \mathcal{B}$ means that any $\mathcal{C}$-transformation can be *decomposed* into the product of a $\mathcal{A}$-transformation and a $\mathcal{B}$-transformation, and
- $\mathcal{C} \circ \mathcal{C} \subseteq \mathcal{C}$ means that $\mathcal{C}$ is *closed under composition*.

Moreover, a class $\mathcal{C}$ of tree transformations *preserves* a class $\mathcal{T}$ of tree languages if $T\tau \in \mathcal{T}$ for all $\tau \in \mathcal{C}$ and $T \in \mathcal{T}$.

Almost any interesting class of tree transformations includes as a subclass the class $\mathcal{ID}_\tau$ of *identity tree transformations* $\{(t,t) \mid t \in T_\Sigma(X)\}$. If $\mathcal{ID}_\tau \subseteq \mathcal{C}$ for some class $\mathcal{C}$, then $\mathcal{C} \subseteq \mathcal{C} \circ \mathcal{C}$ and, even more, $\mathcal{B} \subseteq \mathcal{C} \circ \mathcal{B}$ and $\mathcal{B} \subseteq \mathcal{B} \circ \mathcal{C}$ for any class $\mathcal{B}$ of tree transformations.

Furthermore, any useful tree transformation admits a finite (effective) specification method. Consequently, in what follows, we present the basic types of machines, called *tree transducers*, that compute tree transformations, and generating devices, called *synchronous grammars*, that generate pairs of trees. Moreover, we will survey the properties of the tree transformations classes defined by the presented devices, as well as their translation power.

5.2 Tree transducers

We consider tree transducers as formal models of syntax-directed translations (cf. Fülöp and Vogler, 1998). A tree transducer is a finite-state machine which computes a tree transformation. Given an input tree over the input (ranked and leaf) alphabets, the tree transducer computes, using a finite set of transition rules, an output tree over the output (ranked and leaf) alphabets. Due to the complexity of trees and the asymmetry between top-down and bottom-up direction of processing a tree, there is a rich variety of different types of tree transducers and corresponding classes of tree transformations. In this section we shall introduce the basic tree transducer types and the tree transformations defined by them. We borrowed

the notation and terminology developed by Engelfriet (1975a), Steinby (2005) and Fülöp (2004), but also Gécseg and Steinby (1984), Gécseg and Steinby (1997), Fülöp and Vogler (1998), and Comon et al. (2007) have been consulted.

Before starting our exposition we introduce some auxiliary terminology and notation. For any alphabets Ω, Y and $Q = Q_1$, and any set H of trees, we define $T_\Omega(Y \cup Q(H))$ as the smallest set $\mathcal{T}$ such that:

(1) $Y \cup \Omega_0 \cup Q(H) \subseteq \mathcal{T}$;

(2) if $g \in \Omega_m$ and $t_1, \ldots, t_m \in \mathcal{T}$ for $m \geq 1$, then $g(t_1, \ldots, t_m) \in \mathcal{T}$.

If $H \subseteq T_\Sigma(X)$ and $Q \cap (\Sigma \cup X \cup \Omega \cup Y) = \emptyset$, then each tree $t \in T_\Omega(Y \cup Q(H))$ has a unique representation $t = r[q_1(s_1), \ldots, q_n(s_n)]$, where $n \in \mathbb{N}$, $r \in \widetilde{T}_\Omega(Y \cup \Xi_n)$, $q_1, \ldots, q_n \in Q$ and $s_1, \ldots, s_n \in H$.

Example 5.2.1. If $\Sigma = \{g/1, e/0\}$, $X = \{x\}$, $\Omega = \{f/2\}$, $Y = \{y\}$, $Q = \{q/1\}$ and $H = \{g(x), e\}$, then $r = f(f(y, \xi_1), \xi_2) \in \widetilde{T}_\Omega(Y \cup \Xi_2)$ and $t = f(f(y, q(e)), q(g(x))) \in T_\Omega(Y \cup Q(H))$ has a unique representation $r[q(e), q(g(x))]$. □

5.2.1 Top-down tree transducers

A top-down, or root-to-frontier, tree transducer starts at the root of the given input tree and moves towards the frontier formed by the leaves. Because of this way of moving, we can discern immediately the two main properties of top-down tree transducers: during a computation, a subtree of the input tree may be deleted before processing, or all copies of the subtree are then processed independently, possibly in different ways. The precise definition follows (Rounds, 1970a, Thatcher, 1970).

Definition 5.2.2. A *top-down tree transducer* (TOP-transducer) is a system $TD = (Q, \Sigma, X, \Omega, Y, P, I)$ that consists of the following parts.

(1) Q is the unary ranked alphabet of *states* such that $Q \cap (\Sigma \cup X \cup \Omega \cup Y) = \emptyset$.

(2) Σ and X are the *input alphabets*, and Ω and Y are the *output alphabets*.

(3) P is a finite set of *rules* each one of which is either of the form

(T1) $q(d) \to r$, where $q \in Q$, $d \in X \cup \Sigma_0$ and $r \in T_\Omega(Y)$, or of the form

(T2) $q(f(\xi_1, \ldots, \xi_m)) \to r(q_1(\xi_{i_1}), \ldots, q_k(\xi_{i_k}))$, where $q \in Q$, $m \geq 1$, $f \in \Sigma_m$, $k \geq 0$, $q_1, \ldots, q_k \in Q$ and $r \in T_\Omega(Y \cup Q(\Xi_m))$ ($i_1, \ldots, i_k \in [m]$ and k is the total number of occurrences of the variables ξ_i in r).

(4) $I \subseteq Q$ is the set of *initial states*. □

The one-step *derivation relation* $\Rightarrow_{TD}$ is defined as follows. For any $s, t \in T_\Omega(Y \cup Q(T_\Sigma(X)))$, $s \Rightarrow_{TD} t$ holds if t is obtained from s either

- by replacing an occurrence of a subtree $q(d)$ with r, where $q(d) \to r$ is a rule in P of type (T1), or
- by replacing an occurrence of a subtree $q(f(s_1, \ldots, s_m))$ with $r(q_1(s_{i_1}), \ldots, q_k(s_{i_k}))$, where $q(f(\xi_1, \ldots, \xi_m)) \to r(q_1(\xi_{i_1}), \ldots, q_k(\xi_{i_k}))$ is a rule in P of type (T2).

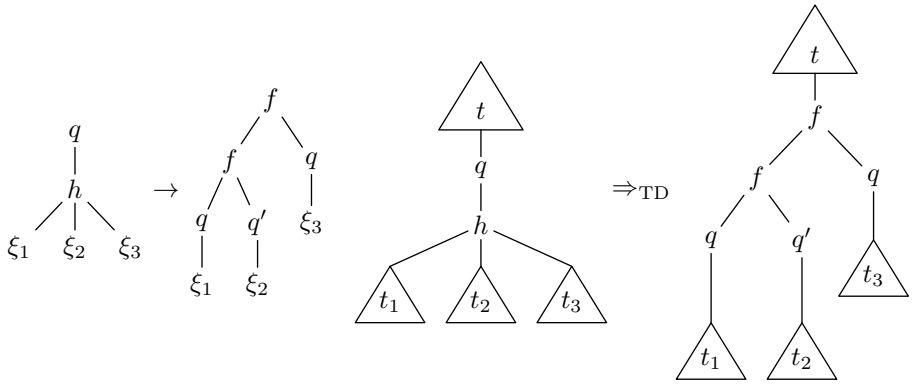


Figure 5.2.1: A sample rule in a TOP-transducer and an illustration of a derivation step using that rule.

Then, the *tree transformation computed* by TD is the relation

$$\tau(TD) := \{(s,t) \mid s \in T_\Sigma(X), t \in T_\Omega(Y), q(s) \Rightarrow^*_{TD} t \text{ for some } q \in I\} \ .$$

If $\tau = \tau(TD)$ for some TOP-transducer TD, then τ is called a *top-down tree transformation* (TOP-transformation). The class of all TOP-transformations is denoted by τ[TOP], and λ[TOP] denotes the class of all translations defined by TOP-transducers.

Example 5.2.3. Consider $\Sigma = \{h/3, f/2, g/1\}$, $X = \{x\}$ and $Q = \{q, q'\}$. Let $TD = (Q, \Sigma, X, \Sigma, X, P, \{q\})$ be a top-down tree transducer with the following rules

$$\begin{aligned} q(h(\xi_1,\xi_2,\xi_3)) &\to f(f(q(\xi_1), q'(\xi_2)), q(\xi_3)) & \qquad q(x) &\to x \\ q(g(\xi_1)) &\to q(\xi_1) & \qquad q'(x) &\to x \\ q'(g(\xi_1)) &\to g(q'(\xi_1)) \ . \end{aligned}$$

Then $q(h(g^l(x), g^m(x), g^n(x))) \Rightarrow^*_{TD} f(f(x, g^m(x)), x)$ for every $l, m, n \in \mathbb{N}$. The first rule and a derivation step involving that rule are illustrated in Figure 5.2.1. □

Because of TOP-transducers capability to make copies of subtrees of the input tree, it is obvious that a TOP-transformation does not always preserve recognizability and the class τ[TOP] is not closed under composition (Engelfriet, 1975a, Gécseg and Steinby, 1984, Comon et al., 2007). Therefore, it makes sense to consider also some restricted types of TOP-transducers.

Definition 5.2.4. A type (T2) rule $q(f(\xi_1, \ldots, \xi_m)) \to r(q_1(\xi_{i_1}), \ldots, q_k(\xi_{i_k}))$ is

- *linear* if no ξ_i ($i \in [m]$) appears more than once in r, and
- *non-deleting* if every ξ_i ($i \in [m]$) appears at least once in r.

Any type (T1) rule $q(d) \to r$ is regarded as both linear and non-deleting. □

Now we can define some special types of top-down tree transducers.

Definition 5.2.5. A TOP-transducer $TD = (Q, \Sigma, X, \Omega, Y, P, I)$ is called

(a) *linear*, if every rule in P is linear,

(b) *non-deleting*, if every rule in P is non-deleting,

(c) *deterministic*, if I consists of one state only, and there are no two rules in P with the same left-hand side, and

(d) *total*, if P contains

(1) for each pair $q \in Q$ and $d \in X \cup \Sigma_0$ a rule $q(d) \to r$, and

(2) for each pair $q \in Q$ and $f \in \Sigma_m$ $(m \geq 1)$, a rule $q(f(\xi_1, \ldots, \xi_m)) \to r$.

Following our general convention, the classes of tree transformations which correspond to the subclasses (a)-(d) of TOP-transducers are denoted by τ[l-TOP], τ[n-TOP], τ[d-TOP] and τ[t-TOP], respectively. Moreover, λ[l-TOP], λ[n-TOP], λ[d-TOP] and λ[t-TOP], respectively, denote the classes of translations defined by the corresponding subclasses (a)-(d) of TOP-transducers. If more than one restriction is imposed, then TOP is prefixed by the corresponding letters. For example, τ[ln-TOP] (λ[ln-TOP], respectively) denotes the class of tree transformations (translations, respectively) computed by linear non-deleting TOP-transducers. □

The TOP-transducer of Example 5.2.3 is linear, non-deleting and deterministic but it is not total (there is no rule defined for the pair q and f or for the pair q' and h).

Because a TOP-transducer has to decide whether to delete a subtree before reading it, no TOP-transducer is capable of computing the tree transformation $\tau = \{(f(x,x),x)\}(\subseteq T_\Sigma(X) \times T_\Sigma(X)$, where $\Sigma = \{f/2\}$ and $X = \{x\})$. Such drawback is solved by equipping a TOP-transducer with a look-ahead facility: the possibility to inspect subtrees in some way before deciding what rule to apply (Engelfriet, 1977). Note that the idea of regular look-ahead also occurs in the theory of parsing of CFLs (Culik II and Cohen, 1973).

Definition 5.2.6. A *top-down tree transducer with regular look-ahead* (TOP$^{\mathrm{R}}$-transducer) is a device $TD^{\mathrm{R}} = (Q, \Sigma, X, \Omega, Y, P, I)$, where Q, Σ, X, Ω, Y and I are specified as in Definition 5.2.2 for TOP-transducers and P is a finite set of rules of the form

$$\langle q(f(\xi_1, \ldots, \xi_m)) \to r(q_1(\xi_{i_1}), \ldots, q_k(\xi_{i_k})), M \rangle$$

with $q(f(\xi_1, \ldots, \xi_m)) \to r(q_1(\xi_{i_1}), \ldots, q_k(\xi_{i_k}))$ a (T2)-rule of the usual kind (see Definition 5.2.2), and $M \colon \Xi_m \to \wp(T_\Sigma(X))$ a mapping that defines the domain of application of the rule by specifying for each ξ_i, $i \in [m]$, a recognizable ΣX-tree language $M(\xi_i)$. The look-ahead M is *finite* if for every $i \in [m]$, $M(\xi_i)$ is a finite tree language. □

The semantics of a TOP$^{\mathrm{R}}$-transducer is defined as for a TOP-transducer with the additional condition that a rule

$$\langle q(f(\xi_1, \ldots, \xi_m)) \to r(q_1(\xi_{i_1}), \ldots, q_k(\xi_{i_k})), M \rangle$$

can be applied in state q at the root of an input subtree $f(s_1, \ldots, s_m)$ exactly in case $s_i \in M(\xi_i)$ for all $i \in [m]$. The class of tree transformations computed by top-down tree transducers with finite (respectively, regular) look-ahead is denoted by τ[TOP$^{\mathrm{F}}$] (respectively, τ[TOP$^{\mathrm{R}}$]). Moreover, λ[TOP$^{\mathrm{F}}$] and λ[TOP$^{\mathrm{R}}$] denote the corresponding classes of translations defined by these tree transducers.

Example 5.2.7. The system

$$TD^{\mathrm{R}} = (\{q_0\}, \{f/2\}, \{x\}, \{f/2\}, \{x\}, \{\langle q_0(f(\xi_1, \xi_2)) \to x, M\rangle\}, \{q_0\}) \ ,$$

where $M(\xi_1) = M(\xi_2) := \{x\}$, is a TOP-transducer with finite (and therefore regular) look-ahead, which computes the tree transformation $\{(f(x,x),x)\}$. □

Moreover, the definition of linear and non-deleting $\mathrm{TOP}^{\mathrm{R}}$-transducers is identical to the top-down case (see Definition 5.2.5). The class of tree transformations computed by linear top-down tree transducers with finite (respectively, regular) look-ahead is denoted by $\tau[\text{l-TOP}^{\mathrm{F}}]$ (respectively, $\tau[\text{l-TOP}^{\mathrm{R}}]$).

5.2.2 Bottom-up tree transducers

A bottom-up, or frontier-to-root, tree transducer starts at the leaves of the given input tree and moves towards the root. Because of this way of moving, we can distinguish immediately the two main properties of these tree transducers: during a computation, a subtree of the input tree cannot be deleted before processing, or it is first processed and then the (only) result may be multiplied. The precise definition follows (Thatcher, 1973).

Definition 5.2.8. A *bottom-up tree transducer* (BOT-transducer) is a system $BU = (Q, \Sigma, X, \Omega, Y, P, F)$ that consists of the following parts.

(1) Q is the unary ranked alphabet of *states* such that $Q \cap (\Sigma \cup X \cup \Omega \cup Y) = \emptyset$.

(2) Σ and X are the *input alphabets*, and Ω and Y are the *output alphabets*.

(3) P is a finite set of *rules* each one of which is either of the form

(B1) $d \to q(r)$, where $d \in X \cup \Sigma_0$, $q \in Q$ and $r \in T_\Omega(Y)$, or of the form

(B2) $f(q_1(\xi_1), \ldots, q_m(\xi_m)) \to q(r(\xi_{i_1}, \ldots, \xi_{i_k}))$, where $m \geq 1$, $f \in \Sigma_m$, $k \geq 0$, $q_1, \ldots, q_k, q \in Q$ and $r \in T_\Omega(Y \cup \Xi_m)$ $(i_1, \ldots, i_k \in [m]$, and k is the total number of occurrences of the variables ξ_i in $r)$.

(4) $F \subseteq Q$ is the set of *final states*. □

The one-step *derivation relation* $\Rightarrow_{BU}$ is defined as follows. For any $s, t \in T_\Sigma(X \cup Q(T_\Omega(Y)))$, $s \Rightarrow_{BU} t$ holds if t is obtained from s either

- by replacing an occurrence of a subtree $d \in X \cup \Sigma_0$ with $q(r)$, where $d \to q(r)$ is a rule in P of type (B1), or
- by replacing an occurrence of a subtree $f(q_1(t_1), \ldots, q_m(t_m))$ with $q(r(t_{i_1}, \ldots, t_{i_k}))$, where $f(q_1(\xi_1), \ldots, q_m(\xi_m)) \to q(r(\xi_{i_1}, \ldots, \xi_{i_k}))$ is a rule in P of type (B2).

Thus, the *tree transformation computed* by BU is the relation

$$\tau(BU) := \{(s,t) \mid s \in T_\Sigma(X), t \in T_\Omega(Y), s \Rightarrow_{BU}^* q(t) \text{ for some } q \in F\} \ .$$

If $\tau = \tau(BU)$ for some BOT-transducer BU, then τ is called a *bottom-up tree transformation* (BOT-transformation). The class of all BOT-transformations is denoted by $\tau[\text{BOT}]$, and $\lambda[\text{BOT}]$ denotes the class of all translations defined by TOP-transducers.

Example 5.2.9. The system

$$BU = (\{q, q'\}, \{g/1\}, \{x\}, \{f/2, h/1, h'/1\}, \{y\}, P, \{q'\})$$

with P consisting of the rules

$$\begin{aligned} g(q(\xi_1)) &\to q(h(\xi_1)) \\ g(q(\xi_1)) &\to q(h'(\xi_1)) \\ g(q(\xi_1)) &\to q'(f(\xi_1, \xi_1)) \\ x &\to q(y) \end{aligned}$$

is a BOT-transducer computing the tree transformation

$$\tau(BU) = \{(g^n(x), f(t,t)) \mid n \geq 1, t \in T_{\{h/1,h'/1\}}(\{y\}), \mathrm{hg}(t) = n-1\} \ .$$

Then

$$\begin{aligned} g(g(g(x))) &\Rightarrow_{BU} g(g(g(q(y)))) \Rightarrow_{BU} g(g(q(h(y)))) \Rightarrow_{BU} g(q(h'(h(y)))) \\ &\Rightarrow_{BU} q'(f(h'(h(y)), h'(h(y)))) \end{aligned}$$

is a sample computation in BU. □

Let us now introduce the special BOT-transducers that correspond to the types of TOP-transducers defined in Section 5.2.1.

Definition 5.2.10. Let $BU = (Q, \Sigma, X, \Omega, Y, P, F)$ be a BOT-transducer. A type (B2) rule $f(q_1(\xi_1), \ldots, q_m(\xi_m)) \to q(r(\xi_{i_1}, \ldots, \xi_{i_k}))$ is called

- *linear* if no ξ_i $(i \in [m])$ appears more than once in r, and
- *non-deleting* if every ξ_i $(i \in [m])$ appears at least once in r.

Any type (B1) rule $d \to q(r)$ is regarded as both linear and non-deleting. Then BU is called

(a) *linear*, if every rule in P is linear,

(b) *non-deleting*, if every rule in P is non-deleting,

(c) *deterministic*, if there are no two rules in P with the same left-hand side, and

(d) *total*, if P contains

(1) a rule $d \to q(r)$ of type (B1) for any $d \in X \cup \Sigma_0$, and

(2) a rule $f(q_1(\xi_1), \ldots, q_m(\xi_m)) \to q(r(\xi_{i_1}, \ldots, \xi_{i_k}))$ of type (B2) for any $f \in \Sigma_m$ $(m \geq 1)$ and $q_1, \ldots, q_m \in Q$. □

The classes of tree transformations which correspond to the subclasses (a)-(d) of BOT-transducers are denoted by τ[l-BOT], τ[n-BOT], τ[d-BOT] and τ[t-BOT], respectively. If more than one restriction is imposed, then BOT is prefixed by the corresponding letters. For example, τ[ln-BOT] denotes the class of tree transformations computed by linear non-deleting BOT-transducers.

5.2.3 Some special tree transformation classes

In this section we present some special tree transformations that are computed both by TOP- and BOT-transducers (cf. Engelfriet, 1975a, c, 2015).

First, note that for any tree homomorphism $\varphi\colon T_\Sigma(X) \to T_\Omega(Y)$, its tree transformation $\tau(\varphi)$ (see Example 5.1.2) is computed by the 1-state TOP-transducer $TD = (\{q\}, \Sigma, X, \Omega, Y, P, \{q\})$, where P consists of the following rules

- $q(x) \to \varphi_X(x)$ for every $x \in X$,
- $q(e) \to \varphi_0(e)$ for every $e \in \Sigma_0$, and
- $q(f(\xi_1, \ldots, \xi_m)) \to \varphi_m(f)[q(\xi_1), \ldots, q(\xi_m)]$ for every $m \geq 1$ and $f \in \Sigma_m$.

In an analogous way, a 1-state BOT-transducer computing $\tau(\varphi)$ may be defined. Let HOM denote the class of all tree homomorphisms viewed as tree transformations. Moreover, it is obvious that a linear or non-deleting tree homomorphism is linear or non-deleting, respectively, also as a tree transformation.

A *relabeling* is a tree transformation that transforms each tree to some trees of exactly the same shape but the label of each node may be replaced with a symbol of the same rank.

Definition 5.2.11. A one-state TOP-transducer $TD = (\{q\}, \Sigma, X, \Omega, Y, P, \{q\})$ is a *relabeling TOP-transducer* if each rule in P has one of the following forms:

(1) $q(x) \to y$ with $q \in Q$, $x \in X$ and $y \in Y$;

(2) $q(e) \to e'$ with $q \in Q$, $e \in \Sigma_0$ and $e' \in \Omega_0$;

(3) $q(f(\xi_1, \ldots, \xi_m)) \to g(q(\xi_1), \ldots, q(\xi_m))$, where $q \in Q$, $m \geq 1$, $f \in \Sigma_m$ and $g \in \Omega_m$.

A tree transformation is called a *relabeling* if it is computed by a relabeling TOP-transducer. Let REL denote the class of all relabellings. □

It is easy to see that exactly the same relabellings would be computed by similarly defined one-state BOT-transducers.

On the other hand, since they are also natural generalizations of alphabetic tree homomorphisms, we may define a relabeling as a tree transformation $\eta \subseteq T_\Sigma(X) \times T_\Omega(Y)$ which is determined by a mapping $\eta_X\colon X \to \wp(Y)$ and mappings $\eta_m\colon \Sigma_m \to \wp(\Omega_m)$ for each $m \geq 0$ such that $\Sigma_m \neq \emptyset$, as follows:

(1) $x\eta := \eta_X(x)$ for $x \in X$;

(2) $e\eta := \eta_0(e)$ for $e \in \Sigma_0$;

(3) $s\eta := \{g(t_1, \ldots, t_m) \mid g \in \eta_m(f), (s_1, t_1), \ldots, (s_m, t_m) \in \eta\}$ for $s = f(s_1, \ldots, s_m)$ $(m \geq 1)$.

The relabeling η is *total* if none of the sets $\eta_X(x)$, $\eta_0(c)$ or $\eta_m(f)$ is empty.

A more general type of relabeling is obtained if the set of symbols that may replace a given input symbol at node ω depends on some finite-state information about the input subtree rooted at ω.

Definition 5.2.12. A TOP-transducer $TD = (Q, \Sigma, X, \Omega, Y, P, I)$ is a *finite-state relabeling TOP-transducer* if each rule in P has one of the following forms:

(1) $q(x) \to y$ with $q \in Q$, $x \in X$ and $y \in Y$;

(2) $q(e) \to e'$ with $q \in Q$, $e \in \Sigma_0$ and $e' \in \Omega_0$;

(3) $q(f(\xi_1, \ldots, \xi_m)) \to g(q_1(\xi_1), \ldots, q_m(\xi_m))$, where $m \geq 1$, $f \in \Sigma_m$, $g \in \Omega_m$ and $q_1, \ldots, q_m$, $q \in Q$.

A tree transformation is called a *finite-state relabeling* if it is computed by a finite-state relabeling TOP-transducer. Let QREL denote the class of finite-state relabellings. Moreover, the class of translations defined by finite-state relabeling TOP-transducers is denoted by λ[QREL]. □

Again, it is easy to see that finite-state relabelings are also computed by a similarly defined class of BOT-transducers. Moreover, they are defined by top-down or bottom-up *shape preserving tree transducers* of Fülöp and Gazdag (2003) and Gazdag (2006a) (cf. Gazdag, 2006b, for an overview).

Any ndB ΣX-recognizer $BR = (Q, \Sigma, X, P, F)$, where states are unary symbols, can be converted into a BOT-transducer $BU = (Q, \Sigma, X, \Sigma, X, P, F)$ that defines the tree transformation $\tau = \{(t, t) \mid t \in T(BR)\}$ $(\subseteq T_\Sigma(X) \times T_\Sigma(X))$, i.e., for any $t \in T_\Sigma(X)$, $t\tau = \{t\}$ if $t \in T(BR)$, and $t\tau = \emptyset$ if $t \notin T(BR)$. The class of these tree transformations is denoted by FTA. Note they can also be computed by non-deterministic top-down tree recognizers viewed as TOP-transducers.

5.2.4 Extended top-down tree transducers

Already Rounds (1970a) proposed top-down transducers with limited look-ahead, and subsequently, Engelfriet (1975a) introduced and studied the more powerful top-down tree transducers with regular look-ahead. The tree transducers to be considered in this section have a finite look-ahead capability but they also process in one step the whole inspected part of the input tree. Both bottom-up and top-down versions of such transducers were considered already in late 1970s by Dauchet (1975), Arnold and Dauchet (1976a), and Lilin (1978), but the recent interest in them stems from the demands of NLP, as explained by Graehl and Knight (2004), Knight and Graehl (2005), Graehl et al. (2008), Maletti et al. (2009) and Maletti (2008, 2010a), for example. We shall adopt the current terminology and speak about extended (top-down) tree transducers.

We start our exposition by presenting our formal definition, which is an adaptation of the one used by Knight and Graehl (2005), Maletti (2008, 2010a) and Maletti et al. (2009), for example.

Definition 5.2.13. An *extended top-down tree transducer* (XTT-transducer) is a system $XT = (Q, \Sigma, X, \Omega, Y, P, I)$ specified as follows.

(1) Q is a unary ranked alphabet of *states* such that $Q \cap (\Sigma \cup X \cup \Omega \cup Y) = \emptyset$.

(2) Σ and X are the *input alphabets*, and Ω and Y are the *output alphabets*.

(3) P is a finite set of *rules* of the form

$$q(\ell[\xi_1, \ldots, \xi_n]) \to r[q_1(\xi_{i_1}), \ldots, q_k(\xi_{i_k})] \ , \tag{5.2.1}$$

where $\ell \in \widetilde{T}_\Sigma(X \cup \Xi_n)$, $r \in \widetilde{T}_\Omega(Y \cup \Xi_k)$, $q, q_1, \ldots, q_k \in Q$ and $i_1, \ldots, i_k \in [n]$ for some $n, k \geq 0$.

(4) $I \subseteq Q$ is the set of *initial states*. □

For any $s, t \in T_{\Sigma\cup\Omega\cup Q}(X \cup Y)$, we write $s \Rightarrow_{XT} t$ if and only if there exist a context $c \in C_{\Sigma\cup\Omega\cup Q}(X \cup Y)$, a rule of type (5.2.1) in P and ΣX-trees $s_1, \ldots, s_n$ such that

$$s = c(q(\ell[s_1, \ldots, s_n])) \quad \text{and} \quad t = c(r[q_1(s_{i_1}), \ldots, q_k(s_{i_k})]) \ .$$

Then the *tree transformation computed by* XT (XTT-transformation) is the relation

$$\tau(XT) := \{(s,t) \mid s \in T_\Sigma(X), t \in T_\Omega(Y), q(s) \Rightarrow^*_{XT} t \text{ for some } q \in I\} \ ,$$

and the *translation defined by* XT is naturally the relation

$$\lambda(XT) := \{(\mathrm{yd}(s), \mathrm{yd}(t)) \mid (s,t) \in \tau(XT)\} \ (\subseteq X^* \times Y^*) \ .$$

The classes of all tree transformations and translations definable by XTT-transducers are denoted by τ[XTT] and λ[XTT], respectively.

Now, we shall give an example of an XTT-transducer and its features.

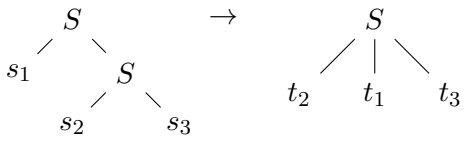
Example 5.2.14. The device $XT = (\{q\}, \{f/2\}, \{x\}, \{h/3\}, \{x\}, P, \{q\})$ with P consisting of the following 2 rules

$$q(f(\xi_1, f(\xi_2, \xi_3)) \to h(q(\xi_2), q(\xi_1), q(\xi_3))$$
$$q(x) \to x$$

is an XTT-transducer. A sample computation of XT is

$$q(f(x, f(f(x, f(x,x)), x))) \Rightarrow_{XT} h(q(f(x, f(x,x))), q(x), q(x)) \Rightarrow_{XT}$$
$$\Rightarrow_{XT} h(h(q(x), q(x), q(x)), q(x), q(x)) \Rightarrow^5_{XT} h(h(x,x,x), x, x) \ .$$

Note that XT expresses a common translation rule



specific to English-Arabic translations, where s_1, s_2, and s_3 are the parse trees of the subject, the verb, and the object in English and each t_i is the corresponding transform of s_i, $i \in [3]$, into Arabic. Maletti et al. (2009) showed that such a tree transformation cannot be computed by a linear TOP-transducer (cf. also Shieber, 2004).

Although, TOP-transducers can represent such a tree transformation by introducing new states and using a certain number of non-linear rules, such as

$$q(S(\xi_1, \xi_2)) \to S(q_1(\xi_2), q_2(\xi_1), q_3(\xi_2)) \ ,$$

and deleting rules, such as

$$q_1(VP(\xi_1, \xi_2)) \to q_2(\xi_1) \qquad \text{and} \qquad q_3(VP(\xi_1, \xi_2)) \to q_2(\xi_2) \ ,$$

this is computationally very costly in practical applications and uncharacteristic for natural-language phenomena since the copied subtrees may be processed differently as explained by Shieber (2004, Section 3.1) and Maletti (2010a, Section 3).

Many other examples that show the expressive power of XTT-transducers for linguistic applications are presented by Knight (2007), Knight and Graehl (2005), Maletti et al. (2009), Maletti (2010a), and Razmara (2011). □

Again, for practical purposes, it makes sense to consider special cases of extended top-down tree transducers as we did for TOP- and BOT- tree transducers.

Definition 5.2.15. Let $XT = (Q, \Sigma, X, \Omega, Y, P, I)$ be an XTT-transducer. A type (5.2.1) rule is called

- *linear* if each variable $\xi_i \in \Xi_n$ appears in r at most once,
- *non-deleting* if each variable $\xi_i \in \Xi_n$ appears in r at least once,
- *non-erasing* if $r \notin \Xi_n$, and
- an *epsilon-rule* if $\ell = \xi_1$ (and hence $n = 1$).

Then, XT is *linear* or *non-deleting* if all of its rules are, respectively, linear or non-deleting. Moreover, it is called *epsilon-free* or *non-erasing* if it has, respectively, no epsilon-rules or no erasing rules. □

We use the prefixes "l", "n", "q" and "e" followed by '-XTT' to denote the classes of tree transformations and translations which correspond to linear, non-deleting, quasi-alphabetic and epsilon-free XTT-transducers, respectively. For example, the XTT-transducer of Example 5.2.14 is linear, non-deleting, epsilon-free and non-erasing. Further subclasses of tree transformations and translations definable by XTT-transducers can be obtained by combining any of these prefixes. For example, the classes of all tree transformations and translations definable by linear non-deleting XTT-transducers are denoted by τ[ln-XTT] and λ[ln-XTT], respectively.

In the same way as it was done for TOP-transducers (see Definition 5.2.6), we can add a look-ahead facility to an XTT-transducer (Maletti et al., 2009). We will denote by $\tau[\mathrm{XTT}^{\mathrm{R}}]$ ($\tau[\mathrm{XTT}^{\mathrm{F}}]$, respectively), the class of tree transformations computed by XTT-*transducers with regular* (*finite*, respectively) *look-ahead*. Moreover, we use again the prefixes "l", "n" and "e" in front of $\mathrm{XTT}^{\mathrm{R}}$ to restrict the tree transformations in $\tau[\mathrm{XTT}^{\mathrm{R}}]$ to those computed by linear, non-deleting and epsilon-free, respectively, XTT-transducers with regular look-ahead. Combinations of prefixes are also allowed with the obvious effect.

5.2.5 Properties and hierarchies of tree transducers

We start this subsection by comparing, with respect to inclusion, some of the tree transformations classes computed by different versions of TOP-, BOT- and XTT-transducers introduced so far.

Thus, the Hasse diagram of Figure 5.2.2 (cf. Fülöp, 2004, Figure 8) shows the inclusion relations between the tree transformation classes τ[BOT], τ[TOP], τ[l-BOT], τ[l-TOP], τ[ln-BOT], τ[ln-TOP], QREL, $\tau[\text{l-TOP}^{\mathrm{F}}]$, and $\tau[\text{l-TOP}^{\mathrm{R}}]$. On the other hand, Figure 5.2.3 is the Hasse diagram of the inclusion relations between various classes of TOP- and XTT-

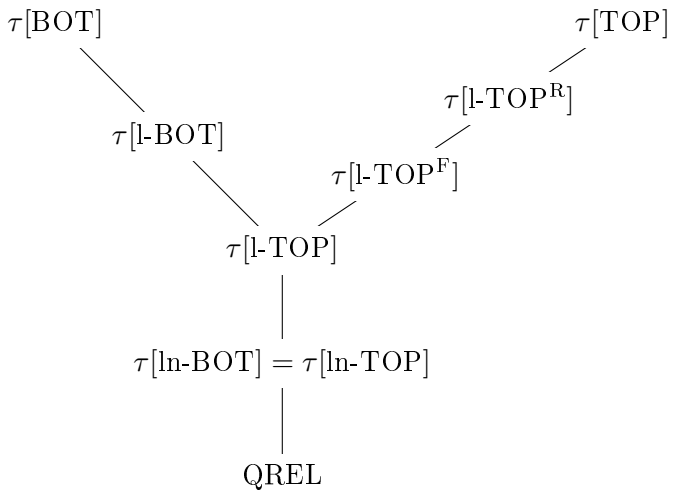


Figure 5.2.2: HASSE diagram representing the inclusion relations among some top-down and bottom-up tree transformations.

transformations (Maletti et al., 2009, Figure 4.3): τ[ln-XTT], τ[XTT], τ[l-XTT], τ[le-XTT], τ[lne-XTT], τ[e-XTT], τ[l-TOP], τ[ln-TOP], $\tau[\text{TOP}^{\text{R}}]$, τ[TOP], $\tau[\text{l-TOP}^{\text{F}}]$, $\tau[\text{l-TOP}^{\text{R}}]$, $\tau[\text{l-XTT}^{\text{R}}]$, $\tau[\text{TOP}^{\text{F}}]$, $\tau[\text{XTT}^{\text{R}}]$, $\tau[\text{le-XTT}^{\text{R}}]$, and $\tau[\text{e-XTT}^{\text{R}}]$,.

Compositions and decomposition results of tree transformations computed by tree transducers have been studied extensively by Engelfriet (1975a, c, 2015), Baker (1978b, 1979), Gécseg and Steinby (1984), Gécseg and Steinby (2015), Fülöp and Vágvölgyi (1987), Fülöp (1991), Fülöp and Vogler (1998), and Maletti et al. (2009). As an example, we show the following result to be used later on (cf. Engelfriet, 1975a, Theorem 3.5).

Theorem 5.2.16. *The following inclusion relations hold effectively in the sense that the indicated decompositions always can be found for a given BOT-, linear BOT-, or linear non-deleting BOT- transducer:*

(i) $\tau[\text{BOT}] \subseteq \text{REL} \circ \text{FTA} \circ \text{HOM}$;

(ii) $\tau[\text{l-BOT}] \subseteq \text{REL} \circ \text{FTA} \circ \text{l-HOM}$;

(iii) $\tau[\text{ln-BOT}] \subseteq \text{REL} \circ \text{FTA} \circ \text{ln-HOM}$.

In all three cases, the inclusion relation actually can be replaced with equality.

Proof. The computations of a BOT-transducer $BU = (Q, \Sigma, X, \Omega, Y, P, F)$ can be simulated by composing a relabeling, a tree transformation in FTA and a tree homomorphism as follows:

1. The relabeling replaces nondeterministically each label in the input tree by the name of a rule of BU that begins with that label (and hence could possibly be applied at that node).

2. The BOT-transducer defining the FTA-transformation checks that the states appearing in the new labels are consistent with the state behavior of BU, thus eliminating all trees that do not represent computations of BU.

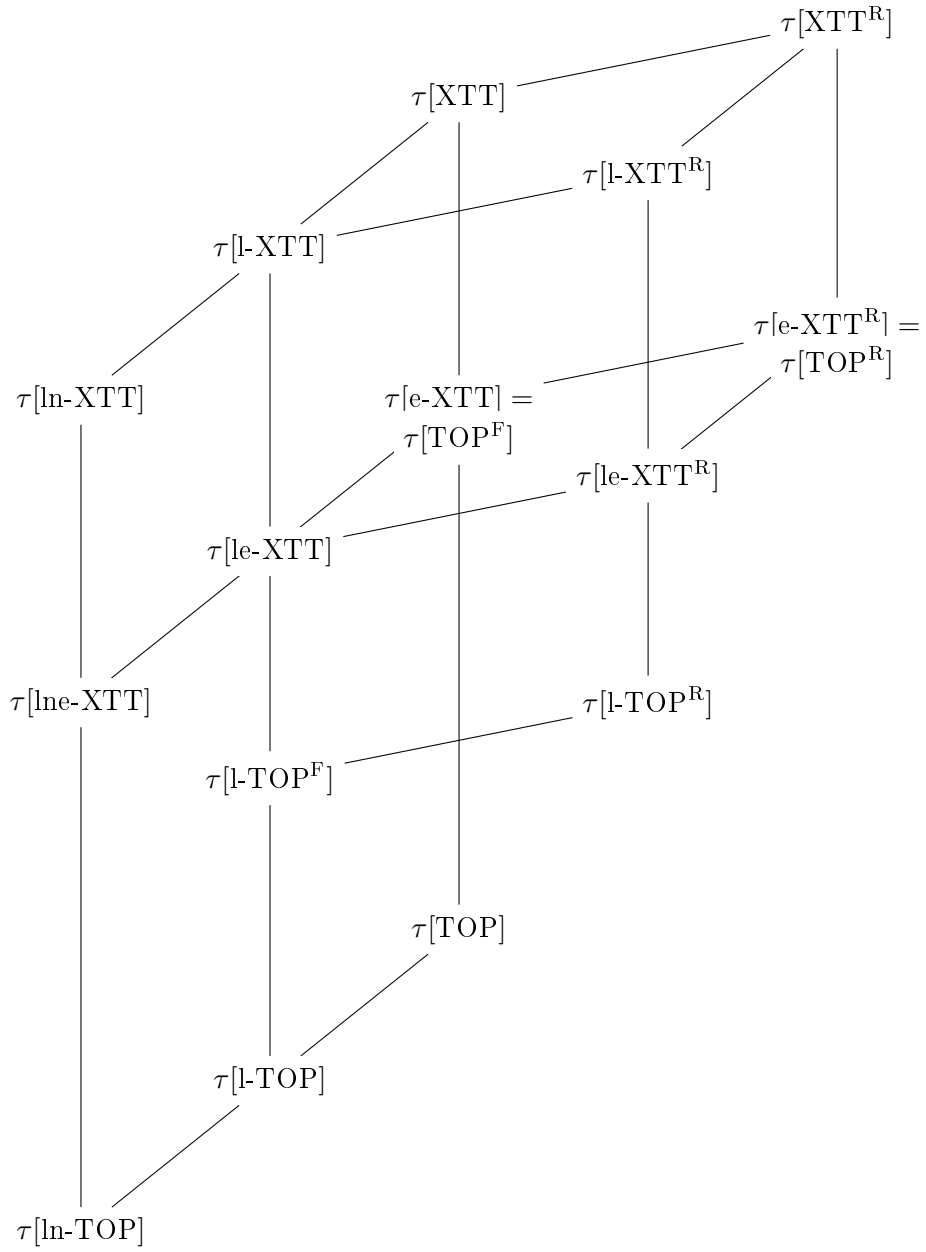


Figure 5.2.3: HASSE diagram representing the inclusion relations between the classes of tree transformations computed by extended top-down tree transducers.

Tree transducer model	EXPR	SYM	PRES	PRES^{-1}	COMP
TOP-transducers	-	-	-	+	-
Linear TOP-transducers	-	-	+	+	-
Linear non-deleting TOP-transducers	-	-	+	+	+
TOP^{R}-transducers	+	-	-	+	-
Linear TOP^{R}-transducers	-	-	+	+	+
BOT-transducers	+	-	-	+	-
Linear BOT-transducers	-	-	+	+	+
Linear non-deleting BOT-transducers	-	-	+	+	+
XTT-transducers	+	-	-	+	-
Linear XTT-transducers	+	-	+	+	-
Linear non-deleting XTT-transducers	+	+	+	+	-
XTT-transducers with regular look-ahead	+	-	-	+	-
Linear XTT-transducers with regular look-ahead	+	-	+	+	-

Table 5.2.1: Overview of formal properties of various TOP-, BOT- and XTT- transducers (+ stands for 'the property holds' and - means 'the property does not hold').

3. The tree homomorphism produces at each node the same output as BU would produce using that rule that appears as the label of the node.

If BU is a linear or linear non-deleting BOT-transducer, then the tree homomorphism of the above construction will also be linear or linear non-deleting, respectively, and hence also (ii) and (iii) follow. □

We should also mention the famous *Hierarchy Theorem* of Engelfriet (1982) (cf. also Gécseg and Steinby, 1984) that states, among other facts, that the composition powers of $\tau[\text{TOP}]$ and $\tau[\text{BOT}]$ form two properly ascending chains $\tau[\text{TOP}]^n \subset \tau[\text{TOP}]^{n+1}$ and $\tau[\text{BOT}]^n \subset \tau[\text{BOT}]^{n+1}$ for every $n \geq 1$ that are interleaved with each other. Note that for a tree transformation class $\mathcal{C}$, $\mathcal{C}^1 := \mathcal{C}$ and $\mathcal{C}^{n+1} := \mathcal{C}^n \circ \mathcal{C}$ for $n \geq 1$.

We conclude this section by exhibiting in Table 5.2.1 the main properties appealing for machine translation of several tree-transducer models (see Gécseg and Steinby, 1984, Gécseg and Steinby, 2015, Comon et al., 2007, Engelfriet, 1975c, Maletti, 2010a, Maletti et al., 2009, Knight, 2007, and the references therein)

5.3 Synchronous tree grammars

The idea to simultaneously generate pairs of strings using formal grammars (cf. Satta, 2004, 2009, for an overview) is about as old as the development of better compilers and the introduction of the SDTSs in the 1960s (see Chapter 3 and Section 8.3.2 for further references). In the 1990s the NLP community got interested in generating devices that synchronously produce pairs of trees, being motivated especially by

- automatic translation between natural languages (Abeillé et al., 1990, Schabes, 1990), where it is needed to explicitly represent subject-object swapping and discontinuous

constituents,

- language interpretation tasks (Shieber and Schabes, 1990b), where it is useful to relate a syntactic analysis of a natural language (e.g., English) to some other structure such as the associated semantics represented in a logical form language or the analysis of a sentence in another natural language (e.g., French), and
- natural language generation (Shieber and Schabes, 1990a, 1991).

Subsequently, many other synchronous grammars explicitly generating tree transformations were investigated, usually for machine translation purposes (cf. Chiang and Knight, 2006, Chiang, 2006, Razmara, 2011, for short overviews).

In Section 5.3.1, we deal with synchronous tree substitution grammars introduced by Schabes (1990) and further studied by Eisner (2003) and Shieber (2004). Basically, such a generative device consists of two TSGs where the mutual nonterminals marked for substitution are bijectively associated via a permutation. Here, we give a new formal definition (Definition 5.3.1), exemplify its usage by formalizing a well-known example in the NLP community (Example 5.3.2), and prove how this new definition makes it easier to see how synchronous tree substitution grammars relate to SDTSs (Proposition 5.3.7).

Next, in Section 5.3.2, we introduce generalized synchronous tree substitution grammars, also called *synchronous tree substitution grammars with states* by Fülöp et al. (2010) and Maletti (2010c, 2011a), where their weighted version is algorithmically studied for tasks in (statistical) machine translation. In such a synchronous grammar, the nonterminals (also called *states*) do not depend anymore on the given input and output alphabets, and hence, the root of the input tree (output tree, respectively) is not related to the nonterminal rewritten on the input left-hand side (output right-hand side, respectively). As explained by Knight (2007) and Fülöp et al. (2010), this freedom removes the nonterminal (i.e., state) information from the input and output tree, and therefore, it overcomes the formal drawbacks of the constructions of Shieber (2004), that used the less powerful synchronous tree substitution grammars. Moreover, our Definition 5.3.4 permits us to give a direct and effective characterization in terms of generating devices (Proposition 5.3.8) of linear non-deleting XTT-transducers as stated by Knight (2007) and differently proved by Maletti (2008). Also, we get new characterizations of syntax-directed translations (Proposition 5.3.7 and Corollary 5.3.9).

5.3.1 Synchronous tree substitution grammars

A synchronous tree-substitution grammar introduced by Schabes (1990) and further studied by Eisner (2003) and Shieber (2004) consists of two synchronously working TSGs. First we give a new formal definition of this notion (cf. Shieber, 2004) that makes it easier to see how synchronous tree-substitution grammars relate to other syntax-directed translations devices such as SDTSs.

Definition 5.3.1. A *synchronous tree-substitution grammar* (STSG) is a system $ST = (N, \Sigma, X, N', \Omega, Y, P, S, S')$ specified by the following clauses.

(1) Σ and X are the *input alphabets* of ST, and Ω and Y are the *output alphabets*.

(2) $N := \{f_\downarrow \mid f \in \Sigma \setminus \Sigma_0\}$ and $N' := \{f'_\downarrow \mid f' \in \Omega \setminus \Omega_0\}$ are the sets of *nonterminal symbols*.

(3) $S \in N$ and $S' \in N'$ are the *start symbols*.

(4) P is a finite set of *productions* $A; A' \to r; r' \, (\sigma)$ such that

(a) $A \in N$, $A' \in N'$, $r \in T_\Sigma(X \cup N)$ and $r' \in T_\Omega(Y \cup N')$,

(b) if $A = f_\downarrow$ and $A' = f'_\downarrow$, then $\mathrm{root}(r) = f$ and $\mathrm{root}(r') = f'$, and

(c) σ is a bijection between the (occurrences of) nonterminals in r' and those in r that *associates* with each nonterminal in r' a unique nonterminal in r.

It is assumed that $N \cap (\Sigma \cup X) = N' \cap (\Omega \cup Y) = \emptyset$. If $\mathrm{yd}_N(r) = A_1 A_2 \dots A_n$ and $\mathrm{yd}_{N'}(r') = A'_1 A'_2 \dots A'_n$, then σ in clause (4c) is given as a permutation of $[n]$ such that for every $i \in [n]$, the i^{th} nonterminal A'_i in r' is associated with the nonterminal $A_{\sigma(i)}$ of r. When $r \in T_\Sigma(X)$ and $r' \in T_\Omega(Y)$ in a production $A; A' \to r; r' \, (\sigma)$, we may omit the permutation σ (on the empty set) and write the production simply as $A; A' \to r; r'$. □

Consider now any STSG $ST = (N, \Sigma, X, N', \Omega, Y, P, S, S')$. Let $AP(ST)$ be the set of all *associated pairs of trees* (s, s') such that $s \in T_\Sigma(X \cup N)$, $s' \in T_\Omega(Y \cup N')$ and the nonterminals in s' are bijectively associated to those in s. If the trees $(s, s') \in AP(ST)$ contain n nonterminals each, then the associating bijection between them can be represented by a permutation ς on $[n]$ such that the i^{th} nonterminal of s' is associated with the $\varsigma(i)^{\text{th}}$ nonterminal of s. For any $(s, s'), (t, t') \in AP(ST)$, we write $(s, s') \Rightarrow_{ST} (t, t')$, if and only if there are contexts $c \in C_\Sigma(X \cup N)$, $c' \in C_\Omega(Y \cup N')$ and a production $A; A' \to r; r' \, (\sigma)$ in P such that $s = c(A)$, $s' = c'(A')$, $t = c(r)$, $t' = c'(r')$, and the explicit nonterminals A and A' are associated in the pair (s, s'). Moreover, the associating bijection of the pair (t, t') is inherited from (s, s') and (r, r') in the natural way. We have $(s, s') \Rightarrow^n_{ST} (t, t')$ if and only if there is an n-step *derivation*

$$(s, s') \Rightarrow_{ST} (s_1, s'_1) \Rightarrow_{ST} \dots \Rightarrow_{ST} (s_{n-1}, s'_{n-1}) \Rightarrow_{ST} (t, t')$$

of (t, t') from (s, s'). The set $SP(ST)$ of *synchronous pairs of trees* is now defined inductively as follows:

(1) $(S, S') \in SP(ST)$ and the only occurrences of S and S' are *associated* in this pair;

(2) if $(s, s') \in SP(ST)$ and $(s, s') \Rightarrow_{ST} (t, t')$, then $(t, t') \in SP(ST)$ and the associating bijection of (t, t') is defined as above.

Clearly, $SP(ST) = \{(s, s') \in AP(ST) \mid (S, S') \Rightarrow^*_{ST} (s, s')\}$. The *tree transformation defined by* ST is the relation

$$\tau(ST) := \{(s, s') \in SP(ST) \mid s \in T_\Sigma(X), s' \in T_\Omega(Y)\} \ ,$$

and the *translation defined by* ST is given by

$$\lambda(ST) := \mathrm{yd}(\tau(ST)) = \{(\mathrm{yd}(s), \mathrm{yd}(s')) \mid (s, s') \in \tau(ST)\} \ .$$

The classes of the STSG-definable tree transformations and the STSG-definable translations are denoted by $\tau[STSG]$ and $\lambda[STSG]$, respectively.

Next, we clarify the notions presented so far by formalizing, via our Definition 5.3.1, the machine translation example of Abeillé et al. (1990), Chiang (2006) and Chiang and Knight (2006).

Example 5.3.2. The device $ST = (N, \Sigma, X, N', \Omega, Y, P, S_\downarrow, S_\downarrow)$ with

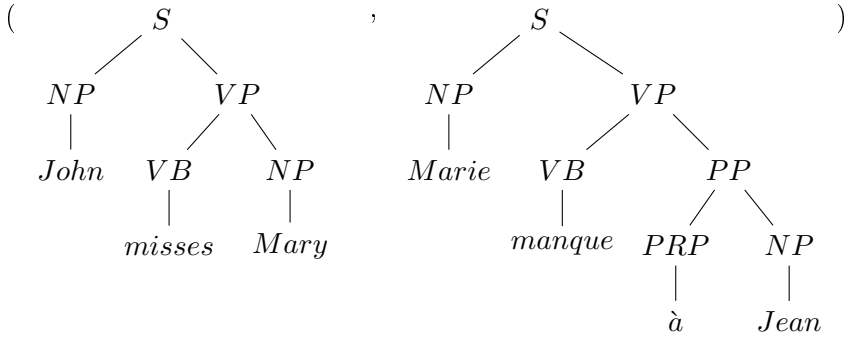
- $\Sigma = \{S/2, VP/2, NP/1, VB/1\}$,
- $X = \{John, misses, Mary\}$,
- $\Omega = \{S/2, VP/2, PP/2, VB/1, NP/1, PRP/1\}$,
- $Y = \{Marie, Jean, manque, \grave{a}\}$,
- $N = \{S_\downarrow, VP_\downarrow, NP_\downarrow, VB_\downarrow\}$,
- $N' = \{S_\downarrow, VP_\downarrow, PP_\downarrow, VB_\downarrow, NP_\downarrow, PRP_\downarrow\}$, and
- P consisting of the following five productions

$$S_\downarrow; S_\downarrow \to S(NP_\downarrow, VP(VB_\downarrow, NP_\downarrow)); S(NP_\downarrow, VP(VB_\downarrow, PP(PRP(\grave{a}), NP_\downarrow))) \ (3\,2\,1)$$
$$NP_\downarrow; NP_\downarrow \to NP(John); NP(Jean)$$
$$NP_\downarrow; NP_\downarrow \to NP(Mary); NP(Marie)$$
$$VB_\downarrow; VB_\downarrow \to VB(misses); VB(manque)$$

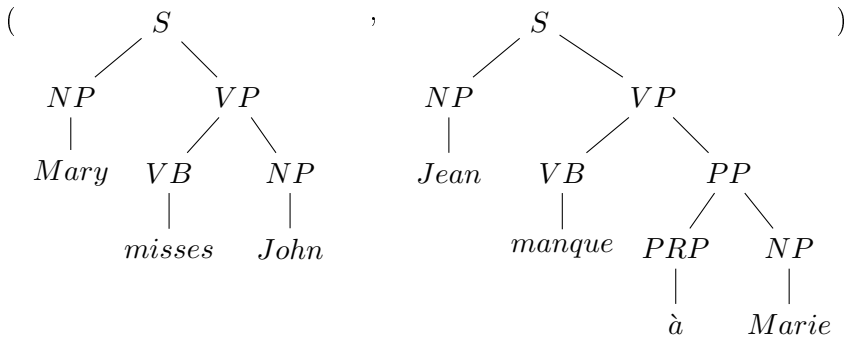
is an STSG. A derivation in ST is

$$\begin{aligned}
(S_\downarrow, S_\downarrow) &\Rightarrow_{ST} \big(S(NP_\downarrow, VP(VB_\downarrow, NP_\downarrow)), S_\downarrow(NP_\downarrow, VP(VB_\downarrow, PP(PRP(\grave{a}), NP_\downarrow)))\big)\ (3\,2\,1)\\
&\Rightarrow_{ST} \big(S(John, VP(VB_\downarrow, NP_\downarrow)), S_\downarrow(NP_\downarrow, VP(VB_\downarrow, PP(PRP(\grave{a}), Jean)))\big)\ (2\,1)\\
&\Rightarrow_{ST} \big(S(John, VP(misses, NP_\downarrow)), S_\downarrow(NP_\downarrow, VP(manque, PP(PRP(\grave{a}), Jean)))\big)\ (1)\\
&\Rightarrow_{ST} \big(S(John, VP(misses, Mary)), S_\downarrow(Marie, VP(manque, PP(PRP(\grave{a}), Jean)))\big)\ .
\end{aligned}$$

Thus, the following two pairs of trees are generated by ST:



and



Moreover, $\lambda(ST)$ contains the correct English-to-French translations (*John misses Mary, Marie manque à Jean*) and (*Mary misses John, Jean manque à Marie*).

Therefore, STSGs are able to naturally perform subject-object swapping characteristic to translations between natural languages with different structures without flattening the syntax trees and integrate the verb into a synchronous production (to make sure that this swapping does not occur with just any verb). This would not have been the case for SDTSs or SCFGs. □

When STSGs are defined this way, it seems quite obvious that any STSG-definable translation is also SDTS-definable. In Section 5.3.2, we shall prove this fact in a more general form. Here we note the converse statement.

Proposition 5.3.3. $\lambda[SDTS] \subseteq \lambda[STSG]$.

Proof. Let $SD = (N, X, Y, P, S)$ be an SDTS. By Theorem 3.3.12(iv), we may suppose without loss of generality that SD is proper and each production in P is of the form

(R1) $A; A \to A_1 \dots A_m; A_{\sigma(1)} \dots A_{\sigma(m)}\,(\sigma)$, where $m \geq 1$, $A, A_1, \dots, A_m \in N$, and σ is a permutation of $[m]$, or of the form

(R2) $A; A \to v; w$, where $A \in N$, $v \in X^*$ and $w \in Y^*$.

The *length* of a production $A; A \to \alpha; \beta\,(\sigma)$ is defined to be the maximum of the lengths of α and β. For any nonterminal $A \in N$, let $\lg(A)$ denote the maximum of the lengths of all the productions in P in which A appears in the left-hand side. Obviously, we may assume that there is no $A \in N$ with $\lg(A) = 0$. Otherwise, if SD has just the rule $S; S \to \varepsilon; \varepsilon$, the result is obvious because one can easily construct the STSG

$$(\{S_\downarrow\}, \{e/0\}, \emptyset, \{S_\downarrow\}, \{e/0\}, \emptyset, \{S_\downarrow; S_\downarrow \to S(e); S(e)\}, S_\downarrow, S_\downarrow)$$

that defines the same translation $(\varepsilon, \varepsilon)$ as SD.

We define now an STSG $ST = (\bar{N}, \Sigma, X, \bar{N}, \Sigma, Y, \bar{P}, S_\downarrow, S_\downarrow)$ as follows.

(1) Let $\Sigma_0 := \{e\}$, and $\Sigma_m := \{A \in N \mid \lg(A) = m\}$ for $m \geq 1$.

(2) Let $\bar{N} := \{A_\downarrow \mid A \in \Sigma \setminus \Sigma_0\}$.

(3) Each production in $\bar{P}$ is constructed from a production in P as follows:

(a) For any production $A; A \to A_1 \dots A_m; A_{\sigma(1)} \dots A_{\sigma(m)}\,(\sigma)$ in P of type (R1), we include in $\bar{P}$ the production

$$A_\downarrow; A_\downarrow \to A(A_{1\downarrow}, \dots, A_{m\downarrow}, e, \dots, e); A(A_{\sigma(1)\downarrow}, \dots, A_{\sigma(m)\downarrow}, e, \dots, e)\,(\sigma)\ ,$$

where there are $\lg(A) - m$ e's in each sequence.

(b) For any production $A; A \to x_1 \dots x_i; y_1 \dots y_j$ in P of type (R2) with $x_1, \dots, x_i \in X$ and $y_1, \dots, y_j \in Y$, we add to $\bar{P}$ the production

$$A_\downarrow; A_\downarrow \to A(x_1, \dots, x_i, e, \dots, e); A(y_1, \dots, y_j, e, \dots, e)\ ,$$

where the two sequences of e's are of length $\lg(A) - i$ and $\lg(A) - j$, respectively.

Let $\mathrm{yd}'_X : T_\Sigma(\bar{N} \cup X) \to (N \cup X)^*$ and $\mathrm{yd}'_Y : T_\Sigma(\bar{N} \cup Y) \to (N \cup Y)^*$ be the modified yield-mappings such $\mathrm{yd}'_X(x) := x$ for $x \in X$, $\mathrm{yd}'_Y(y) := y$ for $y \in Y$, $\mathrm{yd}'_X(A_\downarrow) = \mathrm{yd}'_Y(A_\downarrow) := A$ for any $A \in N$, and $\mathrm{yd}'_X(e) = \mathrm{yd}'_Y(e) := \varepsilon$. It is then obvious that if $A_\downarrow; A_\downarrow \to r; r'\,(\sigma)$ is a production in $\bar{P}$ obtained from the production $A; A \to \alpha; \beta\,(\sigma)$ in P, then $\mathrm{yd}'_X(r) = \alpha$ and $\mathrm{yd}'_Y(r') = \beta$.

Next we show that any derivation step in ST is matched by a corresponding step in SD. If $(s, s') \Rightarrow_{ST} (t, t')$, then there exist contexts $c \in C_\Sigma(X \cup \bar{N})$ and $c' \in C_\Sigma(Y \cup \bar{N})$ and a production $A_\downarrow; A_\downarrow \to r; r'\,(\sigma)$ in $\bar{P}$ such that $s = c(A_\downarrow)$, $s' = c'(A_\downarrow)$, $t = c(r)$ and $t' = c'(r')$, where the two $A_\downarrow$'s singled out are associated in the pair (s, s'). Furthermore, $\mathrm{yd}'_X(s) = \gamma A \delta$ and $\mathrm{yd}'_Y(s') = \gamma' A \delta'$ for some $\gamma, \delta \in (N \cup X)^*$ and $\gamma', \delta' \in (N \cup Y)^*$, and the two A's are associated. Then $\mathrm{yd}'_X(t) = \gamma \mathrm{yd}'_X(r) \delta$ and $\mathrm{yd}'_Y(t') = \gamma' \mathrm{yd}'_Y(r') \delta'$. Since $A; A \to \mathrm{yd}'_X(r); \mathrm{yd}'_Y(r')\,(\sigma)$ is in P, this means that $(\mathrm{yd}'_X(s), \mathrm{yd}'_Y(s')) \Rightarrow_{SD} (\mathrm{yd}'_X(t), \mathrm{yd}'_Y(t'))$ and we can complete the induction step. Now it is clear that for any derivation

$$(S_\downarrow, S_\downarrow) \Rightarrow_{ST} (s_1, s'_1) \Rightarrow_{ST} \ldots \Rightarrow_{ST} (s_n, s'_n) \qquad (D_{ST})$$

in ST, we get the derivation

$$(S, S) \Rightarrow_{SD} (\mathrm{yd}'_X(s_1), \mathrm{yd}'_Y(s'_1)) \Rightarrow_{SD} \ldots \Rightarrow_{SD} (\mathrm{yd}'_X(s_n), \mathrm{yd}'_Y(s'_n)) \qquad (D_{SD})$$

in SD. Hence, $\lambda(ST) \subseteq \lambda(SD)$.

The inclusion $\lambda(SD) \subseteq \lambda(ST)$ follows when one shows by induction on $n \geq 0$ that for every derivation

$$(S, S) \Rightarrow_{SD} (\gamma_1, \gamma'_1) \Rightarrow_{SD} \ldots \Rightarrow_{SD} (\gamma_n, \gamma'_n) \qquad (D'_{SD})$$

in SD, there is a derivation (D_{ST}) in ST such that $\mathrm{yd}'_X(s_i) = \gamma_i$ and $\mathrm{yd}'_Y(s'_i) = \gamma'_i$ for every $i \in [n]$. The case $n = 0$ is obvious, and for the induction step, we consider a derivation step $(\gamma, \gamma') \Rightarrow_{SD} (\delta, \delta')$ in SD and assume that there are trees $s \in T_\Sigma(\bar{N} \cup X)$ and $s' \in T_\Sigma(\bar{N} \cup Y)$ such that $\mathrm{yd}'_X(s) = \gamma$ and $\mathrm{yd}'_Y(s') = \gamma'$. If $A; A \to \alpha; \beta\,(\sigma)$ is the production applied in the derivation step, then there are contexts $c \in C_\Sigma(\bar{N} \cup X)$ and $c' \in C_\Sigma(\bar{N} \cup Y)$ such that $s = c(A_\downarrow)$ and $s' = c'(A_\downarrow)$, where two occurrences of $A_\downarrow$ are associated. Furthermore, $\mathrm{yd}'_X(s) = \delta_1 A \delta_2$ and $\mathrm{yd}'_Y(s') = \delta'_1 A \delta'_2$. Thus, when we apply the production $A_\downarrow; A_\downarrow \to r; r'\,(\sigma)$ of $\bar{P}$ (that corresponds to $A; A \to \alpha; \beta\,(\sigma)$) to the pair (s, s'), we get a derivation step $(s, s') \Rightarrow_{ST} (c(r), c'(r'))$ such that $\mathrm{yd}'_X(c(r)) = \delta_1 \alpha \delta_2 = \delta$ and $\mathrm{yd}'_Y(c'(r')) = \delta'_1 \beta \delta'_2 = \delta'$, and thus the induction can be completed. □

5.3.2 Generalized synchronous tree substitution grammars

The following generalization of the STSGs is suggested rather immediately by our Definition 5.3.1. Using this definition we will get new characterizations of syntax-directed translations and later, in Section 6.4, we will re-examine the representation of tree transformations defined by STSGs by certain tree bimorphisms suggested by Shieber (2004). Observe that as opposed to the STSGs, the nonterminals of this synchronous tree grammar are now separated from the input and output alphabets and the generation process is not guided anymore by the roots of the input and output trees. Also note that this model is called *synchronous tree substitution grammar with states* (i.e., nonterminal sets disjoint from the input and output alphabets) by Maletti (2010c, 2011a) and Fülöp et al. (2010), where

its weighted version is algorithmically studied and compared with other similar weighted formalisms.

Definition 5.3.4. A *generalized synchronous tree-substitution grammar* (GSTSG) is a system $GS = (N, \Sigma, X, N', \Omega, Y, P, S, S')$ specified as follows.

(1) Σ, X, Ω, Y, S and S' are as in an STSG (see Definition 5.3.1).

(2) N and N' are finite sets of *nonterminal symbols* such that $N \cap (\Sigma \cup X) = N' \cap (\Omega \cup Y) = \emptyset$.

(3) P is a finite set of *productions* $A; A' \to r; r'\,(\sigma)$ such that $A \in N$, $A' \in N'$, $r \in T_\Sigma(X \cup N)$ and $r' \in T_\Omega(Y \cup N')$, and σ is a bijection between the (occurrences of) nonterminals in r' and those in r that *associates* with each nonterminal in r' a unique nonterminal in r. Again, we write simply $A; A' \to r; r'$ when $r \in T_\Sigma(X)$ and $r' \in T_\Omega(Y)$.

The relations $\Rightarrow_{GS}$ and $\Rightarrow_{GS}^n$, as well as the tree transformation $\tau(GS)$ and the translation $\lambda(GS)$ are defined the same way as for an STSG. The classes of tree transformations and translations definable by GSTSGs are denoted by $\tau[GSTSG]$ and $\lambda[GSTSG]$, respectively. □

We call $(c(A), c'(A') \Rightarrow_{GS} (c(r), c'(r'))$, where $c \in C_\Sigma(X \cup N)$, $c' \in C_\Omega(Y \cup N')$ and $A; A' \to r; r'\,(\sigma)$ is a production in P, a *leftmost derivation step* if the explicit instance of A is the leftmost occurrence of any nonterminal symbol in $c(A)$, and in a *leftmost derivation* every step is leftmost. Since the productions of a GSTSG are 'context-free', every derivation can obviously be replaced with a leftmost one that yields the same result and in which exactly the same productions are used as in the original derivation.

Observe that any STSG is a special GSTSG in which the sets of nonterminal symbols depend on the ranked alphabets and the productions satisfy the additional condition (4b) of Definition 5.3.1.

Example 5.3.5. The system $GS = (\{S\}, \Sigma, X, \{S\}, \Sigma, X, P, S, S)$, where $\Sigma = \{f/1, g/1\}$, $X = \{x\}$, and

$$P = \{S; S \to f(S); f(S)\,(1), S; S \to g(S); g(S)\,(1), S; S \to x; x\}\ ,$$

is a GSTSG that defines the tree transformation $\tau(GS) = \{(s, s) \mid s \in T_\Sigma(X)\}$. It is clear that $\tau(GS)$ is not defined by any STSG because in an STSG the start symbols must be either $f_\downarrow$ or $g_\downarrow$; if, for example, the input start symbol is $f_\downarrow$, then no pair of the form $\big(g(s), g(s)\big)$ can be generated. □

This example, together with Definitions 5.3.1 and 5.3.4, allow us to state (cf. Fülöp et al., 2010, Maletti, 2011a).

Proposition 5.3.6. $\tau[STSG] \subset \tau[GSTSG]$.

The following proposition is a strengthened form of the converse of Proposition 5.3.3.

Proposition 5.3.7. $\lambda[GSTSG] \subseteq \lambda[SDTS]$.

Proof. Let $GS = (N, \Sigma, X, N', \Omega, Y, P, S, S')$ be any GSTSG, and consider the SDTS $SD = (N \times N', X, Y, P_0, (S, S'))$, where the productions in P_0 are obtained from the productions in P as follows. If $A; A' \to r; r'\,(\sigma)$ is a production in P such that

$$\mathrm{yd}(r) = v_0 A_1 v_1 \dots v_m A_m v_{m+1} \qquad \text{and} \qquad \mathrm{yd}(r') = w_0 A'_1 w_1 \dots w_m A'_m w_{m+1}\ ,$$

where $v_0, v_1, \dots, v_{m+1} \in X^*$, $A_1, \dots, A_m \in N$, $w_0, w_1, \dots, w_{m+1} \in Y^*$ and $A'_1, \dots, A'_m \in N'$, we include in P_0 the production $(A, A'); (A; A') \to \alpha; \beta\,(\sigma)$, where

$$\alpha = v_0 (A_1, A'_{\sigma^{-1}(1)}) v_1 \dots v_m (A_m, A'_{\sigma^{-1}(m)}) v_{m+1}$$

and

$$\beta = w_0 (A_{\sigma(1)}, A'_1) w_1 \dots w_m (A_{\sigma(m)}, A'_m) w_{m+1}\ .$$

Example Let $\Sigma = \{f/3, e/0\}$, $X = \{x\}$, $\Omega = \{h/3\}$ and $Y = \{y_1, y_2\}$. If $A, B, C \in N$ and $A', B' \in N'$ are nonterminals, then

$$A; A' \to f(f(x, A, B), e, C); h(y_1, h(B', y_2, A'), B'))\,(3\,2\,1)$$

is a possible GSTSG-production and

$$(A, A'); (A; A') \to x(A, B')(B, A')(C, B'); y_1(C, B') y_2(B, A')(A, B')\,(3\,2\,1)$$

is the corresponding SDTS-production. □

Let $\mu\colon (X \cup (N \times N'))^* \to (X \cup N)^*$ and $\mu'\colon (Y \cup (N \times N'))^* \to (Y \cup N')^*$ be the homomorphisms such that $x\mu := x$, $y\mu' := y$, $(A, A')\mu := A$ and $(A, A')\mu' := A'$ for any $x \in X$, $y \in Y$ and $(A, A') \in N \times N'$. Then it is easy to see that for any derivation

$$(S, S') \Rightarrow_{GS} (s_1, s'_1) \Rightarrow_{GS} \dots \Rightarrow_{GS} (s_n, s'_n) \qquad (D_{GS})$$

in GS there is a derivation

$$((S, S'), (S, S')) \Rightarrow_{SD} (\delta_1, \delta'_1) \Rightarrow_{SD} \dots \Rightarrow_{SD} (\delta_n, \delta'_n) \qquad (D_{SD})$$

in SD such that $\mathrm{yd}(s_i) = \delta_i\mu$ and $\mathrm{yd}(s'_i) = \delta'_i\mu'$ for every $i \in [n]$. If, in particular, $(s_n, s'_n) \in \tau(GS)$, then

$$(\mathrm{yd}(s_n), \mathrm{yd}(s'_n)) = (\delta_n\mu, \delta'_n\mu') = (\delta_n, \delta'_n) \in \lambda(SD)$$

because then $\mathrm{yd}(s_n) \in X^*$ and $\mathrm{yd}(s'_n) \in Y^*$. Hence, $\lambda(GS) \subseteq \lambda(SD)$.

For the converse, it suffices to verify by induction on the length of the derivation that for every derivation (D_{SD}) there is a derivation (D_{GS}) such that $\mathrm{yd}(s_i) = \delta_i\mu$ and $\mathrm{yd}(s'_i) = \delta'_i\mu'$ for every $i \in [n]$. □

Now we prove a result in view of which generalized synchronous tree substitution grammars appear as the generative counterparts of linear non-deleting XTT-transducers (Knight, 2007, p.127).

Proposition 5.3.8. $\tau[GSTSG] = \tau[\text{ln-XTT}]$.

Proof. Given a GSTSG $GS = (N, \Sigma, X, N', \Omega, Y, P, S, S')$ we introduce a linear non-deleting XTT-transducer $XT = (Q, \Sigma, X, \Omega, Y, P', I)$, where $Q := N \times N'$, $I := \{(S, S')\}$, and P' is defined as follows.

If $A; A' \to r; r'\,(\sigma)$ is in P, and $\mathrm{yd}_N(r) = A_1 \dots A_m$ and $\mathrm{yd}_{N'}(r') = A'_1 \dots A'_m$, where $m \geq 0$, $A_1, \dots, A_m \in N$ and $A'_1, \dots, A'_m \in N'$, then there are trees $\tilde{r} \in \widetilde{T}_\Sigma(X \cup \Xi_m)$ and $\tilde{r}' \in \widetilde{T}_\Omega(Y \cup \Xi_m)$ such that $r = \tilde{r}[A_1, \dots, A_m]$ and $r' = \tilde{r}'[A'_1, \dots, A'_m]$. Now we put into P' the rule

$$(A, A')(\tilde{r}) \to \tilde{r}'[(A_{\sigma(1)}, A'_1)(\xi_{\sigma(1)}), \dots, (A_{\sigma(m)}, A'_m)(\xi_{\sigma(m)})] \ .$$

For example, if

$$S; S \to f(A, f(B, C)); f\big(A, f(B, f(g(x), C))\big)\,(3\,2\,1)$$

is a production in the GSTSG GS, then the XTT-transducer rule

$$(S, S)\big(f(\xi_1, f(\xi_2, \xi_3))\big) \to f\big((C, A)(\xi_3), f((B, B)(\xi_2), f(g(x), (A, C)(\xi_1))\big)$$

is in P'.

It can be shown that $\tau(XT) = \tau(GS)$, and hence $\tau[GSTSG] \subseteq \tau[\text{ln-XTT}]$.

To prove the converse inclusion, it suffices to consider any linear non-deleting XTT-transducer $XT = (Q, \Sigma, X, \Omega, Y, P', \{q_0\})$ with exactly one initial state because $\tau[GSTSG]$ is obviously closed under union. Let $GS = (Q, \Sigma, X, Q, \Omega, Y, P, q_0, q_0)$ be the GSTSG in which P is obtained from P' as follows.

Because XT is linear and non-deleting, each rule in P' is of the form

$$q(\ell) \to r[q_1(\xi_{\sigma(1)}), \dots, q_n(\xi_{\sigma(n)})] \ ,$$

where $n \geq 0$, $\ell \in \widetilde{T}_\Sigma(X \cup \Xi_n)$, $r \in \widetilde{T}_\Omega(Y \cup \Xi_n)$, $q, q_1, \dots, q_n \in Q$ and σ is a permutation of $[n]$. For such a rule we put into P the production

$$q; q \to \ell[q_{\sigma(1)}, \dots, q_{\sigma(n)}]; r[q_1, \dots, q_n]\,(\sigma) \ .$$

For example, if

$$q(f(\xi_1, f(\xi_2, \xi_3)) \to h(q(\xi_2), q(\xi_1), q(\xi_3))$$

is the XTT-transducer rule of Example 5.2.14, then

$$q; q \to f(q_2, f(q_1, q_3)); h(q_1, q_2, q_3)\,(2\,1\,3) \ .$$

is the corresponding constructed GSTSG-production.

Again, it can be shown that this way $\tau(GS) = \tau(XT)$, and so $\tau[\text{ln-XTT}] \subseteq \tau[GSTSG]$, which concludes the proof. □

Finally, we gather the results of Propositions 5.3.6, 3.3.21, 5.3.3, 5.3.7 and 5.3.8 into the following conclusion, obtaining thus new characterizations of syntax-directed translations.

Corollary 5.3.9. *The following hold:*

$$\mathcal{SDT} = \lambda[\text{ln-XTT}] = \lambda[GSTSG] = \lambda[STSG] = \lambda[SDTS] = \lambda[SCFG] \ .$$

Very often tree transducers and synchronous grammars are connected via tree bimorphisms, the formalism that we study in detail in the following chapter, in an attempt to improve the mathematical foundations of both devices and subsequently, their relevance in practice.

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

CHAPTER 6

Tree Bimorphisms and Translations

As we already noted in Section 3.2, Nivat (1968) has shown that a translation is regular if and only if it can be represented as a composition of an inverse string homomorphism, the intersection with a regular language, and a string homomorphism. This fact yields a powerful algebraic tool for the study of regular translations - the string bimorphisms, and most of the properties of regular translations (see Theorem 3.2.4) could be proved using this characterization (cf. Berstel, 1979). Soon afterwards the tree bimorphism emerged as a natural counterpart to this notion in tree language theory being formed by two tree homomorphisms and a recognizable tree language. These general tree bimorphisms are quite powerful, but by suitable restrictions on its constituents one can get tree transformations with useful properties such as preservation of recognizability and closure under composition and which thus correspond to some extent to the regular translations.

It is the aim of this chapter to present in detail some of these classes with 'regular-like' properties together with their connection with synchronous grammars and tree transducers: *quasi-alphabetic tree transformations and translations* of Steinby and Tîrnăucă (2007, 2009) in Section 6.2, *primitive transformations* and *primitive transformations with permutation* of Takahashi (1972) in Section 6.3, *linear complete bimorphisms* of Arnold and Dauchet (1976a, 1982) in Section 6.4, and *alphabetic tree relations* of Bozapalidis (1992) in Section 6.5. Moreover, the HASSE diagram of Figure 6.6.1 shows for the first time the inclusion relations between these classes on tree and string level. Furthermore, Section 6.7 presents a quite complete and up-to-date overview of other less-known classes of tree transformations defined by means of tree bimorphisms that share appealing properties. We bring them into the spotlight of new applications by showing their connection with well-established synchronous grammars studied in computational linguistics community. Raoult (1992) and Dauchet and Tison (1992) briefly surveyed a few results from the 1970s and 1980s concerning tree bimorphisms. On the other hand, Tîrnăucă (2008) presented an informal overview on several relations between synchronous grammars and tree transducers via tree bimorphisms.

6.1 Basic notions

We start the presentation of tree bimorphisms with their formal definition.

Definition 6.1.1. A *tree bimorphism* is a triple $TB = (\varphi, R, \psi)$ that consists of

- a tree language $R \subseteq T_\Gamma(Z)$ called the *center*,
- an *input tree homomorphism* $\varphi\colon T_\Gamma(Z) \to T_\Sigma(X)$, and
- an *output tree homomorphism* $\psi\colon T_\Gamma(Z) \to T_\Omega(Y)$.

The *tree transformation defined by TB* is the relation

$$\tau(TB) := \varphi^{-1} \circ \mathrm{id}_R \circ \psi = \{(r\varphi, r\psi) \mid r \in R\} \, (\subseteq T_\Sigma(X) \times T_\Omega(Y)) \ ,$$

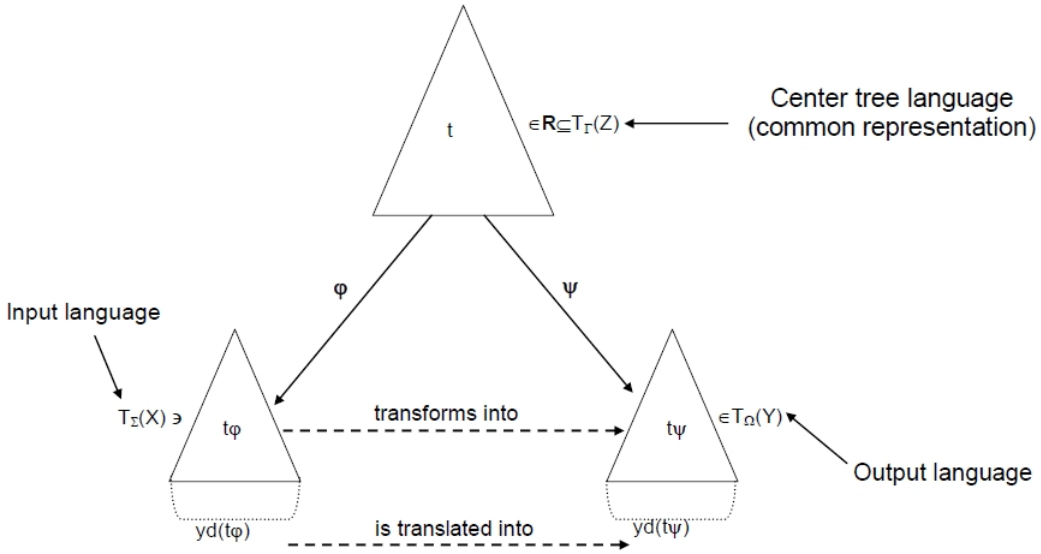


Figure 6.1.1: A pictorial representation of a tree bimorphism (φ, R, ϕ).

and the *translation defined by TB* is the relation

$$\lambda(TB) := \mathrm{yd}(\tau(TB)) = \{(\mathrm{yd}(r\varphi), \mathrm{yd}(r\psi)) \mid r \in R\} \ (\subseteq X^* \times Y^*) \ .$$

Moreover, we say that a tree bimorphism (φ, R, ψ) is *variable-free* (respectively, *almost variable-free*) if $R \subseteq T_\Gamma$ (respectively, $R \subseteq T_\Gamma \cup Z$). □

A pictorial representation of a tree bimorphism is shown in Figure 6.1.

For any classes $\mathcal{H}_1$ and $\mathcal{H}_2$ of tree homomorphisms and any class Fam of tree languages, we denote by $\mathbf{B}[\mathcal{H}_1, \mathrm{Fam}, \mathcal{H}_2]$ the class of all tree bimorphisms $TB = (\varphi, R, \psi)$ with $\varphi \in \mathcal{H}_1$, $R \in \mathrm{Fam}$ and $\psi \in \mathcal{H}_2$, and the corresponding classes of tree transformations and translations are denoted by $\tau[\mathcal{H}_1, \mathrm{Fam}, \mathcal{H}_2]$ and $\lambda[\mathcal{H}_1, \mathrm{Fam}, \mathcal{H}_2]$, respectively.

To clarify the notions introduced so far, we shall give an example.

Example 6.1.2. Let $\Sigma = \{f/3, g/1\}$, $\Omega = \{h/1\}$, $X = \{x\}$, and $Y = \{1\}$. Moreover, let

$$R = \{f(g^l(x), g^m(x), g^n(x)) \mid l, m, n \in \mathbb{N}\} \subseteq T_\Sigma(X) \ ,$$

and let $\varphi\colon T_\Sigma(X) \to T_\Sigma(X)$ and $\psi\colon T_\Sigma(X) \to T_\Omega(Y)$ be the tree homomorphisms given by

$$\begin{array}{lll} \varphi_3(f) := f(\xi_3, \xi_2, \xi_1) & \varphi_1(g) := g(\xi_1) & \varphi_X(x) := x \\ \psi_3(f) := \xi_3 & \psi_1(g) := h(\xi_1) & \psi_X(x) := 1 \ . \end{array}$$

Then the tree transformation defined by $TB = (\varphi, R, \psi)$ is

$$\tau(TB) = \{(f(g^n(x), g^m(x), g^l(x)),\ h^n(1)) \mid l, m, n \in \mathbb{N}\} \ \in \tau[\mathrm{pH}, \mathrm{Rec}, \mathrm{fH}] \ .$$

Moreover, the translation defined by TB is $\lambda(TB) = \{(xxx, 1)\}$. □

Note that by definition, a class of tree transformations defined by a tree bimorphism is naturally decomposed into three classes of tree transformations computable by classical tree transducers and easy to handle: HOM^{-1}, FTA and HOM. The decomposition results of Engelfriet (1975a, c) (cf. also Theorem 5.2.16) showed that all TOP- and BOT-tree transformations can be defined by such tree bimorphisms, but this did not lead to any useful characterizations of these classes of tree transformations. Moreover, the family of tree transformations defined by these tree bimorphisms enjoy few of the good properties demanded by practice (see Table 5.2.1) and shared by the original string bimorphisms of Nivat (1968) (cf. Theorem 3.2.4). Therefore, attention has turned to special classes of tree bimorphisms which are obtained by imposing some restrictions on the center tree language or the tree homomorphisms (cf. Takahashi, 1972, Arnold and Dauchet, 1976a, 1982, Bozapalidis, 1992, Steinby and Tîrnăucă, 2007, 2009, for example).

Definition 6.1.3. A tree bimorphism $TB = (\varphi, R, \psi)$ is said to be

- *alphabetic* if $R \in \mathrm{Rec}$ and $\varphi, \psi \in \mathrm{aH}$;
- *permuting* if $R \in \mathrm{Rec}$ and $\varphi, \psi \in \mathrm{pH}$;
- *quasi-alphabetic* if $R \in \mathrm{Loc}$ and $\varphi, \psi \in \mathrm{qH}$;
- *linear non-deleting* if $R \in \mathrm{Rec}$ and $\varphi, \psi \in \mathrm{lnH}$;
- *fine* if $R \in \mathrm{Rec}$ and $\varphi, \psi \in \mathrm{fH}$. □

These tree bimorphisms are studied next (cf. Raoult, 1992, Dauchet and Tison, 1992, Tîrnăucă, 2008, Maletti, 2010a, for briefer expositions).

6.2 Quasi-alphabetic tree bimorphisms

Quasi-alphabetic tree bimorphisms were introduced by Steinby and Tîrnăucă (2007, 2009) as the natural counterpart of SDTSs, and their formal properties were further investigated by Maletti and Tîrnăucă (2010, 2009). In Section 6.2.1, we exhibit the properties of quasi-alphabetic tree bimorphisms that make them appealing for certain applications in machine translation (see Chiang and Knight, 2006, Yamada and Knight, 2001, Knight, 2007, Chiang, 2007, 2006, for example): preservation of recognizability of tree languages and closure under inverses, union and composition. We continue in Section 6.2.2 by showing that quasi-alphabetic tree bimorphisms and SDTSs are equally powerful as translation defining devices. Also, a formal definition of tree transformations definable by SDTSs and SCFGs is given for the first time (cf. Tîrnăucă, 2011). Moreover, we present subclasses of quasi-alphabetic tree bimorphisms that define regular translations and simple syntax-directed translations. This way, we may now also use tree language theory for proving properties of regular translations and syntax-directed translations as we show in a couple of examples. Finally, in Section 6.2.3, we place quasi-alphabetic tree transformations into the tree transducer hierarchy of Figures 5.2.2 and 5.2.3 (cf. Tîrnăucă, 2009), thus making quasi-alphabetic tree bimorphisms easy to implement in TIBURON (May and Knight, 2006, May, 2010).

We start by fixing some notation and terminology.

Definition 6.2.1. The members of $\tau[\mathrm{qH}, \mathrm{Loc}, \mathrm{qH}]$ are called *quasi-alphabetic tree transformations*. Similarly, we refer to the elements of $\lambda[\mathrm{qH}, \mathrm{Loc}, \mathrm{qH}]$ as the *quasi-alphabetic*

translations. Let us denote the classes $\tau[\mathrm{qH}, \mathrm{Loc}, \mathrm{qH}]$ and $\lambda[\mathrm{qH}, \mathrm{Loc}, \mathrm{qH}]$ simply by $\tau[qTB]$ and $\lambda[qTB]$, respectively. Similarly, $\tau[\mathrm{qH}, \mathrm{Loc}^{\mathrm{vf}}, \mathrm{qH}]$ and $\lambda[\mathrm{qH}, \mathrm{Loc}^{\mathrm{vf}}, \mathrm{qH}]$ are generically denoted $\tau[qTB^{\mathrm{vf}}]$ and $\lambda[qTB^{\mathrm{vf}}]$, respectively. □

Since the composition of an alphabetic tree homomorphism and a quasi-alphabetic tree homomorphism obviously is a quasi-alphabetic tree homomorphism, the following facts easily follow from Proposition 4.5.5 (cf. Steinby and Tîrnăucă, 2009, Theorems 6.2 and 6.3).

Proposition 6.2.2. *The following hold.*

(i) $\tau[\mathrm{qH}, \mathrm{Loc} \cap \mathrm{DRec}, \mathrm{qH}] = \tau[\mathrm{qH}, \mathrm{Loc}, \mathrm{qH}] = \tau[\mathrm{qH}, \mathrm{Rec}, \mathrm{qH}]$.

(ii) $\lambda[\mathrm{qH}, \mathrm{Loc} \cap \mathrm{DRec}, \mathrm{qH}] = \lambda[\mathrm{qH}, \mathrm{Loc}, \mathrm{qH}] = \lambda[\mathrm{qH}, \mathrm{Rec}, \mathrm{qH}]$.

(iii) $\tau[\mathrm{qH}, \mathrm{Loc} \cap \mathrm{DRec}^{\mathrm{vf}}, \mathrm{qH}] = \tau[\mathrm{qH}, \mathrm{Loc}^{\mathrm{vf}}, \mathrm{qH}] = \tau[\mathrm{qH}, \mathrm{Rec}^{\mathrm{vf}}, \mathrm{qH}]$.

(iv) $\lambda[\mathrm{qH}, \mathrm{Loc} \cap \mathrm{DRec}^{\mathrm{vf}}, \mathrm{qH}] = \lambda[\mathrm{qH}, \mathrm{Loc}^{\mathrm{vf}}, \mathrm{qH}] = \lambda[\mathrm{qH}, \mathrm{Rec}^{\mathrm{vf}}, \mathrm{qH}]$.

Hence, when considering quasi-alphabetic tree transformations or translations, one may either assume that the defining tree bimorphisms belong to $\mathbf{B}[\mathrm{qH}, \mathrm{Loc} \cap \mathrm{DRec}, \mathrm{qH}]$ or allow a more general tree bimorphism in $\mathbf{B}[\mathrm{qH}, \mathrm{Rec}, \mathrm{qH}]$. The same assumption can be made for tree transformations and translations defined by variable-free quasi-alphabetic tree bimorphisms.

Moreover, we can show that almost variable-free quasi-alphabetic tree bimorphisms define exactly the quasi-alphabetic tree transformations. To eliminate variables is essential to our approach to closure under composition of $\tau[qTB]$ that is shown in Section 6.2.1.

Lemma 6.2.3. $\tau[\mathrm{qH}, \mathrm{Rec}^{\mathrm{avf}}, \mathrm{qH}] = \tau[qTB]$.

Proof. Every tree transformation defined by an almost variable-free quasi-alphabetic tree bimorphism is obviously quasi-alphabetic. To prove the converse inclusion, let $TB = (\varphi, R, \psi)$ be a quasi-alphabetic tree bimorphism with $R \subseteq T_\Gamma(Z)$, $\varphi\colon T_\Gamma(Z) \to T_\Sigma(X)$ and $\psi\colon T_\Gamma(Z) \to T_\Omega(Y)$. Moreover, let V be an alphabet and $\phi\colon Z \to V$ a bijection. Finally, let $BU = (Q, \Gamma', \emptyset, \Omega', \emptyset, P, Q)$ be the linear BOT-transducer with

- $Q := V \cup \{\star\}$.
- $\Gamma'_m := \Gamma_m$ for every $m \geq 1$ and $\Gamma'_0 := \Gamma_0 \cup Z$.
- $\Omega'_m := \{t \in \Gamma(Q) \mid m = |t|_\star\}$ for every $m \geq 1$, and let $\Omega'_0 := \Gamma_0 \cup Z$.
- The set P of rules is given as follows:
  - For every $z \in Z$, let $z \to q(z)$ be a rule of P where $q = \phi(z)$.
  - For every $m \geq 1$, $f \in \Gamma_m$ and $q_1, \ldots, q_m \in Q$, let

$$f(q_1(\xi_1), \ldots, q_m(\xi_m)) \to \star(f(q_1, \ldots, q_m)(\xi_{i_1}, \ldots, \xi_{i_n}))$$

    be a rule of P, where $i_1 < \ldots < i_n$ and $\{i_1, \ldots, i_n\} = \{i \in [m] \mid q_i = \star\}$.

Let $R' := \tau(BU)(R)$ be the image of R under τ_M. Clearly, R' is almost variable-free, and, by Table 5.2.1, it is recognizable. We construct the tree bimorphism $TB' = (\varphi', R', \psi')$ such that $\varphi'_Z := \varphi_Z$, $\psi'_Z := \psi_Z$, and

$$\varphi'_m(t) := \varphi(t[\star \leftarrow (\xi_1, \ldots, \xi_m)]) \quad \text{and} \quad \psi'_m(t) := \psi(t[\star \leftarrow (\xi_1, \ldots, \xi_m)])$$

for every $t \in \Omega_m$ ($m \geq 0$). Clearly, φ' and ψ' are quasi-alphabetic. Thus, TB' is an almost variable-free quasi-alphabetic tree bimorphism. Note that BU is deterministic and total and thus $\tau(BU)$ is a mapping (Engelfriet, 1975a, Fülöp, 2004). Finally, let us prove that $\tau(TB') = \tau(TB)$. For this, we show that $t\varphi = t\tau(BU)\varphi'$ and $t\psi = t\tau(BU)\psi'$ for every $t \in T_\Gamma(Z)$. Obviously, it is sufficient to prove the former statement since the argument is totally symmetric. We proceed by tree induction on $t \in T_\Gamma(Z)$.

(1) Let $t \in Z$. Then clearly $t\varphi = t\tau(BU)\varphi = t\tau(BU)\varphi'$.

(2) Let $t = f(t_1, \ldots, t_m)$ for some $m \geq 1$, $f \in \Sigma_m$ and $t_1, \ldots, t_m \in T_\Gamma(Z)$. Moreover, for every $i \in [m]$, let $q_i := \phi(t_i)$ if $t_i \in Z$ and $q_i := \star$ otherwise. Then

$$\begin{aligned} f(t_1, \ldots, t_m)\tau(BU)\varphi' &= f(q_1, \ldots, q_m)(t_{i_1}\tau(BU), \ldots, t_{i_n}\tau(BU))\varphi' \\ &= \varphi'_n(f(q_1, \ldots, q_m))[t_{i_1}\tau(BU)\varphi', \ldots, t_{i_n}\tau(BU)\varphi'] \\ &= \varphi(f(q_1, \ldots, q_m)[\star \leftarrow (x_1, \ldots, x_n)])[t_{i_1}\varphi, \ldots, t_{i_n}\varphi] \\ &= f(q_1, \ldots, q_m)[\star \leftarrow (t_{i_1}, \ldots, t_{i_n})]\varphi \\ &= f(t_1, \ldots, t_m)\varphi = t\varphi \end{aligned}$$

with $i_1 < \ldots < i_n$ and $\{i_1, \ldots, i_n\} = \{i \in [m] \mid q_i = \star\}$.

This completes the proof. □

Next, we shall give an example of a quasi-alphabetic tree bimorphism.

Example 6.2.4. Let $\Gamma = \{f/3, g/1, e/0\}$, $\Sigma = \{h/6, g/1\}$, $\Omega = \{f/3, g/1\}$, $X = \{x\}$, and $Y = \{0, 1\}$. Consider the recognizable tree language

$$R = \{f(g^m(x), e, g^n(x)) \mid m, n \in \mathbb{N}\} \subseteq T_\Gamma(X)$$

and the quasi-alphabetic tree homomorphisms $\varphi\colon T_\Gamma(X) \to T_\Sigma(X)$ and $\psi\colon T_\Gamma(X) \to T_\Omega(Y)$ given by

$$\begin{array}{llll} \varphi_3(f) := h(x, \xi_2, \xi_3, x, x, \xi_1) & \varphi_1(g) := g(\xi_1) & \varphi_0(e) := x & \varphi_X(x) := x \\ \psi_3(f) := f(\xi_1, \xi_2, \xi_3) & \psi_1(g) := g(\xi_1) & \psi_0(e) := f(1, 1, 0) & \psi_X(x) := 1 \ . \end{array}$$

Then, $TB = (\varphi, R, \psi) \in \mathbf{B}[\mathrm{qH}, \mathrm{Rec}, \mathrm{qH}]$. The quasi-alphabetic tree transformation defined by TB is

$$\tau(TB) = \{\big(h(x, x, g^n(x), x, x, g^m(x)),\ f(g^m(1), f(1, 1, 0), g^n(1))\big) \mid m, n \in \mathbb{N}\} \ ,$$

and the quasi-alphabetic translation defined by TB is $\lambda(TB) = \{(xxxxxx,\ 11101)\}$. □

The following proposition is trivial, but indicates why closure under composition (Theorem 6.2.17) is possible whereas closure under intersection (Theorem 6.2.12) fails.

Proposition 6.2.5. *For every quasi-alphabetic tree bimorphism TB, there exist a quasi-alphabetic tree bimorphism TB_1 with a normalized input tree homomorphism and a quasi-alphabetic tree bimorphism TB_2 with a normalized output tree homomorphism such that $\tau(TB) = \tau(TB_1) = \tau(TB_2)$. If TB is variable-free (almost variable-free, respectively), then TB_1 and TB_2 can be chosen such that they are variable-free (almost variable-free, respectively).*

6.2.1 Properties

Now we exhibit the fundamental properties of quasi-alphabetic tree transformations studied by Steinby and Tîrnăucă (2007, 2009) and Maletti and Tîrnăucă (2009, 2010). As shown in the next section, this improves the mathematical foundations of the well-known syntax-directed translations and their specification methods. We start by elegantly proving the recognizability of tree languages by the quasi-alphabetic tree bimorphisms.

Theorem 6.2.6. *If* $TB \in \mathbf{B}[\mathrm{qH}, \mathrm{Rec}, \mathrm{qH}]$ *and* $R \in \mathrm{Rec}$*, then* $R\tau(TB) \in \mathrm{Rec}$.

Proof. If $TB = (\varphi, R', \psi)$ is a quasi-alphabetic tree bimorphism and R is any recognizable tree language, then $R\tau(TB) = (R\varphi^{-1} \cap R')\psi$ is also recognizable because recognizable tree languages are closed under inverse tree homomorphisms, intersection and linear tree homomorphisms by Theorem 4.4.17(i-iii). □

When quasi-alphabetic tree transformations are defined by means of bimorphisms, it seems natural to prove decidability results for them such as the membership, the emptiness and the finiteness, as shown below (as it was done by Steinby, 1984, for other tree bimorphisms and their tree transformation classes).

Theorem 6.2.7. *Suppose* $\tau \subseteq T_\Sigma(X) \times T_\Omega(Y)$ *is a quasi-alphabetic tree transformation and let* $(s,t) \in T_\Sigma(X) \times T_\Omega(Y)$*. Each of the following questions is decidable:*

(i) *Is* (s,t) *in* τ*?*

(ii) *Is* τ *empty?*

(iii) *Is* τ *finite?*

Proof. Assume $\tau = \tau(TB)$ for some quasi-alphabetic tree bimorphism $TB = (\varphi, R, \psi)$. A quasi-alphabetic tree homomorphism never decreases the height of a tree by Proposition 4.2.3(i), and hence, by considering a finite number of trees $r \in R$, one can decide whether there exists such a tree r in R that $r\varphi = s$ and $r\psi = t$. So, the membership problem is decidable. The decidability of questions (b) and (c) follows from the fact that τ is empty, finite or infinite depending on whether R is empty, finite or infinite, and that these questions are decidable for recognizable tree languages by Theorem4.4.17(vi). □

Another useful immediate consequence of defining quasi-alphabetic tree transformations by tree bimorphisms is that every $\tau \in \tau[qTB]$ with $\tau \subseteq T_\Sigma(X) \times T_\Omega(Y)$ is *symmetrically locally finite* (cf. Elgot and Mezei, 1965), i.e., for any $s \in T_\Sigma(X)$ and any $t \in T_\Omega(Y)$, the set $s\tau$ of transforms of s and the pre-image $t\tau^{-1}$ of t are both finite. This is so because a quasi-alphabetic tree homomorphisms never reduces the height of a tree by Proposition 4.2.3(i).

The local finiteness of quasi-alphabetic tree transformations implies that $T_\Sigma(X) \times T_\Omega(Y) \setminus \tau$ never is finite or a quasi-alphabetic tree transformation τ unless the ranked alphabets have only nullary symbols. Therefore, the questions "Is $T_\Sigma(X) \times T_\Omega(Y) \setminus \tau$ in $\tau[qTB]$?" and "Is $\tau = T_\Sigma(X) \times T_\Omega(Y)$?" are trivial for a tree transformation $\tau \in \tau[qTB]$.

We continue by presenting a *canonical representation* of quasi-alphabetic tree transformations in the spirit of Bozapalidis (1992, Proposition 3.1). Note that our product alphabet is simpler than the corresponding one of Bozapalidis (1992) (see also Definition 6.5.2). Such a representation will allow us to conclude that quasi-alphabetic tree transformations are

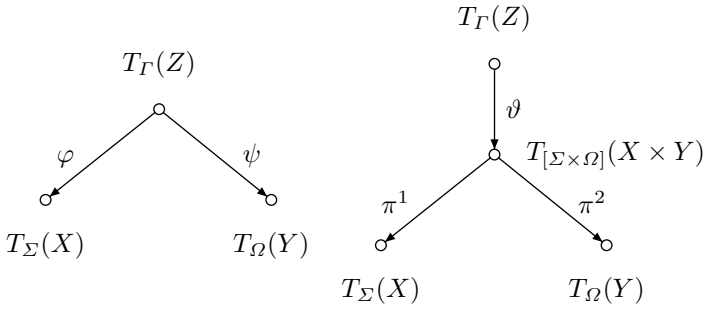


Figure 6.2.1: Illustration of the construction in Theorem 6.2.9.

closed under union. Also, it can be used to prove that various classes of tree transformations defined by tree bimorphisms are computed by certain tree transducers (cf. Rahonis, 2001, Tîrnăucă, 2009, for example).

For the rest of this section, let $TB = (\varphi, R, \psi)$ be a quasi-alphabetic bimorphism with $\varphi\colon T_\Gamma(Z) \to T_\Sigma(X)$ and $\psi\colon T_\Gamma(Z) \to T_\Omega(Y)$. Let $[\Sigma \times \Omega]$ be the ranked alphabet such that for every $m \geq 0$

$$[\Sigma \times \Omega]_m := \{\langle s, t\rangle \mid s \in \Sigma(X \cup \Xi_m) \cap C^m_\Sigma(X) \text{ and } t \in \Omega(Y \cup \Xi_m) \cap C^m_\Omega(Y)\} \ .$$

In other words, this product alphabet allows us to store $\langle \varphi_m(f), \psi_m(f)\rangle$ for any $f \in \Gamma_m$ $(m \geq 0)$. Clearly, there are canonical quasi-alphabetic tree homomorphisms

$$\pi^1\colon T_{[\Sigma\times\Omega]}(X \times Y) \to T_\Sigma(X) \qquad \text{and} \qquad \pi^2\colon T_{[\Sigma\times\Omega]}(X \times Y) \to T_\Omega(Y)$$

given by

$$\pi^1_{X\times Y}(\langle x, y\rangle) := x \quad \text{and} \quad \pi^2_{X\times Y}(\langle x, y\rangle) := y \quad \text{for } \langle x, y\rangle \in X \times Y \ ,$$

and

$$\pi^1_m(\langle s, t\rangle) := s \quad \text{and} \quad \pi^2_m(\langle s, t\rangle) := t \quad \text{for } m \geq 0 \text{ and } \langle s, t\rangle \in [\Sigma \times \Omega]_m \ .$$

Henceforth, we will use these projections also for other product ranked alphabets.

Proposition 6.2.8. *There exists a quasi-alphabetic tree homomorphism*

$$\vartheta\colon T_\Gamma(Z) \to T_{[\Sigma\times\Omega]}(X \times Y)$$

such that $t\varphi = (t\vartheta)\pi^1$ *and* $t\psi = (t\vartheta)\pi^2$ *for every* $t \in T_\Gamma(Z)$.

Proof. Let $\vartheta\colon T_\Gamma(Z) \to T_{[\Sigma\times\Omega]}(X \times Y)$ be the tree homomorphism such that $\vartheta_Z(z) := \langle \varphi_Z(z), \psi_Z(z)\rangle$ for every $z \in Z$, and $\vartheta_m(f) := \langle \varphi_m(f), \psi_m(f)\rangle$ for every $f \in \Gamma_m$ with $m \geq 0$. Clearly, ϑ is quasi-alphabetic, and it is easy to check by tree induction that $t\varphi = (t\vartheta)\pi^1$ and $t\psi = (t\vartheta)\pi^2$ for every $t \in T_\Gamma(Z)$. □

Using the previous proposition, we can now eliminate the ranked alphabet Γ, the leaf alphabet Z, and the particular tree homomorphisms φ and ψ from the tree bimorphism TB. Essentially, every quasi-alphabetic tree transformation $\tau \subseteq T_\Sigma(X) \times T_\Omega(Y)$ is determined

by a recognizable language $R \in \mathrm{Rec}_{[\Sigma\times\Omega]}(X \times Y)$. The construction is illustrated in Figure 6.2.1.

Theorem 6.2.9. *A relation $\tau \subseteq T_\Sigma(X) \times T_\Omega(Y)$ is a quasi-alphabetic tree transformation if and only if there exists $R \subseteq \mathrm{Rec}_{[\Sigma\times\Omega]}(X \times Y)$ such that $\tau = \{(t\pi^1, t\pi^2) \mid t \in R\}$.*

Proof. The if-direction is trivial because (π^1, R, π^2) is a quasi-alphabetic bimorphism defining τ. For the converse, let $TB = (\varphi, R', \psi)$ be a quasi-alphabetic tree bimorphism such that $\tau(TB) = \tau$. By Proposition 6.2.8 there exists a quasi-alphabetic tree homomorphism $\vartheta\colon T_\Gamma(Z) \to T_{[\Sigma\times\Omega]}(X \times Y)$ such that $\tau = \{(t\vartheta\pi^1, t\vartheta\pi^2) \mid t \in R'\}$. Thus, $R = \vartheta(R')$ has the desired properties because it is recognizable by Theorem 4.4.17(iii). □

Now, we use Theorem 6.2.9 to prove that quasi-alphabetic tree transformations are closed under union (cf. Maletti and Tîrnăucă, 2010, Corollary 10).

Corollary 6.2.10. *The class $\tau[qTB]$ is closed under union.a*

Proof. Let $\tau_1, \tau_2 \subseteq T_\Sigma(X) \times T_\Omega(Y)$ be quasi-alphabetic tree transformations. By Theorem 6.2.9, there exist $R_1, R_2 \subseteq \mathrm{Rec}_{[\Sigma\times\Omega]}(X \times Y)$ such that

$$\tau_1 = \{(t\pi^1, t\pi^2) \mid t \in R_1\} \quad \text{and} \quad \tau_2 = \{(t\pi^1, t\pi^2) \mid t \in R_2\} \ .$$

Then

$$\tau_1 \cup \tau_2 = \{(t\pi^1, t\pi^2) \mid t \in R_1\} \cup \{(t\pi^1, t\pi^2) \mid t \in R_2\} = \{(t\pi^1, t\pi^2) \mid t \in R_1 \cup R_2\} \ ,$$

which proves that $\tau_1 \cup \tau_2$ is a quasi-alphabetic tree transformation by Theorem 6.2.9 (because $R_1 \cup R_2$ is recognizable by item (i) of Theorem 4.4.17). □

Note that there is a simple direct proof of this fact as shown next. However, we prefer to also give an alternative proof using the canonical representation of quasi-alphabetic tree transformations just to demonstrate how such a result can thus be utilized (cf. also Bozapalidis, 1992, Bozapalidis and Rahonis, 1994, Rahonis, 2001, Takahashi, 1972, for example).

Let $\tau_1, \tau_2 \subseteq T_\Sigma(X) \times T_\Omega(Y)$ be two quasi-alphabetic tree transformations. Without any loss of generality, we may assume that τ_1 and τ_2 are defined by the quasi-alphabetic tree bimorphisms $TB_1 = (\varphi, R, \psi)$ and $TB_2 = (\varphi', R', \psi')$, respectively, where $R \in \mathrm{Rec}_\Gamma(Z)$ and $R' \in \mathrm{Rec}_{\Gamma'}(Z')$ are such that $\Gamma \cap \Gamma' = \emptyset$ and $Z \cap Z' = \emptyset$. Moreover, $\varphi\colon T_\Gamma(Z) \to T_\Sigma(X)$ and $\varphi'\colon T_{\Gamma'}(Z') \to T_\Sigma(X)$, and we define $\bar{\varphi}\colon T_{\Gamma\cup\Gamma'}(Z \cup Z') \to T_\Sigma(X)$ such that

- for every $z \in Z \cup Z'$, $\bar{\varphi}_{Z\cup Z'}(z) := \begin{cases} \varphi_Z(z) & \text{if } z \in Z \\ \varphi_{Z'}(z) & \text{if } z \in Z' \end{cases}$,
- for every $m \geq 0$ and $f \in (\Gamma \cup \Gamma')_m$, $\bar{\varphi}_m(f) := \begin{cases} \varphi_m(f) & \text{if } f \in \Gamma_m \\ \varphi'_m(f) & \text{if } f \in \Gamma'_m \end{cases}$.

Similarly, $\psi\colon T_\Gamma(Z) \to T_\Omega(Y)$ and $\psi'\colon T_{\Gamma'}(Z') \to T_\Omega(Y)$, and we define the tree homomorphism $\bar{\psi}\colon T_{\Gamma\cup\Gamma'}(Z \cup Z') \to T_\Omega(Y)$ by

- for every $z \in Z \cup Z'$, $\bar{\psi}_{Z\cup Z'}(z) := \begin{cases} \psi_Z(z) & \text{if } z \in Z \\ \psi_{Z'}(z) & \text{if } z \in Z' \end{cases}$,

- for every $m \geq 0$ and $f \in (\Gamma \cup \Gamma')_m$, $\bar{\psi}_m(f) := \begin{cases} \psi_m(f) & \text{if } f \in \Gamma_m \\ \psi'_m(f) & \text{if } f \in \Gamma'_m \end{cases}$.

Clearly, $TB = (\bar{\varphi}, R \cup R', \bar{\psi})$ is a quasi-alphabetic tree bimorphism and $\tau(TB) = \tau_1 \cup \tau_2$. Therefore, $\tau[qTB]$ is closed under union.

For any tree bimorphism $TB = (\varphi, R, \psi)$, the tree bimorphism (ψ, R, φ) defines the inverse of $\tau(TB)$. Hence, the following fact is obvious.

Proposition 6.2.11. *The class $\tau[qTB]$ is closed under inverses.* □

Let us move on to closure under intersection. For it we would need to align the two input tree homomorphisms and the two output tree homomorphisms at the same time and enforce equality both-sided. The next theorem shows that we are not able to achieve this for quasi-alphabetic tree transformations.

Theorem 6.2.12. *No class $\mathcal{C}$ of tree tree transformations such that*

$$\mathrm{pH} \subseteq \mathcal{C} \subseteq \tau[\mathrm{H}, \mathrm{Rec}, \mathrm{lH}]$$

is closed under intersection.

Proof. Let $\Sigma = \{f/2, g/1, e/0\}$, and let $\psi, \psi' \colon T_\Sigma \to T_\Sigma$ be the permuting tree homomorphisms defined by

$$\begin{array}{lll} \psi_2(f) := f(\xi_1, \xi_2) & \psi_1(g) := g(\xi_1) & \psi_0(e) := e \\ \psi'_2(f) := f(\xi_2, \xi_1) & \psi'_1(g) := g(\xi_1) & \psi'_0(e) := e \ . \end{array}$$

Clearly, ψ and ψ' belong to $\mathcal{C}$. Note that $\psi = \mathrm{id}_{T_\Sigma}$. We observe that for every $m, n \in \mathbb{N}$

$$\begin{aligned} f(g^m(e), g^n(e))\psi &= f(g^m(e), g^n(e)) \\ f(g^m(e), g^n(e))\psi' &= f(g^n(e), g^m(e)) \ . \end{aligned}$$

Let $R := \{f(g^m(e), g^n(e)) \mid m, n \in \mathbb{N}\}$. Clearly, R is a recognizable tree language. Assume that there exists $\tau \in \tau[\mathrm{H}, \mathrm{Rec}, \mathrm{lH}]$ such that $\tau = \psi \cap \psi'$. Since τ preserves recognizable tree languages by Theorem 4.4.17(iii), the image $R\tau$ should be recognizable, but

$$R\tau = \{f(g^n(e), g^n(e)) \mid n \in \mathbb{N}\}$$

is known not to be recognizable by Theorem 4.4.16 (cf. also Gécseg and Steinby, 1984, Gécseg and Steinby, 1997). Hence no τ with the given properties exists, which proves the statement. □

Because obviously $\mathrm{pH} \subseteq \tau[qTB]$, we get.

Corollary 6.2.13. *The class $\tau[qTB]$ is not closed under intersection.* □

Using also Corollary 6.2.10, we note.

Corollary 6.2.14. *The class $\tau[qTB]$ is not closed under complement.* □

Now let us consider some common relations on trees of Section 4.3. Let $R \in \mathrm{Rec}_\Sigma(X)$, and let $\tau \subseteq T_\Sigma(X) \times T_\Sigma(X)$ be a quasi-alphabetic tree transformation. Moreover, τ is defined by the quasi-alphabetic tree bimorphism (φ, R', ψ), where $R' \in \mathrm{Rec}_\Sigma(X)$ and $\varphi(t) := t$ for every $t \in T_\Sigma(X)$. Then the intersection of a quasi-alphabetic tree transformation with id_R is a quasi-alphabetic tree transformation, because $(\varphi, R \cap R', \psi)$ is a quasi-alphabetic tree bimorphism defining $\tau \cap \mathrm{id}_R$. Also the union with id_R is a quasi-alphabetic tree transformation because id_R is a quasi-alphabetic tree transformation and quasi-alphabetic tree transformations are closed under union by Corollary 6.2.10.

In general, the tree transformations 'sub' and 'br' (if we consider the branches as trees over a ranked alphabet of symbols of ranks 0 and 1) are not quasi-alphabetic. Moreover, for $R \subseteq \mathrm{Rec}_\Sigma(X)$ and $x \in X$, also the following relations τ and τ', which are defined for every $t \in T_\Sigma(X)$ by $t\tau := t \bullet_x R$ and $t\tau := t /_x R$, are not quasi-alphabetic tree transformations in general (in contrast to Bozapalidis, 1992, see Proposition 4.2 & pp. 191–200, where it is shown that all those tree transformations are defined by fine tree bimorphisms). All these facts can easily be proved using Proposition 4.2.3. Moreover, in general, quasi-alphabetic tree transformations are not closed under f-concatenation (again in contrast to Bozapalidis, 1992, Proposition 3.6).

Now we turn our attention to one of the most interesting theoretical properties of classes of tree transformations: closure under composition. It was shown by Steinby and Tîrnăucă (2007, Theorem 5) or Steinby and Tîrnăucă (2009, Theorem 7.4) that if we restrict ourselves to quasi-alphabetic tree bimorphisms with a variable-free center tree language, then the resulting class of tree transformations (see also Proposition 6.2.2(iii)) is closed under composition. The method they employed is similar to the one of Nivat (1968) or Bozapalidis (1992), but here we extend this result to include variables by using a slightly different approach (cf. Maletti and Tîrnăucă, 2009). First, we point out why it is far easier to prove the closure only for tree transformations defined by variable-free or almost variable-free quasi-alphabetic tree bimorphisms.

Lemma 6.2.15. *Let $\varphi\colon T_\Sigma(X) \to T_\Gamma(Z)$ and $\psi\colon T_\Omega(Y) \to T_\Gamma(Z)$ be normalized quasi-alphabetic tree homomorphisms, and let $s \in T_\Sigma \cup X$ and $t \in T_\Omega \cup Y$. If $s\varphi = t\psi$, then* $\mathrm{dom}(s) = \mathrm{dom}(t)$.

Proof. Firstly, let $s \in X$. Then $s\varphi \in Z$. Since $s\varphi = t\psi$, it follows that $t \in Y$ and hence $\mathrm{dom}(s) = \mathrm{dom}(t)$. Secondly, let $s = f(s_1, \ldots, s_m)$ for some $f \in \Sigma_m$ and $s_1, \ldots, s_m \in T_\Sigma$. Then $s\varphi = \varphi_m(f)[s_1\varphi, \ldots, s_m\varphi] = t\psi$. Since φ is quasi-alphabetic and $s \in T_\Sigma$, we have $s_i\varphi \notin Z$ for every $i \in [m]$. If we additionally take into account that $s\varphi = t\psi$, then we can conclude that $t = g(t_1, \ldots, t_m)$ for some $g \in \Omega_m$ and $t_1, \ldots, t_m \in T_\Omega$. Moreover, since φ and ψ are normalized, it also follows that $\varphi_m(f) = \psi_m(g)$. Using the induction hypothesis, we thus obtain $\mathrm{dom}(s) = \mathrm{dom}(t)$. □

The previous lemma essentially states that all almost variable-free trees with the same image under two normalized quasi-alphabetic tree homomorphisms can be paired up in a product data structure $T_{\Sigma\times\Omega}(X \times Y)$. In addition, the following result would be useful.

Lemma 6.2.16. *Let $\varphi\colon T_\Omega(Z) \to T_\Sigma(X)$ and $\psi\colon T_\Omega(Z) \to T_\Sigma(X)$ be normalized quasi-alphabetic tree homomorphisms. Then $R = \{t \in T_\Omega \cup Z \mid t\varphi = t\psi\}$ is recognizable.*

Proof. We construct the regular tree grammar $TG = (\{S\}, \Omega', P, S)$, where

- $\Omega'_0 := \Omega_0 \cup Z$,

- $\Omega'_m := \Omega_m$ for every $m \geq 1$, and
- $P := P_1 \cup P_2$ with

$$P_1 = \{S \to z \mid z \in Z, \varphi_Z(z) = \psi_Z(z)\}, \text{ and}$$
$$P_2 = \{S \to f(S, \ldots, S) \mid f \in \Omega_m, \varphi_m(f) = \psi_m(f)\} \ .$$

Then $R = T(TG) \cap (T_\Omega \cup Z)$, which is recognizable by Theorem 4.4.17(i). □

Now we plug the statements of Proposition 6.2.5 and Lemmata 6.2.3, 6.2.15 and 6.2.16 together, and establish the relation to closure under composition (cf. Proposition 6.2.2(i)).

Theorem 6.2.17. *The class $\tau[qTB]$ is closed under composition.*

Proof. Let $TB = (\varphi, R, \psi)$ and $TB' = (\varphi', R', \psi')$ be quasi-alphabetic tree bimorphisms, where $R \subseteq T_\Gamma(Z)$, $\varphi\colon T_\Gamma(Z) \to T_\Sigma(X)$, $\psi\colon T_\Gamma(Z) \to T_\Omega(Y)$, $R' \subseteq T_\Lambda(W)$, $\varphi'\colon T_\Lambda(W) \to T_\Omega(Y)$ and $\psi'\colon T_\Lambda(W) \to T_\Delta(V)$. Without loss of generality, let TB and TB' be almost variable-free by Lemma 6.2.3. Moreover, suppose that ψ and φ' are normalized by Proposition 6.2.5. Let

$$\begin{aligned}\tau &:= \tau(TB) \circ \tau(TB') \\ &= \{(s,u) \mid \exists t\colon (s,t) \in \tau(TB), (t,u) \in \tau(TB')\} \\ &= \{(t\varphi, u\psi') \mid t \in R, u \in R', t\psi = u\varphi'\} \ .\end{aligned}$$

Since $t\psi = u\varphi'$, it follows by Lemma 6.2.15 that $\mathrm{dom}(t) = \mathrm{dom}(u)$. Hence there is a tree $s \in T_{\Gamma\times\Lambda} \cup (Z \times W)$ such that $s\pi_1 = t$ and $s\pi_2 = u$ where π_1 and π_2 are the usual projections to the first and second component.

Let $T := T_{\Gamma\times\Lambda} \cup (Z \times W)$. We can continue the displayed equations by

$$\begin{aligned}\tau &= \{(t\pi_1\varphi, t\pi_2\psi') \mid t \in T, t\pi_1 \in R, t\pi_2 \in R', t\pi_1\psi = t\pi_2\varphi'\} \\ &= \{(t\pi_1\varphi, t\pi_2\psi') \mid t \in \pi_1^{-1}(R) \cap \pi_2^{-1}(R') \cap R''\} \ ,\end{aligned}$$

where $R'' = \{t \in T \mid t\pi_1\psi = t\pi_2\varphi'\}$. It is easily seen that the tree homomorphisms $\pi_1\varphi_1$, $\pi_2\psi_2$, $\pi_1\psi_1$, and $\pi_2\varphi_2$ are quasi-alphabetic by Remark 4.2.4. Moreover, $\pi_1\psi$ and $\pi_2\varphi'$ are normalized. By assumption, R'' is thus recognizable, and $\pi_1^{-1}(R)$ and $\pi_2^{-1}(R')$ are recognizable by Theorem 4.4.17(iii). Consequently, $\pi_1^{-1}(R) \cap \pi_2^{-1}(R') \cap R''$ is recognizable by Theorem 4.4.17(i), and hence, $\tau \in \tau[qTB]$.

So whenever the set $\{t \in T_{\Gamma\times\Lambda} \cup (Z \times W) \mid t\psi = t\varphi'\}$ is recognizable, we can construct a quasi-alphabetic tree bimorphism that computes the composition of two given quasi-alphabetic tree bimorphisms as above. Now, using Lemma 6.2.16, we can complete the proof. □

6.2.2 Translation power

In this section a formal definition for tree transformations definable by SDTSs (and implicitly SCFGs) is given for the first time (cf. also Tîrnăucă, 2011). Moreover, it is shown how this formulation is supported by the relation on string level between syntax-directed translation defining devices and quasi-alphabetic tree bimorphisms: they define the same translations (Steinby and Tîrnăucă, 2007, 2009). Hence, the connections on string and tree

level between formalisms inspired by compiler design and linguistics – SDTSs and SCFGs, and an algebraic tool – the tree bimorphism, are established. This way, the mathematical foundations of such synchronous grammars are strengthen as promised in Section 3.3.3.

Let $SD = (N, X, Y, P, S)$ be any SDTS. Every derivation in SD can obviously be replaced with a leftmost one that yields the same result and in which exactly the same productions are used as in the original derivation. Hence, we may view leftmost derivations as normal forms of derivations, and we will represent them by production trees (cf. Section 4.6). These we define as follows.

First, we associate with SD a ranked alphabet Σ^{SD} such that for every $m \in \mathbb{N}$,

$$\Sigma_m^{SD} := \{[A; A \to \alpha; \beta\,(\sigma)] \mid A; A \to \alpha; \beta\,(\sigma) \in P, |\mathrm{pr}_N(\alpha)| = m\} \ .$$

Now, the sets $P(SD, A, A)$ of Σ^{SD}-trees associated with the nonterminals $A \in N$ are defined inductively as follows:

(1) if $[A; A \to \alpha; \beta] \in \Sigma_0^{SD}$, then $[A; A \to \alpha; \beta] \in P(SD, A, A)$;

(2) if $[A; A \to \alpha; \beta\,(\sigma)] \in \Sigma_m^{SD}$ with $m \geq 1$ and $\mathrm{pr}_N(\alpha) = A_1 \dots A_m$, and $t_1 \in P(SD, A_1, A_1), \dots, t_m \in P(SD, A_m, A_m)$, then $[A; A \to \alpha; \beta\,(\sigma)](t_1, \dots, t_m) \in P(SD, A, A)$.

The set of *production trees* of SD is now the Σ^{SD}-tree language $P(SD) := P(SD, S, S)$.

Then it is easy to see that $P(SD) = L(D, E)$, where

- D is the set of all symbols of the form $[S; S \to \alpha; \beta\,(\sigma)]$ in Σ^{SD}, and
- $E \subseteq \mathrm{fork}(\Sigma^{SD}, \emptyset)$ is the set of all forks of the form

  $$[A; A \to \alpha; \beta\,(\sigma)]([A_1; A_1 \to \alpha_1; \beta_1\,(\sigma_1)], \dots, [A_m; A_m \to \alpha_m; \beta_m\,(\sigma_m)])$$

  with $m \geq 1$ and $A; A \to \alpha; \beta\,(\sigma) \in P$ such that $\mathrm{pr}_N(\alpha) = A_1 \dots A_m$ and $A_i; A_i \to \alpha_i; \beta_i\,(\sigma_i) \in P$ for all $i \in [m]$,

if one shows by tree induction that for all $t \in T_{\Sigma^{SD}}$, $t \in P(SD)$ if and only if $\mathrm{fork}(t) \subseteq E$.

Moreover, we can construct the dT Σ^{SD}-recognizer $TR = (Q, \Sigma^{\mathrm{SD}}, \emptyset, P', I)$ such that $P(SD) = T(TR)$, in the following way:

- $Q := \{q^A \mid A \in N\}$,
- $I := \{q^S\}$, and
- P' is built from P as follows:
  - if $[A; A \to \alpha; \beta] \in \Sigma_0^{SD}$, then $q^A([A; A \to \alpha; \beta]) \to [A; A \to \alpha; \beta]$ is in P';
  - if $[A; A \to \alpha; \beta\,(\sigma)] \in \Sigma_m^{SD}$ for some $m \geq 1$ and for a production $A; A \to \alpha; \beta\,(\sigma)$ of the form (3.3.1) in P with $\mathrm{pr}_N(\alpha) = A_1, \dots, A_m$, then the rule

    $$q^A([A; A \to \alpha; \beta\,(\sigma)](\xi_1, \dots, \xi_m)) \to [A; A \to \alpha; \beta\,(\sigma)](q^{A_1}(\xi_1), \dots, q^{A_m}(\xi_m))$$

    is in P'.

This is obvious if one shows by tree induction that for all $A \in N$ and $t \in T_{\Sigma^{SD}}$, $t \in P(SD, A, A)$ if and only if $q^A(t) \Rightarrow_{TR}^* t$, where $\mathrm{root}(t)$ is of the form $[A; A \to \alpha; \beta\,(\sigma)]$. We record these facts as a proposition (cf. Steinby and Tîrnăucă, 2009, Theorem 6.2).

Proposition 6.2.18. $P(SD) \in \mathrm{Loc} \cap \mathrm{DRec}^{\mathrm{vf}}$. □

By slight modifications (see Tîrnăucă, 2007, Tîrnăucă, 2011), one can associate with any SCFG $SC = (N, X, Y, P, S, S')$ a ranked alphabet Σ^{SC}, specify inductively the sets $P(SC, A, B)$ of Σ^{SC}-trees associated with the nonterminals $A, B \in N$ and define the set of production trees of SC as the Σ^{SC}-tree language $P(SC) := P(SC, S, S')$. Hence, we get.

Corollary 6.2.19. $P(SC) \in \mathrm{Loc} \cap \mathrm{DRec}^{\mathrm{vf}}$. □

To see how a production tree represents a derivation in SD, we split it into a tree that represents the generated input and a tree that represents the generated output. This way we also associate in a natural way a tree transformation with SD.

First, let us arbitrarily number the production in P by $1, 2, \ldots$. Now, two ranked alphabets Σ^{in} and Σ^{out} are defined as follows. For any $m \geq 0$, let

$$\Sigma^{\mathrm{in}}_m := \{[A \to \alpha]_i \mid (\exists \beta, \sigma)\, A; A \to \alpha; \beta\, (\sigma) \text{ is the } i^{\mathrm{th}} \text{ production in } P \text{ and } |\alpha| = m\}\ ,$$

and similarly

$$\Sigma^{\mathrm{out}}_m := \{[A \to \beta]_i \mid (\exists \alpha, \sigma)\, A; A \to \alpha; \beta\, (\sigma) \text{ is the } i^{\mathrm{th}} \text{ production in } P \text{ and } |\beta| = m\}\ .$$

Note that Σ^{in} and Σ^{out} are essentially the ranked alphabets $\Sigma^{G^{\mathrm{in}}}$ and $\Sigma^{G^{\mathrm{out}}}$ belonging to the input and output grammars, respectively, but their symbols are augmented with numbers that identify the original rule in P. Hence, the same production $A \to \alpha$ of G^{in} may appear in more than one symbol of Σ^{in}, and similarly for productions of the output grammar.

Next, we define two quasi-alphabetic tree homomorphisms $\varphi^{SD} : T_{\Sigma^{SD}} \to T_{\Sigma^{\mathrm{in}}}(X)$ and $\psi^{SD} : T_{\Sigma^{SD}} \to T_{\Sigma^{\mathrm{out}}}(Y)$ by the following respective mappings φ^{SD}_m and ψ^{SD}_m $(m \geq 0)$:

(1) If $A; A \to \alpha; \beta$, where $\alpha = x_1 \ldots x_k$ and $\beta = y_1 \ldots y_l$ for some $k, l \geq 0$, $x_1, \ldots, x_k \in X$ and $y_1, \ldots, y_l \in Y$, is the i^{th} production in P, then we set

$$\varphi^{SD}_0([A; A \to \alpha; \beta]) := [A \to \alpha]_i(x_1, \ldots, x_k)$$

and

$$\psi^{SD}_0([A; A \to \alpha; \beta]) := [A \to \beta]_i(y_1, \ldots, y_l)\ .$$

(2) If $m \geq 1$ and $p = A; A \to \alpha; \beta\, (\sigma)$ is the i^{th} production in P of the form (3.3.1), then let

$$\varphi^{SD}_m([p]) := [A \to \alpha]_i(x_{01}, \ldots, x_{0k_1}, \xi_1, x_{11}, \ldots, x_{m-1k_{m-1}}, \xi_m, x_{m1}, \ldots, x_{mk_m})\ ,$$

and

$$\psi^{SD}_m([p]) := [A \to \beta]_i(y_{01}, \ldots, y_{0l_1}, \xi_{\sigma(1)}, y_{11}, \ldots, y_{m-1l_{m-1}}, \xi_{\sigma(m)}, y_{m1}, \ldots, y_{ml_m}).$$

Clearly, $TB_{SD} := (\varphi^{SD}, P(SD), \psi^{SD})$ is a quasi-alphabetic tree bimorphism. We regard $\tau(TB_{SD})\, (\subseteq T_{\Sigma^{\mathrm{in}}}(X) \times T_{\Sigma^{\mathrm{out}}}(Y))$ as the *tree transformation defined* by SD. In an analogous way, we can construct, for every SCFG SC, a quasi-alphabetic tree bimorphism $TB_{SC} := (\varphi^{SC}, P(SC), \psi^{SC})$ and refer to $\tau(TB_{SC})$ as the tree transformation defined by SC (see Tîrnăucă, 2007, Tîrnăucă, 2011, for details). Let $\tau[SDTS]$ ($\tau[SCFG]$, respectively) further denote the class of all tree transformations definable by SDTSs (SCFGs, respectively).

Next, we first note a couple of lemmas that further clarify the meaning of the production trees of an SDTS.

Lemma 6.2.20. *If $t \in P(SD, A, A)$ for some $A \in N$ then $(A, A) \Rightarrow^*_{SD} (\mathrm{yd}(t\varphi^{SD}), \mathrm{yd}(t\psi^{SD}))$.*

Proof. We proceed by tree induction on $t \in P(SD, A, A)$.

(1) If $t = [A; A \to \alpha; \beta] \in \Sigma_0^{SD}$, then $\alpha \in X^*$ and $\beta \in Y^*$, and hence $\mathrm{yd}(t\varphi^{SD}) = \alpha$ and $\mathrm{yd}(t\psi^{SD}) = \beta$. So, $(A, A) \Rightarrow_{SD} (\alpha, \beta)$.

(2) If $t = [A; A \to \alpha; \beta\,(\sigma)](t_1, \ldots, t_m)$, where $m \geq 1$ and $A; A \to \alpha; \beta\,(\sigma)$ is a production of the form (3.3.1) in P, then $t_i \in P(SD, A_i, A_i)$ for every $i \in [m]$. Hence, by the induction assumption, $(A_i, A_i) \Rightarrow^*_{SD} \big(\mathrm{yd}(t_i\varphi^{SD}), \mathrm{yd}(t_i\psi^{SD})\big)$ for every $i \in [m]$. Because we have

$$\mathrm{yd}(t\varphi^{SD}) = v_0\mathrm{yd}(t_1\varphi^{SD})v_1 \ldots v_{m-1}\mathrm{yd}(t_m\varphi^{SD})v_m$$

and

$$\mathrm{yd}(t\psi^{SD}) = w_0\mathrm{yd}(t_{\sigma(1)}\psi^{SD})w_1 \ldots w_{m-1}\mathrm{yd}(t_{\sigma(m)}\psi^{SD})w_m \ ,$$

we finally get $(A, A) \Rightarrow^*_{SD} (\mathrm{yd}(t\varphi^{SD}), \mathrm{yd}(t\psi^{SD}))$.

□

Basically, the above lemma shows that φ^{SD} maps each production tree of SD to a tree that represents the structure of the generated input string. Similarly, ψ^{SD} produces a tree for the generated output string. Also the following converse of Lemma 6.2.20 holds.

Lemma 6.2.21. *If $(A, A) \Rightarrow^*_{SD} (v, w)$ with $v \in X^*$ and $w \in Y^*$, then there exists a t in $P(SD, A, A)$ such that $v = \mathrm{yd}(t\varphi^{SD})$ and $w = \mathrm{yd}(t\psi^{SD})$.*

Proof. We proceed by induction on the length of a derivation witnessing $(A, A) \Rightarrow^*_{SD} (v, w)$.

(1) If $(A, A) \Rightarrow_{SD} (v, w)$, then $A; A \to v; w \in P$, and one may take $t := [A; A \to v; w]$.

(2) Let $(A, A) \Rightarrow^*_{SD} (v, w)$ by a derivation of length $n \geq 2$ and assume that the assertion holds for all shorter derivations. Hence, there exist a production $A; A \to \alpha; \beta\,(\sigma)$ of the form (3.3.1) in P and, for every $i \in [m]$, derivations $(A_i, A_i) \Rightarrow^*_{SD} (\bar{v}_i, \bar{w}_i)$ each in at most $n - 1$ steps such that $v = v_0\bar{v}_1v_1 \ldots v_{m-1}\bar{v}_mv_m$ and $w = w_0\bar{w}_{\sigma(1)}w_1 \ldots w_{m-1}\bar{w}_{\sigma(m)}w_{m+1}$. By the inductive assumption, for every $i \in [m]$, there exists t_i in $P(SD, A_i, A_i)$ such that $\bar{v}_i = \mathrm{yd}(t_i\varphi^{SD})$ and $\bar{w}_i = \mathrm{yd}(t_i\psi^{SD})$. If we take $t := [A; A \to \alpha; \beta\,(\sigma)](t_1, \ldots, t_m)$, then clearly $t \in P(SD, A, A)$. Since

$$\mathrm{yd}(t\varphi^{SD}) = v_0\mathrm{yd}(t_1\varphi^{SD})v_1 \ldots v_{m-1}\mathrm{yd}(t_m\varphi^{SD})v_m$$

and

$$\mathrm{yd}(t\psi^{SD}) = w_0\mathrm{yd}(t_{\sigma(1)}\psi^{SD})w_1 \ldots w_{m-1}\mathrm{yd}(t_{\sigma(m)}\psi^{SD})w_m \ ,$$

we get $v = \mathrm{yd}(t\varphi^{SD})$ and $w = \mathrm{yd}(t\psi^{SD})$.

This concludes the proof. □

The choice of definition of $\tau(TB_{SD})$ is then supported by the following fact.

Lemma 6.2.22. $\lambda(SD) = \lambda(TB_{SD})$.

Proof. If $(v, w) \in \lambda(SD)$, then there exists a derivation $(S, S) \Rightarrow_{SD}^{*} (v, w)$, and hence by Lemma 6.2.21, there is $t \in P(SD)$ such that $v = \mathrm{yd}(t\varphi^{SD})$ and $w = \mathrm{yd}(t\psi^{SD})$. So, $(v, w) \in \mathrm{yd}(\tau(SD))$, and consequently $\lambda(SD) \subseteq \lambda(TB_{SD})$.

Conversely, if $(v, w) \in \lambda(TB_{SD}) = \mathrm{yd}(\tau(TB_{SD}))$, then there exists $t \in P(SD)$ such that $v = \mathrm{yd}(t\varphi^{SD})$ and $w = \mathrm{yd}(t\psi^{SD})$. By Lemma 6.2.20, we get $(S, S) \Rightarrow_{SD}^{*} (v, w)$, so $(v, w) \in \lambda(SD)$. Hence, $\lambda(TB_{SD}) \subseteq \lambda(SD)$, which concludes the proof. □

The following facts follow immediately from the definitions and Lemma 6.2.22.

Proposition 6.2.23. $\tau[SDTS] \subseteq \tau[qTB^{\mathrm{vf}}]$, *and hence* $\lambda[SDTS] \subseteq \lambda[qTB^{\mathrm{vf}}]$. □

It is clear that the converse inclusion $\tau[qTB^{\mathrm{vf}}] \subseteq \tau[SDTS]$ cannot hold because every $\tau \in \tau[SDTS]$ is between trees over ranked alphabets only of the form Σ^{in} and Σ^{out}. However, for translations we can prove the equality $\lambda[SDTS] = \lambda[qTB]$.

Proposition 6.2.24. $\lambda[qTB^{\mathrm{vf}}] \subseteq \lambda[SDTS]$.

Proof. Consider a tree bimorphism $TB = (\varphi, R, \psi)$, where $\varphi\colon T_\Gamma \to T_\Sigma(X)$ and $\psi\colon T_\Gamma \to T_\Omega(Y)$ are quasi-alphabetic tree homomorphisms, and $R = L(D, E) \subseteq T_\Gamma$ is a variable-free local tree language. By Proposition 6.2.5, we can assume without loss of generality that φ is normalized. We construct an SDTS $SD = (N, X, Y, P, S)$ as follows.

For each $f \in \Gamma$, we introduce a nonterminal $\widehat{f}$, and let $N := \{S\} \cup \{\widehat{f} \mid f \in \Gamma\}$, where S is a new nonterminal. Let P consist of the following rules.

(1) If $f \in D$, then $S; S \to \widehat{f}; \widehat{f}$ is in P.

(2) If $e \in \Gamma_0$, then $\widehat{e}; \widehat{e} \to \mathrm{yd}(\varphi_0(e)); \mathrm{yd}(\psi_0(e))$ is in P.

(3) Consider any element $f(f_1, \ldots, f_m)$ of E. If

$$\varphi_m(f) = g(x_{01}, \ldots, x_{0k_1}, \xi_1, x_{11}, \ldots, x_{m-1k_{m-1}}, \xi_m, x_{m1}, \ldots, x_{mk_m})$$

and

$$\psi_m(f) = h(y_{01}, \ldots, y_{0l_1}, \xi_{\sigma(1)}, y_{11}, \ldots, y_{m-1l_{m-1}}, \xi_{\sigma(m)}, y_{m1}, \ldots, y_{ml_m}) \ ,$$

with $g \in \Sigma$, $h \in \Omega$, σ a permutation of $[m]$, and $k_i, l_i \in \mathbb{N}$, $x_{ij} \in X$ $(j \in [k_i])$ and $y_{il} \in Y$ $(l \in [l_i])$ for each $0 \le i \le m$, then P includes the rule

$$\widehat{f}; \widehat{f} \to v_0 \widehat{f}_1 v_1 \ldots v_{m-1} \widehat{f}_m v_m; w_0 \widehat{f}_{\sigma(1)} w_1 \ldots w_{m-1} \widehat{f}_{\sigma(m)} w_m \, (\sigma) \ ,$$

where for every $0 \le i \le m$, $v_i = x_{i1} x_{i2} \ldots x_{ik_i}$ and $w_i = y_{i1} y_{i2} \ldots y_{il_i}$.

It can now be verified that $\lambda(SD) = \lambda(TB) (= \mathrm{yd}(\tau(TB)))$.

For the inclusion $\lambda(TB) \subseteq \lambda(SD)$, it suffices to show by tree induction on $t \in T_\Gamma$ that if $\mathrm{root}(t) = f$ and $\mathrm{fork}(t) \subseteq E$, then $(\widehat{f}, \widehat{f}) \Rightarrow_{SD}^{*} (\mathrm{yd}(t\varphi), \mathrm{yd}(t\psi))$.

(1) For $t \in \Gamma_0$, the result follows from clause (2) of the definition of P.

(2) If $t = f(t_1, \ldots, t_m)$ for some $m \geq 1$ and $f \in \Sigma_m$, and $\mathrm{root}(t_i) = f_i$ for all $i \in [m]$, then $f(f_1, \ldots, f_m) \in E$ and for all $i \in [m]$, $\mathrm{fork}(t_i) \subseteq E$. By the induction assumption, $(\widehat{f}_i, \widehat{f}_i) \Rightarrow^*_{SD} (\mathrm{yd}(t_i\varphi), \mathrm{yd}(t_i\psi))$. If $\varphi_m(f)$ and $\psi_m(f)$ are as in clause (3) of the definition of P, then $\mathrm{fork}(t) \subseteq E$, and hence P contains the rule

$$\widehat{f}; \widehat{f} \to v_0\widehat{f}_1 v_1 \ldots v_{m-1}\widehat{f}_m v_m; w_0\widehat{f}_{\sigma(1)} w_1 \ldots w_{m-1}\widehat{f}_{\sigma(m)} w_m \ ,$$

where $v_i = x_{i1}x_{i2}\ldots x_{ik_i}$ and $w_i = y_{i1}y_{i2}\ldots y_{il_i}$ for every $0 \leq i \leq m$. We immediately obtain $(\widehat{f}, \widehat{f}) \Rightarrow^*_{SD} (\mathrm{yd}(t\varphi), \mathrm{yd}(t\psi))$ because $\mathrm{yd}(t\varphi) = v_0\mathrm{yd}(t_1\varphi)v_1 \ldots v_{m-1}\mathrm{yd}(t_m\varphi)v_m$ and $\mathrm{yd}(t\psi) = w_0\mathrm{yd}(t_{\sigma(1)}\psi)w_1 \ldots w_{m-1}\mathrm{yd}(t_{\sigma(m)}\psi)w_m$, .

Indeed, if $t \in R$, then $\mathrm{root}(t) = f$ belongs to D and $\mathrm{fork}(t) \subseteq E$. Hence, $S; S \to \widehat{f}; \widehat{f} \in P$ and $(\widehat{f}, \widehat{f}) \Rightarrow^*_{SD} (\mathrm{yd}(t\varphi), \mathrm{yd}(t\psi))$, so $(\mathrm{yd}(t\varphi), \mathrm{yd}(t\psi)) \in \lambda(SD)$.

Next, to show $\lambda(SD) \subseteq \lambda(TB)$, one first proves by induction on the length of the derivation that if $(\widehat{f}, \widehat{f}) \Rightarrow^*_{SD} (v, w)$ for some $f \in \Gamma$, $v \in X^*$ and $w \in Y^*$, then there is a tree $t \in T_\Gamma$ such that $\mathrm{root}(t) = f$, $\mathrm{fork}(t) \subseteq E$, $v = \mathrm{yd}(t\varphi)$, and $w = \mathrm{yd}(t\psi)$.

(1) If $(\widehat{f}, \widehat{f}) \Rightarrow_{SD} (v, w)$, then $\widehat{f}; \widehat{f} \to v; w \in P$, $f \in \Gamma_0$, $v = \mathrm{yd}(\varphi_0(f))$ and $w = \mathrm{yd}(\psi_0(f))$. Obviously, we may choose $t = f$.

(2) Let $(\widehat{f}, \widehat{f}) \Rightarrow^*_{SD} (v, w)$ by a derivation of length $n \geq 2$, and assume that the assertion holds for all derivations of at most $n-1$ steps. Then there exists in P a production

$$\widehat{f}; \widehat{f} \to v_0\widehat{f}_1 v_1 \ldots v_{m-1}\widehat{f}_m v_m; w_0\widehat{f}_{\sigma(1)} w_1 \ldots w_{m-1}\widehat{f}_{\sigma(m)} w_m\,(\sigma) \ ,$$

such that $v = v_0\bar{v}_1 v_1 \ldots v_{m-1}\bar{v}_m v_m$ and $w = w_0\bar{w}_{\sigma(1)} w_1 \ldots w_{m-1}\bar{w}_{\sigma(m)} w_m$, where for each $i \in [m]$, $(\widehat{f}_i, \widehat{f}_i) \Rightarrow^*_{SD} (\bar{v}_i, \bar{w}_i)$ in at most $n-1$ steps. By the induction assumption, there exists, for each $i \in [m]$, a tree $t_i \in T_\Gamma$ such that $\mathrm{root}(t_i) = f_i$, $\mathrm{fork}(t_i) \subseteq E$, $\bar{v}_i = \mathrm{yd}(t_i\varphi)$ and $\bar{w}_i) = \mathrm{yd}(t_i\psi)$. Now, $\mathrm{fork}(t) \subseteq E$ for $t = f(t_1, \ldots, t_m)$ because also $f(f_1, \ldots, f_m) \in E$. Moreover, $\mathrm{yd}(t\varphi) = v_0\mathrm{yd}(t_1\varphi)v_1 \ldots v_{m-1}\mathrm{yd}(t_m\varphi)v_m{=}v$ and $\mathrm{yd}(t\psi) = w_0\mathrm{yd}(t_{\sigma(1)}\psi)w_2 \ldots w_{m-1}\mathrm{yd}(t_{\sigma(m)}\psi)w_m{=}w$.

Then we can argue as follows: if $(v, w) \in \lambda(SD)$, then $(S, S) \Rightarrow^*_{SD} (v, w)$, and hence there exists a nonterminal $\widehat{f} \in N$ such that $S; S \to \widehat{f}; \widehat{f}$ is in P and $(\widehat{f}, \widehat{f}) \Rightarrow^*_{SD} (v, w)$. But $\mathrm{root}(t) = f \in D$, $\mathrm{fork}(t) \subseteq E$, $v = \mathrm{yd}(t\varphi)$ and $w = \mathrm{yd}(t\psi)$, and hence $(v, w) \in \lambda(TB)$. □

If we put together Propositions 6.2.23, 6.2.24 and 3.3.21, we get the main result of this section (cf. also Tîrnăucă, 2011, Tîrnăucă, 2007).

Theorem 6.2.25. $\lambda[SDTS] = \lambda[SCFG] = \lambda[qTB^{\mathrm{vf}}]$. □

Let us also note that Melamed (2003, p. 81) has shown that two-dimensional *multitext grammars* are equal in generative power to SDTSs. Hence they also perform the same translations as quasi-alphabetic tree bimorphisms. Whether every n-dimensional multitext grammar can be split into a cascade of quasi-alphabetic tree translations, is an open question.

Next, we give an example that shows how the ideas and constructions presented so far can be extended from SDTSs to the more general case of SCFGs (cf. Tîrnăucă, 2011,

Example 5.1). However, a complete example of how to construct a quasi-alphabetic tree bimorphism from a given SDTS that models a piece of an English-to-Spanish translation was given by Tîrnăucă (2008, Section 5).

Example 6.2.26. Let $SC = (\{S, S', A, B, C\}, \{x, y, z\}, \{0, 1\}, P, S, S')$ be an SCFG, where P consists of the following productions:

$$\begin{aligned}
&p_1 := S; S' \to xyASzB; B0SA1\,(3\ 1\ 2),\\
&p_2 := S; A \to BzC; C00B(2\ 1),\\
&p_3 := S; A \to AS; 1SS'(1\ 2),\\
&p_4 := A; S \to z; \varepsilon\\
&p_5 := C; C \to xyz; 1, \text{ and}\\
&p_6 := B; B \to x; 11\ .
\end{aligned}$$

A (leftmost) derivation in SC is

$$\begin{aligned}
(S, S') &\Rightarrow_{SC}^{p_1} (xyASzB, B0SA1) \Rightarrow_{SC}^{p_4} (xyzSzB, B0A1) \Rightarrow_{SC}^{p_2} (xyzBzCzB, B0C00B1)\\
&\Rightarrow_{SC}^{p_6} (xyzxzCzB, B0C00111) \Rightarrow_{SC}^{p_5} (xyzxzxyzzB, B0100111)\\
&\Rightarrow_{SC}^{p_6} (xyzxzxyzzx, 110100111)\ ,
\end{aligned}$$

where the production applied in each derivation step is mentioned by a superscript on the derivation relation $\Rightarrow_{SC}$. Therefore, $(xyzxzxyzzx, 110100111)$ is an element in the translation of SC.

Now, we have:

- $\Sigma^{SC} = \{[p_1]/3, [p_2]/2, [p_3]/2, [p_4]/0, [p_5]/0, [p_6]/0\}$,
- $\Sigma^{\text{in}} = \{[S \to xyASzB]_1/6, [S \to BzC]_2/3, [C \to xyz]_5/3, [S \to AS]_3/2, [A \to z]_4/1, [B \to x]_6/1\}$, and
- $\Sigma^{\text{out}} = \{[S' \to B0SA1]_1/5, [A \to C00B]_2/4, [A \to 1SS']_3/3, [B \to 11]_6/2, [C \to 1]_5/1, [S \to \varepsilon]_4/0\}$.

Moreover, the set $P(SC) \subseteq T_{\Sigma^{SC}}$ of production trees of SC is the local variable-free tree language $L(D, E)$, where $D = \{[p_1]\}$ and

$$E = \{[p_1]([p_4], [p_2], [p_6]), [p_1]([p_4], [p_3], [p_6]), [p_2]([p_6], [p_5]), [p_3]([p_4], [p_1])\}\ .$$

A production tree in $P(SC)$ representing the above derivation is shown in Figure 6.2.2 (up).

The tree homomorphisms $\varphi^{SC}\colon T_{\Sigma^{SC}} \to T_{\Sigma^{\text{in}}}(X)$ and $\psi^{SC}\colon T_{\Sigma^{SC}} \to T_{\Sigma^{\text{out}}}(Y)$ defined by setting

$$\begin{aligned}
&\varphi_3^{SC}([p_1]) := [S \to xyASzB]_1(x, y, \xi_1, \xi_2, z, \xi_3) && \psi_3^{SC}([p_1]) := [S' \to B0SA1]_1(\xi_3, 0, \xi_1, \xi_2, 1)\\
&\varphi_2^{SC}([p_2]) := [S \to BzC]_2(\xi_1, z, \xi_2) && \psi_2^{SC}([p_2]) := [A \to C00B]_2(\xi_2, 0, 0, \xi_1)\\
&\varphi_2^{SC}([p_3]) := [S \to AS]_3(\xi_1, \xi_2) && \psi_2^{SC}([p_3]) := [S \to 1SS']_3(1, \xi_1, \xi_2)\\
&\varphi_0^{SC}([p_4]) := [A \to z]_4(z) && \psi_0^{SC}([p_4]) := [S \to \varepsilon]_4\\
&\varphi_0^{SC}([p_5]) := [C \to xyz]_5(x, y, z) && \psi_0^{SC}([p_5]) := [C \to 1]_5(1)\\
&\varphi_0^{SC}([p_6]) := [B \to x]_6(x) && \psi_0^{SC}([p_6]) := [B \to 11]_6(1, 1)
\end{aligned}$$

are quasi-alphabetic.

Thus, we constructed the quasi-alphabetic tree bimorphism $TB_{SC} = (\varphi^{SC}, P(SC), \psi^{SC})$ and hence $\tau(TB_{SC}) = \{(t\varphi^{SC}, t\psi^{SC}) \mid t \in P(SC)\}$. Figure 6.2.2 shows the pair $(t\varphi, t\psi)$ in $\tau(TB_{SC})$ for the production tree $t = [p_1]([p_4], [p_2]([p_6], [p_5]), [p_6])$ in $P(SC)$. Consequently, we obtain $(xyzxzxyzzx, 110100111) \in \lambda(TB_{SC}) = \lambda(SC)$ as it should (cf. Proposition 6.2.23). □

Using the connection of Theorem 6.2.25, we can prove properties of syntax-directed translations using the algebraic mechanisms of tree language theory. It is more elegant to show, for example, that the domain and range of the translation defined by any SDTS *SD* are CFLs (cf. Proposition 3.3.9). Indeed, let $TB = (\varphi, R, \psi)$ be a quasi-alphabetic tree bimorphism such that $\lambda(SD) = \lambda(TB)$ for a given SDTS *SD*. Then, by Theorems 4.4.17 and 4.6.2, $\mathrm{dom}(\lambda(SD))$ is a CFL as the yield of the recognizable tree language $\mathrm{dom}(\tau(TB)) = (T_\Omega(Y)\psi^{-1} \cap R)\varphi = R\varphi$. Similarly, $\mathrm{Range}(\lambda(SD)) = \mathrm{yd}(R\psi)$ is seen to be context-free especially when one notes that quasi-alphabetic tree bimorphisms are closed under inverses by Proposition 6.2.11.

We continue by showing that a special type of quasi-alphabetic tree bimorphisms defines the class $\lambda[FST]$ of regular translations (cf. Tîrnăucă, 2011). Thus, we may now also use tree language theory for proving properties of regular translations. We exemplify it by giving a proof based on tree language techniques of the well-known result of Theorem 3.2.4(iv)).

Firstly, we recall the definition of comblike tree languages (Gécseg and Steinby, 1984, p. 136). For any ranked alphabet Σ and leaf alphabet X, the set $\mathrm{Comb}_\Sigma(X)$ of *comblike* ΣX*-trees* is the smallest set $\mathcal{T}$ such that

(1) $X \cup \Sigma_0 \subseteq \mathcal{T}$, and

(2) $f(x_1, \ldots, x_{m-1}, t) \in \mathcal{T}$ whenever $m \geq 1$, $f \in \Sigma_m$, $t \in \mathcal{T}$ and $x_1, \ldots, x_{m-1} \in X$.

We also note the following facts about comblike trees.

Remark 6.2.27. The following hold.

(i) If $T \subseteq \mathrm{Comb}_\Sigma(X)$ and $T \in \mathrm{Rec}_\Sigma(X)$, then $\mathrm{yd}(T)$ is regular (cf. Gécseg and Steinby, 1984, Chapter III, Ex.6, p.136).

(ii) If $T \subseteq \mathrm{Comb}_\Sigma(X)$ and $\varphi\colon T_\Sigma(X) \to T_\Omega(Y)$ is a quasi-alphabetic tree homomorphism, then $\varphi(T)$ is a set of comblike trees.

A quasi-alphabetic tree bimorphism $TB = (\varphi, R, \psi)$ is called a *comblike tree bimorphism* if

- $R \subseteq T_\Gamma$ is a set of comblike trees over Γ, where Γ is a ranked alphabet such that $\bigcup_{m \geq 2} \Gamma_m = \emptyset$.
- $\varphi\colon T_\Gamma \to T_\Sigma(X)$ and $\psi\colon T_\Gamma \to T_\Omega(Y)$ are quasi-alphabetic tree homomorphisms defined as follows.
  - For every $e \in \Gamma_0$,

$$\varphi_0(e) := f(x_1, \ldots, x_k) \qquad \text{and} \qquad \psi_0(e) := g(y_1, \ldots, y_l)$$

for some $k, l \geq 0$, $f \in \Sigma_k$, $g \in \Omega_l$, $x_i \in X$ $(i \in [k])$ and $y_j \in Y$ $(j \in [l])$.

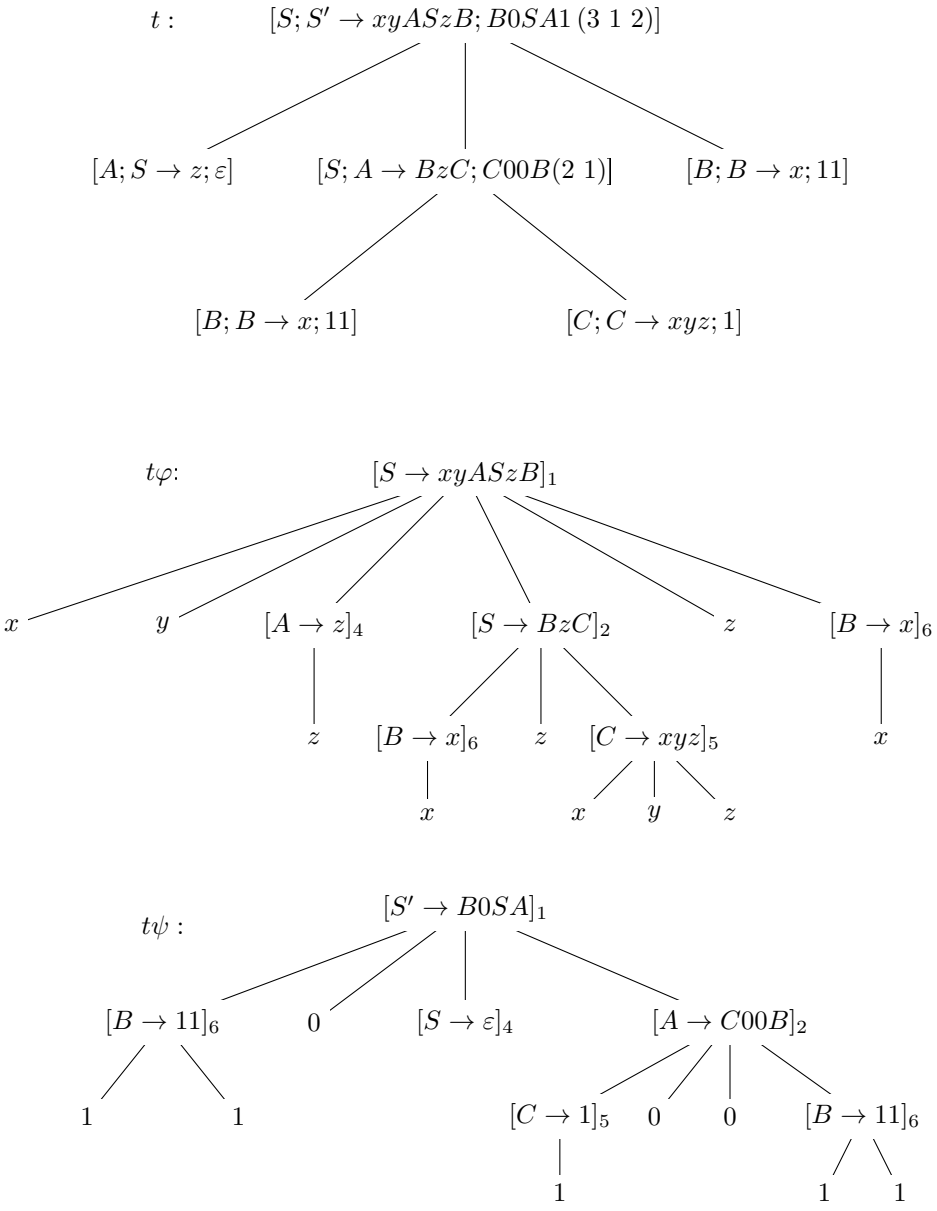


Figure 6.2.2: An element in the tree transformation $\tau(TB_{SC})$ defined by the SCFG SC of Example 6.2.26: the tree $t\psi$ in $T_{\Sigma^{\text{out}}}(Y)$ (down) is a transform of the tree $t\varphi$ in $T_{\Sigma^{\text{in}}}(X)$ (middle) for the production tree t in $P(SC)$ (up).

– For every $g \in \Gamma_1$,

$$\varphi_1(g) := f(x_1, \ldots, x_{k-1}, \xi) \qquad \text{and} \qquad \psi_1(g) := h(y_1, \ldots, y_{l-1}, \xi)$$

for some $k, l \geq 1$, $f \in \Sigma_k$, $h \in \Omega_l$, $x_i \in X$ $(i \in [k-1])$ and $y_j \in Y$ $(j \in [l-1])$.

Now, we recall the constructions given in the beginning of this section. For any given right-linear SDTS $SD = (N, X, Y, P, S)$, we can turn its set of productions P into a ranked alphabet P' consisting only of unary and nullary symbols (i.e., for any $p = A; A \to \alpha; \beta\,(\sigma)$ in P, $\mathrm{rk}(p) = |\mathrm{pr}_N(\alpha)| \in P'$). Then the set of production trees $P(SD) \subseteq T_{P'}$ is a set of comblike trees over P', and the defined quasi-alphabetic tree homomorphisms φ^{SD} and ψ^{SD} are comblike. Thus, for any right-linear SDTS SD we can construct a comblike tree bimorphism $(\varphi^{SD}, P(SD), \psi^{SD})$ defining the same translation (cf. Proposition 6.2.23). On the other hand, if in the proof of Proposition 6.2.24 the given quasi-alphabetic tree bimorphism (φ, R, ψ) is comblike, then the constructed SDTS SD is right-linear. Therefore, we may note.

Proposition 6.2.28. *The class of regular translations is effectively equal to the class of translations defined by comblike tree bimorphisms.* □

Now, many results concerning regular translations can be derived directly from tree language theory by using the connection provided by Proposition 6.2.28. As an example, we give a new proof of a well-known result (see Theorem 3.2.4(iv)).

Proposition 6.2.29. *Regular translations and their inverses preserve regular languages.*

Proof. Let L be a regular language and λ a regular translation. By Proposition 6.2.28, there is a comblike tree bimorphism $TB = (\varphi, R, \psi)$ such that $\lambda = \lambda(TB)$. Thus, $L\lambda = L\lambda(TB) = \mathrm{yd}((\mathrm{yd}^{-1}(L)\varphi^{-1} \cap R)\psi)$, which is regular by the following reasoning:

- $\mathrm{yd}^{-1}(L)$ is a recognizable tree language since L is regular (Gécseg and Steinby, 1984, Theorem III.3.2);
- $(\mathrm{yd}^{-1}(L)\varphi^{-1} \cap R)\psi$ is recognizable since recognizable tree languages are closed under inverse tree homomorphisms, intersection and linear tree homomorphisms (Theorem 4.4.17);
- $(\mathrm{yd}^{-1}(L)\varphi^{-1} \cap R)\psi$ is a set of comblike trees by Remark 6.2.27(ii);
- $\mathrm{yd}((\mathrm{yd}^{-1}(L)\varphi^{-1} \cap R)\psi)$ is regular by Remark 6.2.27(i).

By similar arguments, $L\lambda^{-1} = \mathrm{yd}(\mathrm{yd}^{-1}(L)\psi^{-1} \cap R)\varphi$ is regular. □

Nevertheless, Nivat's original string bimorphism characterization of regular translations of Theorem 3.2.3 yields a simpler, less specialized proof of Proposition 6.2.29 (cf. Berstel, 1979).

Finally, we add to the result of Theorem 6.2.25 that every Cartesian product of CFLs is a quasi-alphabetic translation (cf. Maletti and Tîrnăucă, 2009). This sharpens Proposition 3.6 of Bozapalidis (1992), where the same property was proved for fine tree bimorphisms.

Theorem 6.2.30. *For all context-free languages L_1 and L_2, there exists a quasi-alphabetic tree bimorphism TB such that $\lambda(TB) = L_1 \times L_2$.*

Proof. Without any loss of generality, we can assume that $L_1, L_2 \subseteq X^*$. By Theorem 4.6.2, there exist recognizable tree languages $R_1 \subseteq T_\Sigma(X)$ and $R_2 \subseteq T_\Omega(X)$ such that $L_1 = \mathrm{yd}(R_1)$ and $L_2 = \mathrm{yd}(R_2)$. Let Y be a set such that $Y \cap (\Sigma \cup \Omega) = \emptyset$ and there exists a bijection $\phi\colon Y \to X$. Then, we extend ϕ to $\phi_\Sigma\colon \Sigma \cup Y \to \Sigma \cup X$ and $\phi_\Omega\colon \Omega \cup Y \to \Omega \cup X$ such that $\phi_\Sigma(f) := f$ and $\phi_\Omega(g) := g$ for every $f \in \Sigma$ and $g \in \Omega$. We denote the ranked alphabets $\Sigma \cup Y$ and $\Omega \cup Y$, in which all symbols of Y are nullary, by $\bar{\Sigma}$ and $\bar{\Omega}$, respectively. Next, we define the ranked alphabet

$$\bar{\Sigma} \vee \bar{\Omega} := \{\langle f, g\rangle \mid f \in \bar{\Sigma}, g \in \bar{\Omega}\}$$

such that $\mathrm{rk}(\langle f, g\rangle) = \max(\mathrm{rk}(f), \mathrm{rk}(g))$. In a similar way the ranked alphabets $\Sigma \vee \bar{\Omega}$ and $\bar{\Sigma} \vee \Omega$ are defined. Without loss of generality, we can assume that $\bar{\Sigma}_0 \neq Y \neq \bar{\Omega}_0$ and $\Sigma_1 \neq \emptyset \neq \Omega_1$.

Next we show how to embed a tree of $T_\Sigma(X)$ into $T_{\bar{\Sigma}\vee\bar{\Omega}}$. To this end, we define the linear $\mathrm{TOP}^{\mathrm{R}}$-transducer $TD^R_\Sigma = (\{\star\}, \bar{\Sigma} \vee \bar{\Omega}, \emptyset, \Sigma \cup X, \emptyset, P, \{\star\})$, where for every $\langle f, g\rangle \in (\bar{\Sigma} \vee \bar{\Omega})_m$ $(m \geq 0)$, we have in P the rule

$$\big\langle \star(\langle f, g\rangle(\xi_1, \ldots, \xi_m)) \to \phi_\Sigma(f)(\star(\xi_1), \ldots, \star(\xi_{\mathrm{rk}(m)})), M\big\rangle$$

with look-ahead $M(\xi_i) := T_i$, $i \in [m]$, where $T_1 = \ldots = T_{\mathrm{rk}(f)} = T_{\bar{\Sigma}\vee\bar{\Omega}}$ and $T_{\mathrm{rk}(f)+1} = \ldots = T_m = T_{\Sigma\vee\bar{\Omega}}$. In an analogous way the linear $\mathrm{TOP}^{\mathrm{R}}$-transducer TD^R_Ω is defined. Let $R := \tau(TD^R_\Sigma)^{-1}(R_1) \cap \tau(TD^R_\Omega)^{-1}(R_2)$, which is recognizable by Theorem 4.4.17(i) and Table 5.2.1. Next, we take the quasi-alphabetic tree homomorphism $\varphi\colon T_{\bar{\Sigma}\vee\bar{\Omega}} \to T_{\Sigma\vee\bar{\Omega}}(X)$, which is defined for every $\langle f, g\rangle \in (\bar{\Sigma} \vee \bar{\Omega})_m$ $(m \geq 0)$ by

$$\varphi_m(\langle f, g\rangle) := \begin{cases} \langle f, g\rangle(\xi_1, \ldots, \xi_k) & \text{if } f \in \Sigma_k \\ \langle h_1, h_2\rangle(\phi(f)) & \text{otherwise} \end{cases},$$

where $\langle h_1, h_2\rangle \in \Sigma_1 \times \Omega_1$ is arbitrary. In an analogous fashion, the quasi-alphabetic tree homomorphism $\psi\colon T_{\bar{\Sigma}\vee\bar{\Omega}} \to T_{\bar{\Sigma}\vee\Omega}(X)$ is defined. Now if we take the quasi-alphabetic tree bimorphism $TB = (\varphi, R, \psi)$, then it should be clear that $\mathrm{yd}_X(t\varphi) = \mathrm{yd}_X(t\tau(TD^R_\Sigma))$ and $\mathrm{yd}_X(t\psi) = \mathrm{yd}_X(t\tau(TD^R_\Omega))$ for every $t \in T_{\bar{\Sigma}\vee\bar{\Omega}}$. Consequently, $\lambda(TB) = L_1 \times L_2$, which concludes our proof. □

Note that other less special proof can be given if one directly constructs an SCFG SC such that $\lambda(SC) = L_1 \times L_2\, (\subseteq X^* \times X^*)$ and then uses Theorem 6.2.25. This can be done as follows. By Theorem 2.4.8, there exist CFGs $CF_1 = (N_1, X, P_1, S_1)$ and $CF_2 = (N_2, X, P_2, S_2)$ in CNF such that $L_1 = L(CF_1)$ and $L_2 = L(CF_2)$. We define the SCFG $SC = (N_1 \cup N_2, X, X, P, S_1, S_2)$ thus.

(1) For any productions $A_1 \to B_1C_1$ in P_1 and $A_2 \to B_2C_2$ in P_2 with $A_1, B_1, C_1 \in N_1$ and $A_2, B_2, C_2 \in N_2$, we include the production $A_1; A_2 \to B_1C_1; B_2C_2\,(1\,2)$ in P.

(2) For any productions $A_1 \to x_1$ in P_1 and $A_2 \to x_2$ in P_2 with $A_1 \in N_1$, $A_2 \in N_2$ and $x_1, x_2 \in X$, we include the production $A_1; A_2 \to x_1; x_2$ in P.

(3) For the productions $S_1 \to \varepsilon$ in P_1 and $S_2 \to \varepsilon$ in P_2, we add the production $S_1; S_2 \to \varepsilon; \varepsilon$ to P.

It should be obvious then that $\lambda(SC) = L_1 \times L_2$. However, we preferred to show the power of quasi-alphabetic tree bimorphisms in the spirit of Bozapalidis (1992, Proposition 3.6).

6.2.3 Implementation by tree transducers

We already showed the usefulness of quasi-alphabetic tree bimorphisms in proving properties of tree transformations. Moreover, we improved the mathematical foundations of synchronous grammars such as, for example, SDTSs and SCFGs, by using the connection between quasi-alphabetic tree bimorphisms and syntax-directed translation devices. Consequently, we can now use tree language theory techniques to elegantly prove other well-known properties. However, tree bimorphisms are in general hard to train and less-used in real-world applications, because there is no toolkit to implement them. Therefore, connections with tree transducer classes are looked for.

In this section, we investigate upper bounds to the power of quasi-alphabetic tree bimorphisms by placing them in the widely known tree transducer hierarchy of Figure 5.2.3. Furthermore, we prove that there is a special type of linear non-deleting epsilon-free XTT-transducer, namely the *quasi-alphabetic tree transducer*, that effectively computes all quasi-alphabetic tree transformations (cf. Tîrnăucă, 2009). Thus, quasi-alphabetic tree bimorphisms are easily implemented in TIBURON (May and Knight, 2006, May, 2010) and trainable (Graehl et al., 2008), and therefore suitable for real-word applications, especially in machine translation. The relations between quasi-alphabetic tree bimorphisms (and implicitly synchronous grammars defining syntax-directed translations) and famous types of tree transducers are shown in Figure 6.2.3.

We start by proving that every quasi-alphabetic tree transformation can be computed by a linear TOP-transducer with finite look-ahead. With that we establish a rough upper bound to the power of quasi-alphabetic tree bimorphisms.

Proposition 6.2.31. $\tau[qTB] \subseteq \tau[\text{l-TOP}^{\text{F}}]$.

Proof. Let us consider a quasi-alphabetic bimorphism $TB = (\varphi, R, \psi)$, where $R \subseteq T_\Gamma(Z)$, $\varphi\colon T_\Gamma(Z) \to T_\Sigma(X)$ and $\psi\colon T_\Gamma(Z) \to T_\Omega(Y)$. By Proposition 6.2.5, we can assume without loss of generality that φ is normalized. Moreover, let $TR = (Q, \Gamma, Z, P, I)$ be a ΓZ-ndT such that $T(TR) = R$. We construct the linear TOP-transducer with finite look-ahead $TD^{\text{R}} = (Q, \Sigma, X, \Omega, Y, P', I)$ as follows.

1. For every transition $q(z) \to z \in P$ with $z \in Z$, we have the rule $\langle q(\varphi_Z(z)) \to \psi_Z(z), M\rangle$ in P' with look-ahead $M\colon \emptyset \to T_\Sigma(X)$;
2. For every transition $q(f(\xi_1, \ldots, \xi_m)) \to f(q_1(\xi_1), \ldots, q_m(\xi_m)) \in P$ with $m \geq 0$, $f \in \Gamma_m$ and $q_1, \ldots, q_m \in Q$, we have the rule

$$\langle q(\text{root}(\varphi_m(f))(\xi_1, \ldots, \xi_n)) \to \psi_m(f)[q_1(\xi_{j_1}), \ldots, q_m(\xi_{j_m})], M\rangle$$

in P' with look-ahead $M(\xi_i) := \{s_{j_i}\}$ for every $i \in [m]$, where $\varphi_m(f) = g(s_1, \ldots, s_n)$ for some $n \geq 0$, $g \in \Sigma_n$ and $s_1, \ldots, s_n \in T_\Sigma(X)$, and $j_i = \text{dom}_{\{\xi_i\}}(\varphi_m(f))$ for every $i \in [m]$.

First, let us prove $\tau(TB) \subseteq \tau(TD^{\text{R}})$ by showing $q(t\varphi) \Rightarrow^*_{TD^{\text{R}}} t\psi$ for every $q \in Q$ and $t \in T(TR)_q$, where $T(TR)_q$ stands for the tree language recognized by TR if q were the only initial state, i.e., $I = \{q\}$. We proceed by tree induction on t.

(1) Let $t \in Z$. Then $q(t\varphi) \Rightarrow_{TD^{\mathrm{R}}} t\psi$ using a rule constructed in item 1. above.

(2) Let $t = f(t_1, \ldots, t_m)$ for some $m \geq 0$, $f \in \Gamma_m$ and $t_1, \ldots, t_m \in T_\Gamma(Z)$. Moreover, let $q_1, \ldots, q_m \in Q$ be such that $q(f(\xi_1, \ldots, \xi_m)) \to f(q_1(\xi_1), \ldots, q_m(\xi_m)) \in P$ and $t_i \in T(TR)_{q_i}$ for every $i \in [m]$. Then

$$\begin{aligned} q(f(t_1, \ldots, t_m)\varphi) &= q(\varphi_m(f)[t_1\varphi, \ldots, t_m\varphi]) \\ &= q(g(s_1[t_1\varphi, \ldots, t_m\varphi], \ldots, s_n[t_1\varphi, \ldots, t_m\varphi])) \ , \end{aligned}$$

where $\varphi_m(f) = g(s_1, \ldots, s_n)$ for some $n \geq 0$, $g \in \Sigma_n$ and $s_1, \ldots, s_n \in T_\Sigma(X)$. Let $j_i = \mathrm{dom}_{\{\xi_i\}}(\varphi_m(f))$ for every $i \in [m]$. Then

$$q(f(t_1, \ldots, t_m)\varphi) \Rightarrow_{TD^{\mathrm{R}}} \psi_m(f)[q_1(s_{j_1}[t_1\varphi, \ldots, t_m\varphi]), \ldots, q_m(s_{j_m}[t_1\varphi, \ldots, t_m\varphi])]$$

using a rule constructed in the second item above. Note that the look-ahead restriction is trivially fulfilled. Clearly, $s_{j_i} = \xi_i$ for every $i \in [m]$ and thus we have

$$q(f(t_1, \ldots, t_m)\varphi) \Rightarrow_{TD^{\mathrm{R}}} \psi_m(f)[q_1(t_1\varphi), \ldots, q_m(t_m\varphi)] \ .$$

By the induction hypothesis, we have $q_i(t_i\varphi) \Rightarrow^*_{TD^{\mathrm{R}}} t_i\psi$ for every $i \in [m]$. Consequently, we obtain

$$q(t\varphi) \Rightarrow_{TD^{\mathrm{R}}} \psi_m(f)[q_1(t_1\varphi), \ldots, q_m(t_m\varphi)] \Rightarrow^*_{TD^{\mathrm{R}}} \psi_m(f)[t_1\psi, \ldots, t_m\psi] = t\psi \ .$$

This proves the auxiliary statement and $\tau(TB) \subseteq \tau(TD^{\mathrm{R}})$ if we consider states of I.

The converse inclusion can be proved after one shows that for every $q \in Q$, $s \in T_\Sigma(X)$, and $t \in T_\Omega(Y)$, if $q(s) \Rightarrow^*_{TD^{\mathrm{R}}} t$, then there exists $u \in T(TR)_q$ such that $s = u\varphi$ and $t = u\psi$. This can be done by induction on the length of the derivation in TD^{R}, and we omit further details here. □

Next, we show that linear TOP-transducers are not sufficiently powerful to implement all quasi-alphabetic tree transformations.

Proposition 6.2.32. $\tau[qTB] \parallel \tau[\text{l-TOP}]$.

Proof. Let $\Sigma = \{f/2, e/0\}$ and $X = \{x_1, x_2\}$. Moreover, let $\varphi\colon T_\Sigma \to T_\Sigma(X)$ be a quasi-alphabetic tree homomorphism with $\varphi_0(e) := f(x_1, x_2)$ for $e \in \Sigma_0$, and $\varphi_2(f) := f(\xi_1, \xi_2)$ for $f \in \Sigma_2$. Then $TB = (\varphi, \{e\}, \mathrm{id}_{T_\Sigma})$ is a quasi-alphabetic tree bimorphism that defines the tree transformation $\{(f(x_1, x_2), e)\}$. It is known that $\tau(TB)$ is not in $\tau[\text{l-TOP}]$ (Engelfriet, 1975a, Example 2.6). On the other hand, by Proposition 4.2.3(i) and Lemma 3.5 of Fülöp (2004) (cf. also Fülöp and Vogler, 1998, Lemma 3.27), it is obvious that there are tree transformations computed by linear TOP-transducers that are not quasi-alphabetic (Comon et al., 2007, Example A' on pp.169–170), and hence $\tau[qTB] \parallel \tau[\text{l-TOP}]$. □

Now, we focus on quasi-alphabetic tree transducers introduced by Tîrnăucă (2009). They are a particular type of (linear, non-deleting and epsilon-free) XTT-transducers formally defined as follows.

Definition 6.2.33. An extended top-down tree transducer $XT = (Q, \Sigma, X, \Omega, Y, P, I)$ is called *quasi-alphabetic* if in each production $q(\ell) \to r$ in P, either (1) $\ell \in X$ and $r \in Y$ or (2) for some $n \geq 0$,

$$\ell \in \Sigma(X \cup \Xi_n) \cap C^n_\Sigma(X) \qquad \text{and} \qquad r \in \Omega(Y \cup Q(\Xi_n)) \cap C^n_{\Omega \cup Q}(Y) \ ,$$

where

$$\Sigma(X \cup \Xi_n) := \{f(t_1, \ldots, t_m) \mid m \geq 1, f \in \Sigma_m \text{ and } t_i \in X \cup \Xi_n, \forall i \in [m]\}$$

and

$$\Omega(Y \cup Q(\Xi_n)) := \{g(t_1, \ldots, t_m) \mid m \geq 1, g \in \Omega_m \text{ and } \forall i \in [m], t_i \in Y \text{ or } t_i \in Q(\Xi_n)\} \ .$$

We denote by τ[q-XTT] and λ[q-XTT] the classes of tree transformations and, respectively, translations definable by quasi-alphabetic XTT-transducers. □

Note that since the quasi-alphabetic tree transformations are incomparable with tree transformations computed by linear TOP-transducers, not allowing arbitrary patterns as left-hand sides of the rules, but only shallow patterns of the form $f(\xi_1, \ldots, \xi_m)$ for some symbol f of rank m would not have permitted us to compare such transducers with quasi-alphabetic tree bimorphisms. Next, we give another characterization of the quasi-alphabetic tree transformations and translations.

Theorem 6.2.34. $\tau[qTB] = \tau$[q-XTT], *and hence* $\lambda[qTB] = \lambda$[q-XTT]. *Both equalities are effective.*

Proof. Let $B = (\varphi, R, \psi)$ be a quasi-alphabetic tree bimorphism with $\varphi\colon T_\Gamma(Z) \to T_\Sigma(X)$, $R \subseteq T_\Gamma(Z)$ and $\psi\colon T_\Gamma(Z) \to T_\Omega(Y)$. By Proposition 6.2.5, we can assume without loss of generality that φ is normalized. Since R is recognizable, there is a ΓZ-ndT $TR = (Q, \Sigma, X, P, I)$ such that $R = T(TR)$.

Next, we construct the quasi-alphabetic XTT-transducer $XT = (Q, \Sigma, X, \Omega, Y, I, P')$, where the rules of P' are defined as follows:

- For every $q(z) \to z$ in P with $q \in Q$ and $z \in Z$, $q(\varphi_Z(z)) \to \psi_Z(z)$ is in P'.
- For every $q(e) \to e$ in P with $q \in Q$ and $e \in \Gamma_0$, $q(\varphi_0(e)) \to \psi_0(e)$ is in P'.
- For every $q(f(\xi_1, \ldots, \xi_m)) \to f(q_1(\xi_1), \ldots, q_m(\xi_m))$ in P with $m \geq 1$, $f \in \Gamma_m$, and $q, q_1, \ldots, q_m \in Q$, the rule $q(\varphi_m(f)) \to \psi_m(f)[q_1(\xi_1), \ldots, q_m(\xi_m)]$ is in P'.

To show that $\tau(XT) = \tau(TB)$, it is enough to prove that for every $q \in Q$, $s \in T_\Sigma(X)$ and $t \in T_\Omega(Y)$, $q(s) \Rightarrow^*_{XT} t$ if and only if there exists $r \in T_\Gamma(Z)$ such that $q(r) \Rightarrow^*_{TR} r$, $s = r\varphi$ and $t = r\psi$.

To this end, let $q \in Q$, $s \in T_\Sigma(X)$ and $t \in T_\Omega(Y)$ be such that there exists $r \in T_\Gamma(Z)$ with $q(r) \Rightarrow^*_{TR} r$, $s = r\varphi$ and $t = r\psi$. To show $q(s) \Rightarrow^*_{XT} t$, we proceed by tree induction on r.

(1) If $r = z$, then $q(z) \to z$ is in R, $s = \varphi_Z(z)$ and $t = \psi_Z(z)$. Hence, $q(\varphi_Z(z)) \to \psi_Z(z)$ is in P', which is obviously equivalent to $q(s) \Rightarrow^*_{XT} t$.

(2) If $r = f(r_1, \ldots, r_m)$ with $m \geq 0$, $f \in \Gamma_m$ and $r_1, \ldots, r_m \in T_\Gamma(Z)$, then

$$s = \varphi_m(f)[r_1\varphi, \ldots, r_m\varphi] \qquad \text{and} \qquad t = \psi_m(f)[r_1\psi, \ldots, r_m\psi] \ .$$

Since $q(r) \Rightarrow_{TR}^* r$, there is a rule

$$q(f(\xi_1, \ldots, \xi_m)) \to f(q_1(\xi_1), \ldots, q_m(\xi_m))$$

in P, and for every $i \in [m]$, $q_i(r_i) \Rightarrow_{TR}^* r_i$. By the induction hypothesis, $q_i(r_i\varphi) \Rightarrow_{XT}^* r_i\psi$ for every $i \in [m]$, and because there exists the rule

$$q(\varphi_m(f)) \to \psi_m(f)[q_1(\xi_1), \ldots, q_m(\xi_m)]$$

in P', we get $q(s) \Rightarrow_{XT}^* t$.

Next, let $q(s) \Rightarrow_{XT}^* t$ for some $q \in Q$, $s \in T_\Sigma(X)$ and $t \in T_\Omega(Y)$ and we proceed by induction on the number of steps of this derivation.

(1) If $q(s) \Rightarrow_{XT} t$, then there is a rule $q(s) \to t$ in P', which means that there exists $r \in Z \cup \Gamma_0$ such that $q(r) \to r$ is in P, $s = r\varphi$ and $t = r\psi$.

(2) Assume that the assertion holds for all derivations of at most $n-1$ steps, for some $n \geq 2$, and suppose that $q(s) \Rightarrow_{XT}^* t$ by an n-step derivation. Hence, there exists a rule $q(\varphi_m(f)) \to \psi_m(f)[q_1(\xi_1), \ldots, q_m(\xi_m)]$ in P' and, for every $i \in [m]$, there are derivations $q_i(s_i) \Rightarrow_{XT}^* t_i$, each in at most $n-1$ steps, such that $s = \varphi_m(f)[s_1, \ldots, s_m]$ and $t = \psi_m(f)[t_1, \ldots, t_m]$. By the inductive assumption, for every $i \in [m]$, there exists $r_i \in T_\Gamma(Z)$ such that $q_i(r_i) \Rightarrow_{TR}^* r_i$, $s_i = r_i\varphi$ and $t_i = r_i\psi$. If we set $r := f(r_1, \ldots, r_m)$, then $r\varphi = s$, $r\psi = t$ and $q(r) \Rightarrow_{TR}^* r$.

For the converse inclusion, let $XT = (Q, \Sigma, X, \Omega, Y, P, I)$ be a quasi-alphabetic tree transducer. We can assume without loss of generality that every rule in P has the left-hand side "normalized", i.e., the formal variables from Ξ appear in order. We construct a leaf alphabet Z such that $z = [q(x) \to y]$ is in Z for every rule $q(x) \to y$ in P, and a ranked alphabet Γ such that for every $m \geq 0$, Γ_m contains all the symbols of the form $[q(s) \to t]$, where $s \in \Sigma(X \cup \Xi_m) \cap C_\Sigma^m(X)$, $t \in \Omega(Y \cup Q(\Xi_m)) \cap C_{\Omega \cup Q}^m(Y)$ and $q(s) \to t \in P$.

We define two tree homomorphisms $\varphi\colon T_\Gamma(Z) \to T_\Sigma(X)$ and $\psi\colon T_\Gamma(Z) \to T_\Omega(Y)$, and a ΓZ-ndT $TR = (Q, \Gamma, Z, P', I)$ as follows.

- For every rule $q(x) \to y$ in P with $x \in X$ and $y \in Y$, let

$$\varphi_Z(z) := x, \qquad \psi_Z(z) := y \qquad \text{and} \qquad q(z) \to z \in P' \ .$$

- For every rule $q(s) \to t$ in P with $s \in \Sigma(X \cup \Xi_m) \cap C_\Sigma^m(X)$ and $t \in \Omega(Y \cup Q(\Xi_m)) \cap C_{\Omega \cup Q}^m(Y)$, let u be the unique tree in $T_\Omega(Y \cup \Xi_m) \cap C_\Omega^m(Y)$ such that t is a θ-instance of u, i.e., $t = u\theta$, where $\theta = \{(\xi_1, q_1(\xi_1)), \ldots, (\xi_m, q_m(\xi_m))\}$ is a $Q\Xi$-substitution. Then, $\varphi_m([q(s) \to t]) := s$, $\psi_m([q(s) \to t]) := u$, and

$$q([q(s) \to t](\xi_1, \ldots, \xi_m)) \to [q(s) \to t](q_1(\xi_1), \ldots, q_m(\xi_m)) \in P' \ .$$

Clearly, the tree bimorphism $TB = (\varphi, T(TR), \psi)$ is quasi-alphabetic, and hence it remains to prove that $\tau(XT) = \tau(TB)$. For this, it is enough to show that for every

$s \in T_\Sigma(X)$, $t \in T_\Omega(Y)$ and $q \in Q$, $q(s) \Rightarrow^*_{XT} t$ if and only if there exists $r \in T_\Gamma(Z)$ such that $q(r) \Rightarrow^*_{TR} r$, $s = r\varphi$ and $t = r\psi$.

Firstly, let $s \in T_\Sigma(X)$, $t \in T_\Omega(Y)$ and $q \in Q$ be such that there is $r \in T_\Gamma(Z)$ with $q(r) \Rightarrow^*_{TR} r$, $s = r\varphi$ and $t = r\psi$. To prove $q(s) \Rightarrow^*_{XT} t$, we proceed by tree induction on r.

(1) If $r \in Z$, then there are $x \in X$ and $y \in Y$ such that $r = [q(x) \to y]$ and $q(r) \to r$ is a rule in P'. Therefore, we have a rule $q(r\varphi) \to r\psi$ in P, and since $r\varphi = s$ and $r\psi = t$, we get $q(s) \Rightarrow^*_{XT} t$.

(2) If $r = [q(\bar{s}) \to \bar{t}](r_1, \ldots, r_m)$ for some $m \geq 0$, $r_1, \ldots, r_m \in T_\Gamma(Z)$ and $[q(\bar{s}) \to \bar{t}] \in \Gamma_m$ with $\bar{s} \in \Sigma(X \cup \Xi_m) \cap C^m_\Sigma(X)$, $\bar{t} \in \Omega(Y \cup Q(\Xi_m)) \cap C^m_{\Omega \cup Q}(Y)$ and $q(\bar{s}) \to \bar{t} \in P$, then

$$s = r\varphi = \bar{s}[r_1\varphi, \ldots, r_m\varphi] \qquad \text{and} \qquad t = r\psi = u[r_1\psi, \ldots, r_m\psi] \ ,$$

where $u \in T_\Omega(Y \cup \Xi_m) \cap C^m_\Omega(Y)$ is the unique tree such that $\bar{t} = u\theta$ for some $Q\Xi$-substitution $\theta = \{(\xi_1, q_1(\xi_1)), \ldots, (\xi_m, q_m(\xi_m))\}$. Since $q(r) \Rightarrow^*_{TR} r$, there is a rule

$$q([q(\bar{s}) \to \bar{t}](\xi_1, \ldots, \xi_m)) \to [q(\bar{s}) \to \bar{t}](q_1(\xi_1), \ldots, q_m(\xi_m))$$

in P', and for every $i \in [m]$, $q_i(r_i) \Rightarrow^*_{TR} r_i$. By the induction assumption, $q_i(r_i\varphi) \Rightarrow^*_{XT} r_i\psi$ for every $i \in [m]$. Hence, we obviously get $q(r\varphi) \Rightarrow^*_{XT} r\psi$, which is equivalent to $q(s) \Rightarrow^*_{XT} t$.

Now, let $q(s) \Rightarrow^*_{XT} t$ for some $q \in Q$, $s \in T_\Sigma(X)$ and $t \in T_\Omega(Y)$, and we proceed by induction on the number of steps of this derivation.

(1) If $q(s) \Rightarrow_{XT} t$, then there is a rule $q(s) \to t$ in P, which means that there exists $r = [q(s) \to t] \in Z \cup \Gamma_0$ such that $q(r) \to r$ is in P', $s = r\varphi$ and $t = r\psi$.

(2) Assume that the assertion holds for all derivations of at most $n-1$ steps, for some $n \geq 2$, and suppose that $q(s) \Rightarrow^*_{XT} t$ by a derivation of n steps. Hence, there exist $\bar{s} \in \Sigma(X \cup \Xi_m) \cap C^m_\Sigma(X)$, $\bar{t} \in \Omega(Y \cup Q(\Xi_m)) \cap C^m_{\Omega \cup Q}(Y)$ and the rule $q(\bar{s}) \to \bar{t} \in R$ such that $s = \bar{s}[s_1, \ldots, s_m]$ and $t = u[t_{\sigma^{-1}(1)}, \ldots, t_{\sigma^{-1}(m)}]$, where $s_1, \ldots, s_m \in T_\Sigma(X)$, $t_1, \ldots, t_m \in T_\Omega(Y)$, σ is the permutation on $[m]$ giving the order in which the variables $\xi_1, \ldots, \xi_m$ appear in $\bar{t}$, and $u \in T_\Omega(Y \cup \Xi_m) \cap C^m_\Omega(Y)$ is the unique tree such that $\bar{t} = u\theta$ for some $Q\Xi$-substitution $\theta = \{(\xi_1, q_1(\xi_1)), \ldots, (\xi_m, q_m(\xi_m))\}$. Moreover, for every $i \in [m]$, there are derivations $q_i(s_i) \Rightarrow^*_{XT} t_i$ each in at most $n-1$ steps. Then by induction assumption, for every $i \in [m]$, there is $r_i \in T_\Gamma(Z)$ such that $q(r_i) \Rightarrow^*_{TR} r_i$, $s_i = r_i\varphi$ and $t_i = r_i\psi$. If we set $r := [q(\bar{s}) \to \bar{t}](r_1, \ldots, r_m)$, then clearly we get $q(r) \Rightarrow^*_{TR} r$, $s = r\varphi$ and $t = r\psi$.

This concludes the proof. □

The theorem above gives us the possibility to switch, when the need arises, between algebraic and machine-oriented techniques in tackle problems.

Note that to simulate a quasi-alphabetic tree transducer by a quasi-alphabetic tree bimorphism, the alphabet we construct depends on the tree transducer, and hence it is variable. Though, an alternative is to use the canonical representation of quasi-alphabetic tree bimorphisms of Theorem 6.2.9 and act as follows (cf. also Rahonis, 2001, Section 3). First, the tree transducer simulates the inverse quasi-alphabetic tree homomorphism $(\pi^1)^{-1}$

whereas its rules define a $[\Sigma \times \Omega](X \times Y)$-ndT tree recognizer that accepts the tree language R. Then it computes the quasi-alphabetic tree homomorphism π^2. Thus, the product alphabet becomes a part of the computation mode of the quasi-alphabetic XTT-transducer as an intermediate alphabet.

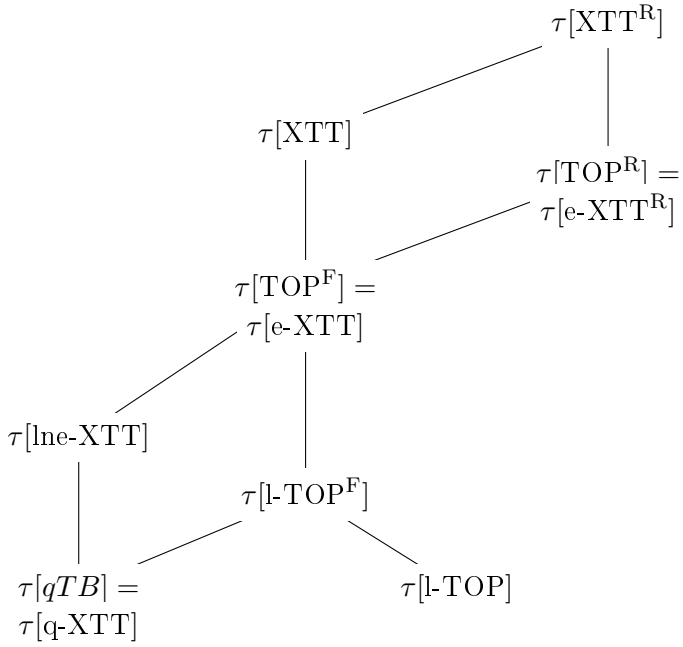


Figure 6.2.3: HASSE diagram representing the inclusion relations among well-known classes of tree transformations computed by tree transducers and quasi-alphabetic tree transformations.

e

As a immediate consequence of Theorem 6.2.34, we get a rough upper bound to the power of quasi-alphabetic tree bimorphisms because every quasi-alphabetic XTT-transducer is clearly linear, non-deleting and epsilon-free.

Corollary 6.2.35. $\tau[qTB] \subset \tau$[lne-XTT].

Moreover, because the image of a recognizable tree language via a quasi-alphabetic tree homomorphism is still a recognizable language, we get.

Proposition 6.2.36. *The domain and range of a quasi-alphabetic extended top-down tree transducer are recognizable tree languages.*

Finally let us gather all the connections presented so far into a HASSE diagram. This offers a complete picture on the power of quasi-alphabetic tree transformations when compared with some well-known classes of tree transformations defined by tree transducers.

Theorem 6.2.37. *The inclusion relations between quasi-alphabetic tree transformations and certain tree transformation classes defined by tree transducers are given by the* HASSE *diagram of Figure 6.2.3.*

Proof. Follows from Propositions 6.2.32 and 6.2.31, Corollary 6.2.35 and Figure 5.2.3. □

6.3 Alphabetic and permuting tree bimorphisms

Alphabetic tree transformations and permuting tree transformations were originally called *primitive transformations on trees* and, respectively, *primitive transformations with permutations* by Takahashi (1972). In this section, we recall their basic properties (Takahashi, 1972, Bozapalidis, 1992), investigate their translation power and relation with synchronous grammars (Maletti and Tîrnăucă, 2009), and connect them with the tree transducer classes defined by Gazdag (2006b). We start by fixing some notation and terminology.

Definition 6.3.1. The members of $\tau[\mathrm{aH}, \mathrm{Rec}, \mathrm{aH}]$ and $\lambda[\mathrm{aH}, \mathrm{Rec}, \mathrm{aH}]$ are called *alphabetic tree transformations* and *alphabetic translations*, respectively. Similarly, we refer to $\tau[\mathrm{pH}, \mathrm{Rec}, \mathrm{pH}]$ and $\lambda[\mathrm{pH}, \mathrm{Rec}, \mathrm{pH}]$ as the classes of *permuting tree transformations* and *permuting translations*. The classes $\tau[\mathrm{pH}, \mathrm{Rec}, \mathrm{pH}]$, $\lambda[\mathrm{pH}, \mathrm{Rec}, \mathrm{pH}]$, $\tau[\mathrm{aH}, \mathrm{Rec}, \mathrm{aH}]$ and $\lambda[\mathrm{aH}, \mathrm{Rec}, \mathrm{aH}]$ are denoted simply $\tau[pTB]$, $\lambda[pTB]$, $\tau[aTB]$ and $\lambda[aTB]$, respectively. □

6.3.1 Properties

As a direct consequence of Theorem 6.2.12, we find that permuting tree transformations are not closed under intersection. Moreover, it is clear that every alphabetic tree homomorphism is permuting, and every permuting tree homomorphism is quasi-alphabetic. Thus, we can note the following.

Proposition 6.3.2. $\tau[aTB] \subseteq \tau[pTB] \subseteq \tau[qTB]$.

Next, we list the main properties of alphabetic and permuting tree transformations proved by Takahashi (1972, Theorems 4, 5 and 6) and Bozapalidis (1992, Section 6). Note that some of them follow directly from our Proposition 6.3.2.

Theorem 6.3.3. *The following hold.*

(i) **Closure properties.** *The classes $\tau[aTB]$ and $\tau[pTB]$ are closed under inverses, union and composition, but not under intersection.*

(ii) *For any $R, R' \in \mathrm{Rec}$, the tree transformation $\{(r, r') \in R \times R' \mid \mathrm{dom}(r) = \mathrm{dom}(r')\}$ is permuting, and therefore, alphabetic.*

(iii) *The classes $\tau[aTB]$ and $\tau[pTB]$ preserve the recognizability of tree languages.*

(iv) **Decidability results.** *The equality is decidable for alphabetic tree transformations.*

(v) **Canonical representation.** *A tree transformation $\tau \subseteq T_\Sigma(X) \times T_\Omega(Y)$ is alphabetic if and only if there is an $R \in \mathrm{Rec}_{\Sigma\times\Omega}(X \times Y)$ such that $\tau = \{(t\pi^1, t\pi^2) \mid t \in R\}$, where for any $m \geq 0$, $(\Sigma \times \Omega)_m := \Sigma_m \times \Omega_m$, and $\pi^1 \colon T_{\Sigma\times\Omega}(X \times Y) \to T_\Sigma(X)$ and $\pi^2 \colon T_{\Sigma\times\Omega}(X \times Y) \to T_\Omega(Y)$ are the projections defined by*

$$\pi^1_{X\times Y}(x, y) := x \quad \text{and} \quad \pi^2_{X\times Y}(x, y) := y \quad \text{for every} \quad (x, y) \in X \times Y, \text{and}$$
$$\pi^1_{\Sigma\times\Omega}(f, g) := f \quad \text{and} \quad \pi^2_{\Sigma\times\Omega}(f, g) := g \quad \text{for every} \quad (f, g) \in \Sigma \times \Omega \ .$$

6.3.2 Translation power

In this section, we show that the SDTSs in strong normal form of Theorem 3.3.12(v) define the same translations as the permuting tree bimorphisms (Maletti and Tîrnăucă, 2009). However, compared with Theorem 6.2.25, a leaf alphabet must be allowed for the center tree language of such tree bimorphisms. Moreover, we prove that simple SDTS correspond to alphabetic tree bimorphisms. The latter result shows that alphabetic tree bimorphisms are strictly less powerful than SDTSs.

Let us first consider the direction in which we construct a tree bimorphism for an SDTS. Since the only difference between quasi-alphabetic and permuting tree bimorphisms is in their homomorphisms, let us reconsider the construction of the tree homomorphisms φ^{SD} and ψ^{SD} given in the beginning of Section 6.2.2. We only change the behavior on productions that have just terminal symbols on the right-hand side.

Definition 6.3.4. Let $SD = (N, X, Y, P, S)$ be an SDTS in strong normal form, and let e be a new leaf symbol. For every production $p = A; A \to \alpha; \beta\,(\sigma) \in P$ let $\mathrm{rk}(p)$ be the number of nonterminals in α. This turns the set P into a ranked alphabet. Moreover, let $P' := \bigcup_{m \geqslant 1} P_m$. We construct the tree homomorphisms

$$\varphi^{SD} \colon T_{P'}(P_0) \to T_{P'}(X') \qquad \text{and} \qquad \psi^{SD} \colon T_{P'}(P_0) \to T_{P'}(Y') \ ,$$

where $X' := X \cup \{e\}$ and $Y' := Y \cup \{e\}$, as follows. Let $p = A; A \to \alpha; \beta\,(\sigma) \in P$.

- If $p \in P_0$, then

$$\varphi_0^{SD}(p) := \begin{cases} e & \text{if } \alpha = \varepsilon \\ \alpha & \text{if } \alpha \in X \end{cases} \qquad \text{and} \qquad \psi_0^{SD}(p) := \begin{cases} e & \text{if } \beta = \varepsilon \\ \beta & \text{if } \beta \in Y. \end{cases}$$

- If $m \geq 1$ and $p \in P'_m$, then

$$\varphi_m^{SD}(p) := p(\xi_1, \ldots, \xi_m) \qquad \text{and} \qquad \psi_m^{SD}(p) := p(\xi_{\sigma(1)}, \ldots, \xi_{\sigma(m)}) \ ,$$

  where σ is the permutation of p. Thus, we constructed the permuting tree bimorphism $TB_{SD} := (\varphi^{SD}, T_{P'}(P_0), \psi^{SD})$. Furthermore, we define the translation of TB_{SD} as the relation

$$\lambda(TB_{SD}) := \{(\mathrm{yd}_{X \setminus \{e\}}(s), \mathrm{yd}_{Y \setminus \{e\}}(t)) \mid (s,t) \in \tau(TB_{SD})\} \ .$$

  Note the special treatment of e. It is never output but acts as the empty string. The constructed tree homomorphisms φ^{SD} and ψ^{SD} are permuting, and if SD is simple, then they are alphabetic. □

By Theorem 3.3.12(v) we can assume strong normal form without loss of generality. Thus, using Definition 6.3.4, a minor modification of the proof of Proposition 6.2.23 yields the following.

Lemma 6.3.5. *For every SDTS SD in strong normal form, there exists a permuting tree bimorphism TB such that $\lambda(SD) = \lambda(TB)$. If SD is simple, then TB can be chosen to be alphabetic.* □

For the converse, we reconsider Proposition 6.2.24, which states that for every quasi-alphabetic tree bimorphism TB there exists an SDTS SD such that $\lambda(SD) = \lambda(TB)$. It is

easy to see that the SDTS constructed in Proposition 6.2.24 is simple if *TB* is alphabetic. The minor modification of the definition of the translation defined by a tree bimorphism (i.e., the special treatment of the symbol e in Definition 6.3.4) requires only a minor change in the proof of Proposition 6.2.24. Therefore, using Proposition 6.3.2, we get.

Lemma 6.3.6. *For every permuting tree bimorphism* $TB \in \mathbf{B}[\mathrm{pH}, \mathrm{Rec}, \mathrm{pH}]$*, there exists an SDTS SD such that* $\lambda(SD) = \lambda(TB)$*. If TB is alphabetic, then SD can be chosen to be simple.* □

Lemmata 6.3.5 and 6.3.6 yield the following relations between SDTSs and permuting tree bimorphisms, which sharpen the result of Theorem 6.2.25. However, in the tree bimorphisms used in this alternative characterization of the syntax-directed translations, the tree homomorphisms belong to a proper subclass of qH, but in return a leaf alphabet must be allowed for the center tree languages.

Theorem 6.3.7. *The following hold effectively.*

(i) $\lambda[SDTS] = \lambda[pTB]$.

(ii) $\lambda[sSDTS] = \lambda[aTB]$.

If we now consider Theorems 3.3.22 and 6.3.7 together, we obtain as a consequence the following relation between alphabetic and permuting tree bimorphisms.

Corollary 6.3.8. $\lambda[aTB] \subset \lambda[pTB]$. □

6.3.3 Implementation by tree transducers

Here, using known results from the literature, we look for the machine-correspondents of alphabetic and permuting tree bimorphisms, building for the first time a bridge between the two classes of devices.

We start with tree transformations defined by alphabetic tree bimorphisms. Using Definition 5.2.12 and the canonical representation of the class $\tau[aTB]$ given by Theorem 6.3.3(v), a much simplified version of the proof of Theorem 7 of Rahonis (2001) shows that finite-state relabeling TOP-transducers compute exactly the class of alphabetic tree transformations. Therefore, we have.

Theorem 6.3.9. $\tau[aTB] = \mathrm{QREL}$*, and hence* $\lambda[aTB] = \lambda[\mathrm{QREL}]$*. Both equalities are effective.*

Now, using Theorems 2.64 and 3.70 of Gazdag (2006b), we get another machine characterization of the class $\tau[aTB]$: top-down or bottom-up shape preserving tree transducers compute exactly the class of alphabetic tree transformations.

Next, we turn our attention to the tree transducer counterpart of the permuting tree bimorphisms (cf. Gazdag, 2006b, Section 2.1).

Definition 6.3.10. A TOP-transducer $TD = (Q, \Sigma, X, \Omega, Y, P, I)$ is *permuting* if each rule in P has one of the following forms:

(1) $q(x) \to y$ with $q \in Q$, $x \in X$ and $y \in Y$;

(2) $q(e) \to e'$ with $q \in Q$, $e \in \Sigma_0$ and $e' \in \Omega_0$;

(3) $q(f(\xi_1,\ldots,\xi_m)) \to g(q_1(\xi_{\sigma(1)}),\ldots,q_m(\xi_{\sigma(m)}))$, where $m \geq 1$, σ is a permutation of $[m]$, $f \in \Sigma_m$, $q_1,\ldots,q_m,q \in Q$ and $g \in \Omega_m$. □

Thus, using just the definitions, a slight change of the first part of the proof of Theorem 6.2.34 shows that we can construct a permuting tree transducer TD for any given permuting tree bimorphism, which computes the same tree transformation.

On the other hand, given any permuting TOP-transducer $TD = (Q, \Sigma, X, \Omega, Y, P, I)$ we can build:

- a leaf alphabet Z such that $z := [q(x) \to y]$ is in Z for every rule $q(x) \to y$ in P,
- a ranked alphabet Γ such that for every $m \geq 0$,

$$\Gamma_m := \{[q(s) \to t] \mid q(s) \to t \in P \text{ with } s \in \Sigma(\Xi_m) \cap \tilde{T}_\Sigma(\Xi_m)\} \ ,$$

- a ΓZ-ndT $TR = (Q, \Gamma, Z, P', I)$, where
  - $q(z) \to z \in P'$ for every rule $q(x) \to y$ in P with $x \in X$ and $y \in Y$ and
  - $q([q(s) \to t](\xi_1,\ldots,\xi_m)) \to [q(s) \to t](q_1(\xi_1),\ldots,q_m(\xi_m)) \in P'$ for every rule $q(s) \to t$ in P with $s \in \Sigma(\Xi_m) \cap \tilde{T}_\Sigma(\Xi_m)$ and $t \in \Omega(Q(\Xi_m)) \cap C^m_{\Omega\cup Q}(\emptyset)$, and
- two tree homomorphisms $\varphi\colon T_\Gamma(Z) \to T_\Sigma(X)$ and $\psi\colon T_\Gamma(Z) \to T_\Omega(Y)$ such that
  - $\varphi_Z(z) := x$ and $\psi_Z(z) := y$ for every $z = [q(x) \to y] \in Z$, and
  - for any $m \geq 0$ and $[q(s) \to t] \in \Gamma_m$,

$$\varphi_m([q(s) \to t]) := s \quad \text{and} \quad \psi_m([q(s) \to t]) := u \ ,$$

where $u \in T_\Omega(\Xi_m) \cap C^m_\Omega(\emptyset)$ is the unique tree such that $t = \theta(u)$ for some $Q\Xi$-substitution $\theta = \{(\xi_1, q_1(\xi_{\sigma(1)})),\ldots,(\xi_m, q_m(\xi_{\sigma(m)}))\}$.

Now, a slight modification of the second part of the proof of Theorem 6.2.34 shows that the permuting tree bimorphism $(\varphi, T(TR), \psi)$ defines the same tree transformation as TD. Therefore, we can state.

Theorem 6.3.11. *A tree transformation is permuting if and only if it is computed by a permuting top-down tree transducer.*

6.4 Linear non-deleting tree bimorphisms

In this section we present the class $\mathbf{B}[\text{lnH}, \text{Rec}, \text{lnH}]$ of linear non-deleting tree bimorphisms, that was first studied by Arnold and Dauchet (1976a) and lately received an increased interest by both formal language and NLP communities because of its relation with XTT-transducers (Arnold and Dauchet, 1982, Maletti, 2007, 2008) and synchronous grammars (Shieber, 2004, cf. also Theorem 6.4.8). The properties of this bimorphism class were investigated by Arnold and Dauchet (1976b, a, 1982), Dauchet and Mongy (1979), Dauchet (1975) and Maletti (2007, 2008) (cf. Maletti, 2010a, for a brief survey). In what follows, we may denote $\tau[\text{lnH}, \text{Rec}, \text{lnH}]$ and $\lambda[\text{lnH}, \text{Rec}, \text{lnH}]$ simply by $\tau[lnTB]$ and $\lambda[lnTB]$.

The main properties of linear non-deleting tree bimorphisms are presented next.

Theorem 6.4.1. *The following hold.*

(i) *The class $\tau[lnTB]$ is closed under union and inverses.*

(ii) *Any tree transformation in $\tau[lnTB]$ preserves the recognizability of tree languages.*

(iii) *The class $\tau[lnTB]$ is not closed under composition.*

Since every quasi-alphabetic tree homomorphism is linear and non-deleting, we get.

Proposition 6.4.2. *$\tau[qTB] \subseteq \tau[lnTB]$, and hence $\lambda[qTB] \subseteq \lambda[lnTB]$.*

Now, we study the translation power of linear non-deleting tree bimorphisms (cf. Shieber, 2004). To this end, we prove that GSTSGs are the natural generative counterpart of this tree bimorphism class, thus obtaining new characterizations of tree transformations and translations definable by GSTSGs (cf. Shieber, 2004, Knight, 2007, Maletti, 2008, 2010a, for example). By using Corollary 5.3.9, we obtain new descriptions of the class $\lambda[SDTS] = \lambda[qTB] = \mathcal{SDT}$.

Firstly, as it was done in Section 6.2.2 for quasi-alphabetic tree bimorphisms, we represent derivations of a GSTSG $GS = (N, \Sigma, X, N', \Omega, Y, P, S, S')$ by 'production trees' which now are defined as follows. Let Γ^{GS} be the ranked alphabet such that for any $m \geq 0$,

$$\Gamma^{GS}_m := \{[A; A' \to r; r'(\sigma)] \mid A; A' \to r; r'(\sigma) \in P, |\mathrm{yd}_N(r)| = m\} .$$

Next, we define for every pair $(A, A') \in N \times N'$ a set $P(GS, A, A')$ inductively as follows:

(1) if $[A; A' \to r; r'] \in \Gamma^{GS}_0$, then $[A; A' \to r; r'] \in P(GS, A, A')$;

(2) if $[A; A' \to r; r'(\sigma)] \in \Gamma^{GS}_m$ for some $m \geq 1$, $\mathrm{yd}_N(r) = A_1 \dots A_m$ and $\mathrm{yd}_{N'}(r') = A'_1 \dots A'_m$, then
$$[A; A' \to r; r'(\sigma)](t_1, \dots, t_m) \in P(GS, A, A')$$
for all $t_1 \in P(GS, A_1, A'_{\sigma^{-1}(1)}), \dots, t_m \in P(GS, A_m, A'_{\sigma^{-1}(m)})$.

The set of *production trees* of GS is now defined as the Γ^{GS}-tree language $P(GS) := P(GS, S, S')$. The following fact is easily verified (cf. Section 6.2.2).

Lemma 6.4.3. *$P(GS) \in \mathrm{Loc} \cap \mathrm{DRec}^{\mathrm{vf}}$.*

Let us now present one half of our characterization of GSTSGs by tree bimorphisms that both generalizes and strengthens a result claimed by Shieber (2004).

Proposition 6.4.4. *$\tau[GSTSG] \subseteq \tau[\mathrm{lnH}, \mathrm{Loc} \cap \mathrm{DRec}^{\mathrm{vf}}, \mathrm{lnH}]$.*

Proof. Consider any $GS = (N, \Sigma, X, N', \Omega, Y, P, S, S')$. Let us define two tree homomorphisms $\varphi\colon T_{\Gamma^{GS}} \to T_\Sigma(X)$ and $\psi\colon T_{\Gamma^{GS}} \to T_\Omega(Y)$ by the following mappings φ_m and ψ_m $(m \geq 0)$:

(1) for any $e = [A; A' \to r; r'] \in \Gamma^{GS}_0$, let $\varphi_0(e) := r$ and $\psi_0(e) := r'$;

(2) for any $f = [A; A' \to r; r'(\sigma)] \in \Gamma^{GS}_m$, where $m \geq 1$, $\mathrm{yd}_N(r) = A_1 \dots A_m$ and $\mathrm{yd}_{N'}(r') = A'_1 \dots A'_m$, let

$$\varphi_m(f) := r(A_1 \leftarrow \xi_1, \dots, A_m \leftarrow \xi_m)$$

and

$$\psi_m(f) := r'(A'_1 \leftarrow \xi_{\sigma(1)}, \ldots, A'_m \leftarrow \xi_{\sigma(m)}) \ ,$$

where $A \leftarrow \xi$ denotes the replacement of the occurrence of A by the variable ξ.

We shall show that the tree bimorphism $TB := (\varphi, P(GS), \psi)$ defines the same tree transformation as GS. For this we need the following two lemmas. The first one can be verified by tree induction on t following the inductive definition of the sets $P(GS, A, A')$ considering simultaneously all pairs $(A, A') \in N \times N'$ (cf. Lemma 6.2.20).

Lemma A. If $t \in P(GS, A, A')$ for some $A \in N$ and $A' \in N'$, then $(A, A') \Rightarrow^*_{GS} (t\varphi, t\psi)$.

The proof of the second lemma can be carried out by induction on the length of the derivation considering again all pairs $(A, A') \in N \times N'$ in parallel (cf. Lemma 6.2.21).

Lemma B. If $(A, A') \Rightarrow^*_{GS} (s, s')$ for some $A \in N$, $A' \in N'$, $s \in T_\Sigma(X)$ and $s' \in T_\Omega(Y)$, then there exists a tree $t \in P(GS, A, A')$ such that $s = t\varphi$ and $s' = t\psi$.

The inclusion $\tau(TB) \subseteq \tau(GS)$ is obtained by applying Lemma A to the pair (S, S'), i.e., to production trees. The converse inclusion follows similarly from Lemma B. Hence, it suffices to note that $TB \in \mathbf{B}[\mathrm{lnH}, \mathrm{Loc} \cap \mathrm{DRec}^{\mathrm{vf}}, \mathrm{lnH}]$. Indeed, the tree homomorphisms φ and ψ are linear and non-deleting, and $P(GS)$ is in $\mathrm{Loc} \cap \mathrm{DRec}^{\mathrm{vf}}$ by Lemma 6.4.3. □

Corollary 6.4.5. $\lambda[GSTSG] \subseteq \lambda[\mathrm{lnH}, \mathrm{Loc} \cap \mathrm{DRec}^{\mathrm{vf}}, \mathrm{lnH}]$.

Let us now prove the converse of Proposition 6.4.4 in a slightly strengthened form.

Proposition 6.4.6. $\tau[\mathrm{lnH}, \mathrm{Rec}, \mathrm{lnH}] \subseteq \tau[GSTSG]$.

Proof. Let $TB = (\varphi, R, \psi)$ be a tree bimorphism such that $R \in \mathrm{Rec}_\Gamma(Z)$, and $\varphi\colon T_\Gamma(Z) \to T_\Sigma(X)$ and $\psi\colon T_\Gamma(Z) \to T_\Omega(Y)$ are linear non-deleting tree homomorphisms with φ normalized. Furthermore, let $RT = (N, \Gamma, Z, P, S)$ be a regular ΓZ-tree grammar in normal form generating R. Let $GS = (N, \Sigma, X, N, \Omega, Y, P', S, S)$ be the GSTSG, where P' is constructed from P and TB as follows:

(1) For every production $A \to z$ in P such that $A \in N$ and $z \in Z$, we include in P' the production $A; A \to \varphi_Z(z); \psi_Z(z)$. Similarly, if $A \to e$ is a production in P such that $A \in N$ and $e \in \Gamma_0$, we add to P' the production $A; A \to \varphi_0(e); \psi_0(e)$.

(2) If P contains a production $A \to f(A_1, \ldots, A_m)$, where $m \geq 1$, $f \in \Gamma_m$ and $A, A_1, \ldots, A_m \in N$, we add to P' the production

$$A; A \to \varphi_m(f)[A_1, \ldots, A_m]; \psi_m(f)[A_1, \ldots, A_m]\,(\sigma) \ ,$$

where σ is the permutation on $[m]$ such that $\mathrm{yd}_{\Xi_m}(\psi_m(f)) = \xi_{\sigma(1)} \ldots \xi_{\sigma(m)}$.

To prove the inclusion $\tau(TB) \subseteq \tau(GS)$, we first show that for any $A \in N$ and $t \in T_\Gamma(Z)$, if $A \Rightarrow^*_{RT} t$, then $(A, A) \Rightarrow^*_{GS} (t\varphi\,, t\psi)$. We proceed by tree induction on t.

(1) If $A \Rightarrow^*_{RT} z$ for some $z \in Z$, then $A \to z$ is in P, and hence $(A, A) \Rightarrow^*_{GS} (z\varphi\,, z\psi)$ because P' contains the production $A; A \to \varphi_Z(z); \psi_Z(z)$. The case $t = e$, where $e \in \Gamma_0$, is similar.

(2) If $t = f(t_1, \ldots, t_m)$ for some $m \geq 1$, $f \in \Gamma_m$ and $t_1, \ldots, t_m \in T_\Gamma(Z)$, then P contains a production $A \to f(A_1, \ldots, A_m)$ such that $A_i \Rightarrow_{RT}^* t_i$ for every $i \in [m]$. By the induction hypothesis, $(A_i, A_i) \Rightarrow_{GS}^* (t_i\varphi, t_i\psi)$ for every $i \in [m]$, and by first applying the production

$$A; A \to \varphi_m(f)[A_1, \ldots, A_m]; \psi_m(f)[A_1, \ldots, A_m]\,(\sigma)$$

of GS that corresponds to $A \to f(A_1, \ldots, A_m)$, we get

$$(A, A) \Rightarrow_{GS}^* (\varphi_m(f)[t_1\varphi, \ldots, t_m\varphi], \psi_m(f)[t_1\psi, \ldots, t_m\psi]) = (t\varphi, t\psi)\ .$$

Now, if $(s, t) \in \tau(TB)$, for some $s \in T_\Sigma(X)$ and $t \in T_\Omega(Y)$, then there is an $r \in T(RT)$ such that $s = r\varphi$ and $t = r\psi$. Since $S \Rightarrow_{RT}^* r$, we get $(S, S) \Rightarrow_{GS}^* (r\varphi, r\psi) = (s, t)$, i.e., $(s, t) \in \tau(GS)$.

To prove the converse inclusion $\tau(GS) \subseteq \tau(TB)$, we first show that if $(A, A) \Rightarrow_{GS}^* (s, t)$ for some $A \in N$, $s \in T_\Sigma(X)$ and $t \in T_\Omega(Y)$, then there exist an $r \in T_\Gamma(Z)$ such that $A \Rightarrow_{RT}^* r$, $s = r\varphi$ and $t = r\psi$. We proceed by induction on the length of the derivation witnessing $(A, A) \Rightarrow_{GS}^* (s, t)$.

(1) If $(A, A) \Rightarrow_{GS} (s, t)$ in one step, then P' must contain the production $A; A \to s; t$. This means that there is a production $A \to r$ in P, where $r \in Z \cup \Gamma_0$, such that $s = r\varphi$ and $t = r\psi$. Hence, this r is the required ΓZ-tree.

(2) Let $(A, A) \Rightarrow_{GS}^* (s, t)$ by a derivation of length $n \geq 2$ and assume that the assertion holds for all shorter derivations. The first production

$$A; A \to \varphi_m(f)[A_1, \ldots, A_m]; \psi_m(f)[A_1, \ldots, A_m]\,(\sigma)$$

applied in the derivation is obtained from some production $A \to f(A_1, \ldots, A_m)$ in P. Moreover, for all $i \in [m]$, there is a derivation $(A_i, A_i) \Rightarrow_{GS} \ldots \Rightarrow_{GS} (s_i, t_i)$ of length at most $n - 1$ such that $s = \varphi_m(f)[s_1, \ldots, s_m]$ and $t = \psi_m(f)[t_1, \ldots, t_m]$. By the induction hypothesis, for every $i \in [m]$, there is a tree $r_i \in T_\Gamma(Z)$ such that $A_i \Rightarrow_{RT}^* r_i$, $s_i = r_i\varphi$ and $t_i = r_i\psi$. For $r := f(r_1, \ldots, r_m)$, we have $A \Rightarrow_{RT}^* r$, $r\varphi = \varphi_m(f)[r_1\varphi, \ldots, r_m\varphi] = s$ and $r\psi = \psi_m(f)[r_1\psi, \ldots, r_m\psi] = t$.

Now, if $(s, t) \in \tau(GS)$, then $(S, S) \Rightarrow_{GS}^* (s, t)$, and hence there exists an $r \in T_\Gamma(Z)$ such that $S \Rightarrow_{RT}^* r$, $s = r\varphi$ and $t = r\psi$. Since $r \in T(RT) = R$, we get $(s, t) \in \tau(TB)$. □

Corollary 6.4.7. $\lambda[\mathrm{lnH}, \mathrm{Rec}, \mathrm{lnH}] \subseteq \lambda[GSTSG]$.

Because the inclusions $\tau[\mathrm{lnH}, \mathrm{Loc} \cap \mathrm{DRec}^{\mathrm{vf}}, \mathrm{lnH}] \subseteq \tau[\mathrm{lnH}, \mathrm{Loc}, \mathrm{lnH}] \subseteq \tau[\mathrm{lnH}, \mathrm{Rec}, \mathrm{lnH}]$ are all obvious (by Proposition 4.5.5 and the trivial fact that $\varphi\psi \in \mathrm{lnH}$ for all $\varphi \in \mathrm{aH}$ and $\psi \in \mathrm{lnH}$), we can gather the results of Propositions 6.4.6 and 6.4.4 into the following (cf. also Corollary 5.3.9 and Proposition 5.3.8).

Theorem 6.4.8. *The following equalities hold effectively.*

(i) $\tau[\mathrm{lnH}, \mathrm{Loc} \cap \mathrm{DRec}^{\mathrm{vf}}, \mathrm{lnH}] = \tau[\mathrm{lnH}, \mathrm{Loc}, \mathrm{lnH}] = \tau[\mathrm{lnH}, \mathrm{Rec}, \mathrm{lnH}] = \tau[GSTSG] = \tau[\text{ln-XTT}]$.

(ii) $\lambda[\mathrm{lnH}, \mathrm{Loc} \cap \mathrm{DRec}^{\mathrm{vf}}, \mathrm{lnH}] = \lambda[\mathrm{lnH}, \mathrm{Loc}, \mathrm{lnH}] = \lambda[\mathrm{lnH}, \mathrm{Rec}, \mathrm{lnH}] = \lambda[GSTSG] = \lambda[\text{ln-XTT}] = \lambda[STSG] = \lambda[SDTS] = \mathcal{SDT}$.

The connection between GSTSGs and tree bimorphisms mentioned in the theorem above can be used to improve the the mathematical background of such synchronous grammars. For example, we can easily show.

Corollary 6.4.9. (i) *$\tau[GSTSG]$ is not closed under composition, but it is closed under inverses and union.*

(ii) *$\tau[GSTSG]$ preserves the recognizability of tree languages.*

(iii) *The emptiness and finiteness are decidable for $\tau[GSTSG]$.*

Proof. For (i) and (ii), it is enough to use Theorems 6.4.1 and 6.4.8. Note that closure under union of $\tau[GSTSG]$ can be alternatively shown by slightly adapting the direct construction given after Corollary 6.2.10 for quasi-alphabetic tree transformations. Now, let GS be any GSTSG, and assume $\tau(GS) = \tau(TB) = \varphi^{-1} \circ \mathrm{id}_R \circ \psi$ for some linear non-deleting tree bimorphism $TB = (\varphi, R, \psi)$. The decidability of questions in item (ii) follows from the fact that $\tau(GS)$ is empty, finite or infinite depending on whether R is empty, finite or infinite, and that these questions are decidable for recognizable tree languages by Theorem4.4.17(vi). □

More theoretical questions related to GSTSG and the class of tree transformation they define are proposed for further study in Section 8.2.1. The tree bimorphism approach could be the key approach in solving most of them.

6.5 Fine tree bimorphisms

Fine tree bimorphisms and the tree transformation they define were introduced by Arnold and Dauchet (1978a) and later systematically studied by Bozapalidis (1992) and by Bozapalidis and Rahonis (1994). Next, their connection with tree transducers was investigated by Rahonis (2001). In this section we overview the main results of these papers. Moreover, in Section 6.6, we will show that they have the same translation power as quasi-alphabetic tree bimorphisms. We start by fixing some notation and terminology.

Definition 6.5.1. The elements in $\tau[\mathrm{fH}, \mathrm{Rec}, \mathrm{fH}]$ are called *fine tree transformations*, and similarly, the members of $\lambda[\mathrm{fH}, \mathrm{Rec}, \mathrm{fH}]$ are named *fine translations*. In what follows, we may denote $\tau[\mathrm{fH}, \mathrm{Rec}, \mathrm{fH}]$ and $\lambda[\mathrm{fH}, \mathrm{Rec}, \mathrm{fH}]$ simply by $\tau[fTB]$ and $\lambda[fTB]$, respectively. □

Now we present the *supremum ranked alphabet* of Bozapalidis (1992, Section 2). This operation of concatenating ranked symbols is useful when we investigate trees over different ranked alphabets and of possibly different shapes. In particular, it was successfully used to prove closure properties by Bozapalidis (1992) and that various classes of tree bimorphisms (alphabetic, fine or quasi-alphabetic, for example) admit canonical representations (Theorems 6.3.3, 6.5.4 and 6.2.9). This characterization of the tree transformations defined by such tree bimorphisms is also essential in the construction of tree transducers that effectively compute fine tree transformations (Rahonis, 2001). Moreover, it was used by Maletti and Tîrnăucă (2010) to compare fine and quasi-alphabetic tree bimorphisms (cf. also Theorem 6.6.2).

Definition 6.5.2. Let Σ and Ω be ranked alphabets, and let $n \in \mathbb{N}$ be the minimal integer such that $\Sigma = \bigcup_{i=0}^{n} \Sigma_i$ and $\Omega = \bigcup_{i=0}^{n} \Omega_i$. We define the ranked alphabet $\Sigma^{[n]}$ such that $\Sigma_0^{[n]} := \Sigma_0$ and, for every $m \in [n]$,

$$\Sigma_m^{[n]} := \{u \in \Sigma(\Xi_m) \mid u \text{ is linear in } \Xi_m \text{ and } |u|_{\xi_m} = 1\} \cup \{m\} \ .$$

The alphabet $\Omega^{[n]}$ is defined the same way. The *supremum* of Σ and Ω, denoted by $\Sigma \vee \Omega$, is the ranked alphabet such that for every $m \geq 0$,

$$(\Sigma \vee \Omega)_m := \bigcup_{\max(k,l)=m} \Sigma_k^{[n]} \times \Omega_l^{[n]} \ .$$

Moreover, the two canonical fine tree homomorphisms $\varphi^{\Sigma} \colon T_{\Sigma\vee\Omega}(X \times Y) \to T_{\Sigma}(X)$ and $\varphi^{\Omega} \colon T_{\Sigma\vee\Omega}(X \times Y) \to T_{\Omega}(Y)$ are defined by

$$\varphi^{\Sigma}_{X\times Y}(\langle x, y\rangle) := x \qquad \varphi^{\Sigma}_{m}(\langle t, u\rangle) := \begin{cases} \xi_m & \text{if } t = m \\ t & \text{otherwise} \end{cases}$$

$$\varphi^{\Omega}_{X\times Y}(\langle x, y\rangle) := y \qquad \varphi^{\Omega}_{m}(\langle t, u\rangle) := \begin{cases} \xi_m & \text{if } u = m \\ u & \text{otherwise} \end{cases}$$

for every $\langle x, y\rangle \in X \times Y$ and $\langle t, u\rangle \in (\Sigma \vee \Omega)_m$ with $m \geq 0$. □

We illustrate the above definition by Example 1 of Rahonis (2001).

Example 6.5.3. If $\Sigma = \{f/2, g/1, e/0\}$ and $\Omega = \{f'/2, e'/0\}$, then

$$\begin{aligned} \Sigma_0^{[2]} &= \{e\}, \\ \Sigma_1^{[2]} &= \{1, g(\xi_1)\}, \\ \Sigma_2^{[2]} &= \{2, g(\xi_2), f(\xi_1, \xi_2), f(\xi_2, \xi_1)\}, \text{ and} \\ \Sigma_3^{[2]} &= \{3, g(\xi_3), f(\xi_1, \xi_3), f(\xi_2, \xi_3), f(\xi_3, \xi_1), f(\xi_3, \xi_2)\} \ . \end{aligned}$$

Moreover, $(\Sigma \vee \Omega)_1 = \{(e, 1), (1, e'), (g(\xi_1), e'), (1, 1), (g(\xi_1), 1)\}$. □

Next, we note the main properties of fine tree transformations.

Theorem 6.5.4. *The following hold.*

(i) **Closure properties**. *The class $\tau[fTB]$ is closed under f-concatenation, union, composition and inverses.*

(ii) **Preservation of recognizability.** *The class $\tau[fTB]$ preserves the recognizable and context-free tree languages.*

(iii) *The tree transformations* sub, br, *and intersection and union with a recognizable tree language are all fine tree transformations.*

(iv) *The identity tree transformations belong to the class $\tau[fTB]$.*

(v) *For any* Σ, X, $R \in \mathrm{Rec}_\Sigma(X)$ *and* $x \in X$, *the relations* $\tau, \tau' \subseteq T_\Sigma(X) \times T_\Sigma(X)$ *defined by the respective conditions* $t\tau := t \bullet_x R$ *and* $t\tau := t /_x R$ ($t \in T_\Sigma(X)$), *are fine tree transformations.*

(vi) **Canonical representation.** *A tree transformation* $\tau \subseteq T_\Sigma(X) \times T_\Omega(Y)$ *is fine if and only if there is an* $R \in \mathrm{Rec}_{\Sigma\vee\Omega}(X \times Y)$ *such that* $\tau = \{(t\varphi^\Sigma, t\psi^\Omega) \mid t \in R\}$, *where* $\Sigma \vee \Omega$, φ^Σ *and* φ^Ω *are as in Definition 6.5.2.*

(vii) **Translation power**. *For all* $L_1, L_2 \in CFL$, *there exists a fine tree bimorphism* TB *such that* $\lambda(TB) = L_1 \times L_2$.

(viii) **Decidability results.** *The equality is decidable for fine tree transformations.*

(ix) **Tree transducers.** *There exists a type of tree transducers (cf. Rahonis, 2001, Definition 5) that effectively computes exactly the class* $\tau[fTB]$ *of all fine tree transformations.*

Because of all these nice properties, Bozapalidis and Rahonis (1994) used fine tree transformations defined by tree bimorphisms in an attempt to lift the theory of full AFL's (cf. Ginsburg, 1975) to trees and tree languages.

Next, we focus on the relations between fine tree bimorphisms and tree transducers. First we show that the class of fine tree transformations is essentially different from the classes of tree transformations computed by TOP-transducers. For the specific class τ[TOP] this was already remarked by Bozapalidis (1992) and Rahonis (2001), and here we only refine their argument to suit our needs.

Proposition 6.5.5. $\tau[fTB] \parallel \tau[\text{l-TOP}^\text{R}]$ *and* $\tau[\text{ln-TOP}] \parallel \tau[fTB]$.

Proof. It is known that $\tau[\text{l-TOP}^\text{R}] \subseteq \tau[\text{BOT}]$ (Engelfriet, 1975a, 1977). As argued by Bozapalidis (1992, p. 188), the class $\tau[fTB]$ is incomparable to τ[BOT]. So, $\tau[fTB] \parallel \tau[\text{l-TOP}^\text{R}]$. Moreover, it is known that ln-HOM $\subseteq \tau$[ln-TOP] (Engelfriet, 1975c, Fülöp, 2004). Suppose that ln-HOM $\subseteq \tau[fTB]$. Then also every linear non-deleting inverse tree homomorphism is a fine tree transformation because fine tree transformations are closed under inverses. However, Arnold and Dauchet (1982, Section 3.4) proved on p. 52 in their main theorem that any class of tree bimorphisms containing $\mathbf{B}$[lnsH, Rec, lnsH] (and their inverses) is not closed under composition (cf. Maletti, 2010a, Arnold and Dauchet, 1982, Figure 1), and therefore, the class of fine tree transformations would not be closed under composition. This contradicts Theorem 6.5.4(i), and thus ln-HOM $\parallel \tau[fTB]$. Consequently, τ[ln-TOP] $\parallel \tau[fTB]$. □

The connection between fine tree bimorphisms and tree transducers was investigated by Rahonis (2001), who defined a new type of tree transducer that computes exactly the class of fine tree transformations. In the proofs, he used Definition 6.5.2 and the canonical representation of fine tree bimorphisms of Theorem 6.5.4(vi), the idea being that, by its seven types of rules, the tree transducer naturally simulates the canonical fine tree homomorphisms $(\varphi^\Sigma)^{-1}$ and φ^Ω and the recognizable tree language $R \subseteq T_{\Sigma\vee\Omega}(X \times Y)$. Note that such a tree transducer is incomparable to the classical BOT- and TOP-transducers. On the other hand, its domain and range are recognizable tree languages.

6.6 Comparisons and relations between tree bimorphisms

Next, we relate the class of quasi-alphabetic tree transformations to other known classes of tree relations presented in this monograph (cf. Maletti and Tîrnăucă, 2010). We focus on classes of tree transformations defined by tree bimorphisms that were exhibited in Sections 6.3, 6.2 and 6.5, and classes of tree transformations computed by various TOP-transducers presented in Section 5.2. Recall that $\tau[qTB] = \tau[\mathrm{qH}, \mathrm{Rec}, \mathrm{qH}]$, $\tau[fTB] = \tau[\mathrm{fH}, \mathrm{Rec}, \mathrm{fH}]$, $\tau[aTB] = \tau[\mathrm{aH}, \mathrm{Rec}, \mathrm{aH}]$ and $\tau[pTB] = \tau[\mathrm{pH}, \mathrm{Rec}, \mathrm{pH}]$ are the classes of quasi-alphabetic, fine, alphabetic and permuting tree transformations, respectively.

We start by showing that every tree transformations computed by a finite-state relabeling TOP-transducer is a permuting tree transformation.

Proposition 6.6.1. $\mathrm{QREL} \subseteq \tau[pTB]$.

Proof. Let $\tau \in \mathrm{QREL}$. Since $\mathrm{QREL} \subseteq \tau[\text{ln-TOP}] = \mathrm{REL} \circ \mathrm{FTA} \circ \text{ln-HOM}$ by Figure 5.2.2 and Theorem 5.2.16(iii), there exists a relabeling $\eta \subseteq T_\Sigma(X) \times T_\Gamma(Z)$, a recognizable tree language $R \subseteq T_\Gamma(Z)$, and a linear non-deleting tree homomorphism $\psi\colon T_\Gamma(Z) \to T_\Omega(Y)$ such that $\tau = \{(t\eta^{-1}, t\psi) \mid t \in R\}$. Moreover, by the constructions of Engelfriet (1975a, Theorem 3.5) (cf. also Theorem 5.2.16), ψ is permuting and $\eta^{-1}\colon T_\Gamma(Z) \to T_\Sigma(X)$, i.e., η^{-1} is computed by a deterministic relabeling TOP-transducer. Consequently, η^{-1} and ψ are permuting because every deterministic relabeling is permuting. Thus, the tree bimorphism (η^{-1}, R, ψ) defining τ is permuting. □

Next we consider the relation of quasi-alphabetic and fine tree transformations by showing that every quasi-alphabetic tree transformation is also fine. The strictness of this inclusion can be obtained using Proposition 6.5.5. Note that for this result we need the supremum ranked alphabet of Definition 6.5.2.

Theorem 6.6.2. $\tau[qTB] \subseteq \tau[fTB]$.

Proof. Let $TB = (\varphi, R, \psi)$ be a quasi-alphabetic tree bimorphism, where $R \subseteq T_\Gamma(Z)$, $\varphi\colon T_\Gamma(Z) \to T_\Sigma(X)$ and $\psi\colon T_\Gamma(Z) \to T_\Omega(Y)$. Moreover, let $x \in X$ and $y \in Y$. We build the linear tree homomorphism $\vartheta\colon T_\Gamma(Z) \to T_{\Sigma\vee\Omega}(X \times Y)$ such that $\vartheta_Z(z) := \langle z\varphi, z\psi\rangle$ for every $z \in Z$ and $\vartheta_k(f) := \langle t(\varepsilon)_\omega, u(\varepsilon)_{\omega'}\rangle(\xi_1, \ldots, \xi_k, t_1, \ldots, t_l)$ for every $f \in \Gamma_k$ $(k \geq 0)$, where

- $t = \varphi_k(f)$ and $u = \psi_k(f)$,
- $\{i_1, \ldots, i_m\} = \mathrm{dom}_X(t)$ and $\{j_1, \ldots, j_n\} = \mathrm{dom}_Y(u)$,
- $l = \max(m, n)$ and

$$t_a = \begin{cases} \langle t(i_a), u(j_a)\rangle & \text{if } a \leq \min(m, n) \\ \langle t(i_a), y\rangle & \text{if } n < a \leq m \\ \langle x, u(j_a)\rangle & \text{if } m < a \leq n \end{cases}$$

  for every $a \in [l]$, and

- $\omega = \omega_1 \ldots \omega_{k+m}$ and $\omega' = \omega'_1 \ldots \omega'_n$ are such that $t(\omega_a) = \vartheta_k(f)(a)\pi^1$ for every $a \in [k+m]$ and $t(\omega'_b) = \vartheta_k(f)(b)\pi^2$ for every $b \in [k+n]$, where π^1 and π^2 are the usual projections to the first and second components, respectively, with $x\pi^1 = x = x\pi^2$ for every $x \in X$.

By Theorem 4.4.17(iii), $R\vartheta$ is recognizable. An easy proof shows that

$$\tau(TB) = \{(t\varphi^{\Sigma}, t\varphi^{\Omega}) \mid t \in R\vartheta\} \ ,$$

where φ^{Σ} and φ^{Ω} are the canonical fine tree homomorphisms of Definition 6.5.2. Hence, $\tau(TB)$ is a fine tree transformation by Theorem 6.5.4(vi), which is the analogue of our Theorem 6.2.9 for quasi-alphabetic tree transformations. □

As an immediate consequence of Theorems 6.6.2 and 6.5.4(ii), we get the following result (cf. also Theorem 6.2.6).

Corollary 6.6.3. *Quasi-alphabetic tree transformations preserve the recognizable and the context-free tree languages.*

Let us collect the results of this section in a HASSE diagram (see Figure 6.6.1).

Theorem 6.6.4. *The inclusion relations between the classes of tree transformations and translations defined by finite-state relabeling TOP-transducers, permuting tree bimorphisms, alphabetic tree bimorphisms, quasi-alphabetic tree bimorphisms, fine tree bimorphisms, linear TOP-transducers, linear non-deleting TOP-transducers, linear* $\mathrm{TOP}^{\mathrm{R}}$*-transducers and linear TOP-transducers with finite look ahead, are given by the* HASSE*Hasse diagram!classes of tree translations defined by tree bimorphisms and tree transducers diagrams of Figure 6.6.1.*

Proof. The following six statements are sufficient to prove the claims of the left diagram.

$$\tau[aTB] = \mathrm{QREL} \subset \tau[pTB] \subseteq \tau[qTB] \subseteq \tau[fTB] \tag{6.6.1}$$

$$\tau[pTB] \subseteq \tau[\text{ln-TOP}] \subset \tau[\text{l-TOP}] \subseteq \tau[\text{l-TOP}^{\mathrm{F}}] \subseteq \tau[\text{l-TOP}^{\mathrm{R}}] \tag{6.6.2}$$

$$\tau[qTB] \subseteq \tau[\text{l-TOP}^{\mathrm{F}}] \tag{6.6.3}$$

$$\tau[qTB] \parallel \tau[\text{l-TOP}] \tag{6.6.4}$$

$$\tau[\text{ln-TOP}] \parallel \tau[fTB] \tag{6.6.5}$$

$$\tau[fTB] \parallel \tau[\text{l-TOP}^{\mathrm{R}}] \tag{6.6.6}$$

Statement (6.6.1) is clear by Theorem 6.3.9 and Proposition 6.6.1. The strictness is due to the fact that QREL is closed under intersection whereas this is not true for $\tau[pTB]$ by Theorem 6.2.12. The final inclusions of (6.6.1) are proved in Theorem 6.6.2 and Proposition 6.3.2. The inclusions of (6.6.2) are all obvious (cf. also Figure 5.2.2), and (6.6.3) is shown in Proposition 6.2.31. Finally, the inequality (6.6.4) is proved in Proposition 6.2.32, and inequalities (6.6.5) and (6.6.6) are proved in Proposition 6.5.5.

It is proved in Theorems 6.3.7 and 3.3.22, and Corollary 6.3.8 that

$$\lambda[FST] \subset \lambda[aTB] = \lambda[\mathrm{QREL}] = \lambda[sSDTS] \subset \lambda[pTB] = \mathcal{SDT} \ .$$

Moreover, Theorem 6.2.25 shows that $\mathcal{SDT} = \lambda[qTB]$. To prove that the remaining classes also collapse to the class of all syntax-directed translations, we prove that for every $\tau \in \tau[\mathrm{lH}, \mathrm{Rec}, \mathrm{lH}]$, we can construct a quasi-alphabetic tree bimorphism TB such that $\lambda(TB) = \lambda(\tau)$. It is clear from the definitions that fine tree bimorphisms are linear, and Maletti (2008, Theorem 4) proved $\tau[\text{l-TOP}^{\mathrm{R}}] \subseteq \tau[\mathrm{lH}, \mathrm{Rec}, \mathrm{lH}]$.

To this end, we first prove that $\lambda(\tau) \in \lambda[\mathrm{lnH}, \mathrm{Rec}, \mathrm{lnH}]$ using a construction that is similar to the one in the proof of Theorem 6.2.30 and Lemma 6.2.3 (eliminating variables

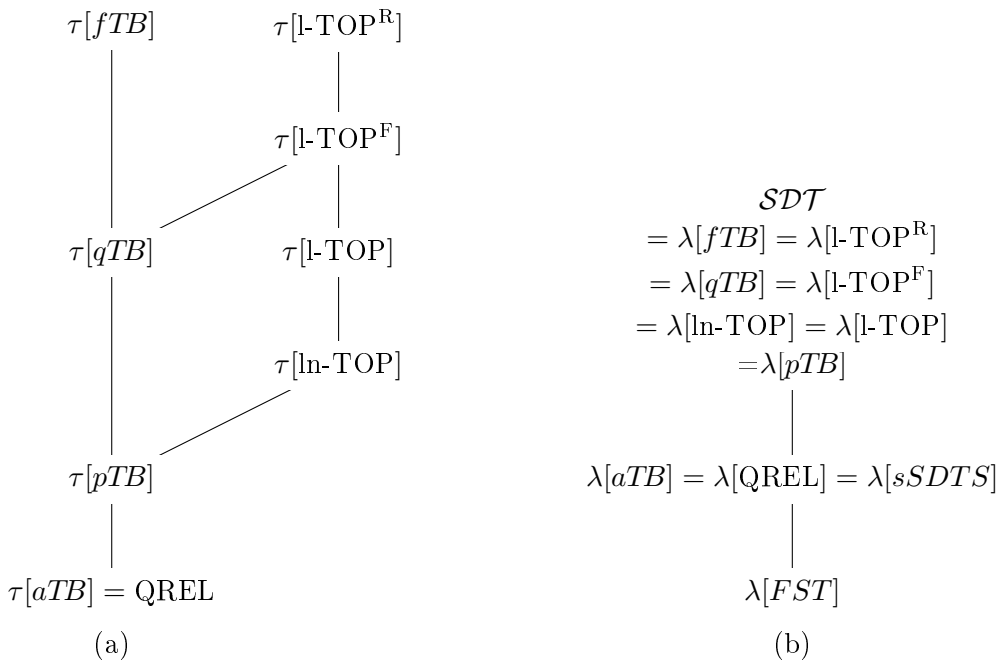


Figure 6.6.1: HASSE diagrams of classes of tree transformations defined by tree bimorphisms and tree transducers (a), and corresponding translations (b).

in the center tree language and turning the tree homomorphisms into non-deleting ones such that no variables are output for the subtrees that were deleted by the original tree homomorphisms). Next we flatten the output trees. Let $TB' = (\varphi, R, \psi)$ be a linear non-deleting tree bimorphism such that $R \subseteq T_\Gamma$, $\varphi\colon T_\Gamma \to T_\Sigma(X)$ and $\psi\colon T_\Gamma \to T_\Omega(Y)$. Then, the quasi-alphabetic tree homomorphisms φ' and ψ' are constructed by setting $\varphi'_m(f) := g(t_1, \ldots, t_n)$ and $\psi'_m(f) := g'(u_1, \ldots, u_{n'})$, where for every $m \geq 0$, $f \in \Gamma_m$, $g \in \Sigma_n$ and $g' \in \Omega_{n'}$ are new output symbols. In addition, $t_1, \ldots, t_n \in X \cup \Xi$ and $u_1, \ldots, u_{n'} \in Y \cup \Xi$ are such that

$$\mathrm{yd}_{X\cup\Xi}(\varphi_m(f)) = t_1 \ldots t_n \quad \text{and} \quad \mathrm{yd}_{Y\cup\Xi}(\psi_m(f)) = u_1 \ldots u_{n'} \ .$$

Now let $TB'' := (\varphi', R, \psi')$. It should be clear that $\lambda(TB'') = \lambda(TB')$, which proves the statement because $\tau(TB'') \in \tau[qTB]$. □

6.7 Other types of tree bimorphisms

Dauchet and Tison (1992) and Raoult (1992), and in light of applications of tree bimorphisms in NLP Maletti (2010a) and Tîrnăucă (2008), were the only ones to briefly survey and link different classes of tree bimorphisms. In this section, we propose a more complete and up-to-date overview on tree transformations classes defined by tree bimorphisms, linking them with well-known types of synchronous grammars and tree transducers. Thus, in Section 6.7.1, we discuss the *rational relations of binary trees* of Takahashi (1977) and link them with ITGs, *linear complete strict bimorphisms* of Arnold and Dauchet (1976a, 1982), *linear bimorphisms* of Comon et al. (2007), *extended top-down tree-to-string transducers*

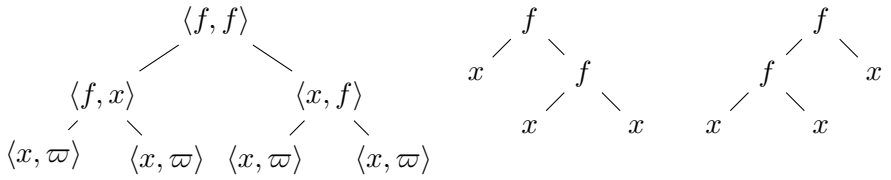


Figure 6.7.1: Example of the canonical representation of rational relations of binary trees used by Takahashi (1977). By applying the projections on the first and second component, respectively, on the balanced tree on the left, we get the two trees on the right.

of Huang et al. (2006) and *synchronized transductions* of Rahonis (2001). Next, in Section 6.7.2, we bring into actuality the tree counterparts of regular translations (Σ-*rational tree transformations*) and of simple SDTSs (Σ-*algebraic tree transformations*) studied by Steinby (1984, 1986, 1990). Finally, in Section 6.7.3, we present *bimorphisms of magmoids* (Arnold and Dauchet, 1978b, 1979, 1982, Estenfeld, 1982, Maletti, 2013) and *bitransformations* (Nederhof and Vogler, 2012, Shieber, 2006), which extend the original tree bimorphisms defined in Section 6.1 by working with tuples of trees (instead of trees) or with homomorphisms computed by machines beyond the power of one-state BOT- or TOP-transducers (see Section 5.2.3).

6.7.1 Miscellaneous

One of the first attempts to extend the constructions and results of Nivat (1968) from strings to trees was made by Takahashi (1977), who introduced *rational relations of binary trees*. Such a relation τ is the tree transformation defined by a linear non-deleting strict tree bimorphism (φ, R, ψ) (Arnold and Dauchet, 1976a) in which $\varphi, \psi \in \mathrm{lnsH}$ and the center is a set $R \subseteq T_\Gamma(Z)$ of balanced binary trees in the sense of Knuth (1968), i.e., $\Gamma_0 \neq \emptyset$ and $\Gamma_2 \neq \emptyset$, and $\Gamma_m = \emptyset$, otherwise. For example, if $\Gamma = \{f/2, e/0\}$ and $Z = \{z\}$, then $f(f(x,e), f(f(x,x), f(x,x)))$ is such a tree but $f(f(f(x,x), f(x,x)), e)$ is not.

The key to such a definition of a rational relation of binary trees was a canonical representation further generalized by Dauchet and Tison (1990) (cf. also Raoult, 1992, p.320): two binary trees are the images under two projections (on the first and second component, respectively) of a common unique balanced binary tree over the Cartesian product of the alphabets augmented by a special symbol ϖ, called *dummy node* by Takahashi (1977) and meaning 'not defined'. For example, Figure 6.7.1 above shows how the trick of Takahashi (1977) works: the two binary trees on the left are the projections on the first and second component, respectively, of the unique balanced binary tree on the right.

Now we list without proof the following properties of rational relations of binary trees:

(i) The class of rational relations of binary trees is closed under inverses and composition.

(ii) Rational relations of binary trees preserve the recognizability of tree languages. In particular, the domain and range of a rational relation of binary trees are recognizable tree languages.

(iii) The class of rational relations of binary trees is equal to the class of tree transformations defined by linear tree bimorphisms with the center a set of balanced binary trees as above.

When defined as in Theorem 3.3.12(ii), it becomes obvious that by similar constructions and proofs as in Section 6.2.2 (cf. also Tîrnăucă, 2011, Section 6), the class of parse tree transformations of ITGs is equal to the class of rational relations of binary trees, and hence, additional theoretical properties (items (i)-(iii) above) are available for such synchronous grammars. Moreover, a new characterization of translations of ITGs is thus given through a pioneer work on tree bimorphisms by Takahashi (1977). For the usefulness of such devices in linguistics, the reader is referred to Wu (1997), Wu and Fung (2005), Wu et al. (2006), Saers and Wu (2009), Saers et al. (2009) and Saers et al. (2012), for example.

Another attempt to extend the results and constructions of Nivat (1968) to the tree case was made by Arnold and Dauchet (1976a, 1982), who introduced the *linear non-deleting strict tree bimorphisms* $\mathbf{B}[\text{lnsH}, \text{Rec}, \text{lnsH}]$ and *transducteurs généralisé ascendants*, fashionable nowadays in machine translation under the name of extended top-down tree transducers (see Section 5.2.4).

For the class $\tau[lnsTB]$ of tree transformations defined by linear non-deleting strict tree bimorphisms $\mathbf{B}[\text{lnsH}, \text{Rec}, \text{lnsH}]$, the following hierarchy seems to be available (cf. Arnold and Dauchet, 1976a, 1982, Theorem 6.2):

$$\tau[lnsTB] \subset \tau[lnsTB]^2 = \tau[lnsTB]^3 \ .$$

However, there is no clear proof of $\tau[lnsTB]^2 = \tau[lnsTB]^3$ to be found in the literature, the authors reasoning that it is too long and technical to be given (see Comon et al., 2007, Raoult, 1992, for example). On the other hand, Arnold and Dauchet (1982, Section 3.4) proved that no class of tree bimorphisms containing $\mathbf{B}[\text{lnsH}, \text{Rec}, \text{lnsH}]$ is closed under composition, a phenomenon originally called "*chevauchement de découpage*" and further explained by Maletti (2010a).

Furthermore, Arnold and Dauchet (1976a) linked linear non-deleting strict tree bimorphisms to linear non-deleting XTT-transducers in which the right-hand side of each rule cannot be a tree with only one node - a variable. Their constructions and results were further used by Maletti (2007, 2008, 2010a) to prove that linear non-deleting XTT-transducers are not closed under composition and show the connection between linear non-deleting tree bimorphisms, STSGs and linear non-deleting XTT-transducers (see Sections 6.4 and 5.2.4; cf. also Shieber, 2004).

Using the idea behind XTT-transducers to naturally model really complicated linguistic phenomena, Huang et al. (2006) extended the expressivity of generalized syntax-directed translators of Aho and Ullman (1971, 1969c), Martin and Vere (1970), Baker (1978a) and Yamada and Knight (2001) by introducing *extended top-down tree-to-string transducers*. Actually, without mentioning it, they show that the tree transformations computed by such transducers can be defined by linear non-deleting tree bimorphisms. Consequently, this class of tree transformations is easily further extended to a weighted model, which is shown to perform better than the classical generalized syntax-directed translators (see Gécseg and Steinby, 1984, Gécseg and Steinby, 1997, for an overview) in a fragment of an English-to-Chinese translation experiment.

Comon et al. (2007, Section 6.5) mentions several results concerning tree bimorphisms. For example, we find in their Theorem 6.5.2 that for the class $\tau[lTB]$ of tree transformations defined by the linear tree bimorphisms $\mathbf{B}[\text{lH}, \text{Rec}, \text{lH}]$, the following hierarchy hold:

$$\tau[lTB] \subset \tau[lTB]^2 \subset \tau[lTB]^3 \subset \tau[lTB]^4 = \tau[lTB]^5 \ .$$

Moreover, Comon et al. (2007, Theorem 6.5.1) offers another characterization of BOT-transformations by showing that $\mathrm{BOT} = \tau[\mathrm{lnssH}, \mathrm{Rec}, \mathrm{H}] = \tau[\mathrm{pH}, \mathrm{Rec}, \mathrm{H}]$.

Rahonis (2001) introduced *synchronized transductions* as the class of tree transformations computed by synchronized tree transducers, machines that implement the idea of parallel computation (Salomaa, 1994). However, an alternative characterization of such tree transformations by means of tree bimorphisms is given byRahonis (2001, Theorem 22): a tree transformation $\tau \subseteq T_\Sigma(X) \times T_\Omega(Y)$ is synchronized if it is defined by a tree bimorphism $(\varphi^\Sigma, R, \varphi^\Omega)$, where

- φ^Σ and φ^Ω are the canonical fine tree homomorphisms of Definition 6.5.2, and
- $R \subseteq T_{\Sigma \vee \Omega}(X \times Y)$ is a nondeterministic synchronized tree language, i.e., the tree language recognized by a nondeterministic synchronized tree recognizer of Salomaa (1994) (note that this class of tree languages strictly contains Rec).

Using the connection with tree bimorphisms, Rahonis (2001) proves several properties of synchronized transductions:

(i) The class of synchronized transductions is not closed under composition but it is closed under composition with fine tree transformations.

(ii) Every fine tree transformations is a synchronized transduction.

(iii) The class of synchronized transductions is incomparable with BOT and TOP.

(iv) The equivalence is undecidable for synchronized transductions.

Further research on synchronized transductions is proposed by Rahonis (2001, p. 398).

6.7.2 Σ-rational and Σ-algebraic tree transformations

Even a more directly algebraic approach to tree transformations was suggested by Steinby (1986, 1984), by extending the notion of a rational subset from monoids to arbitrary algebras as in Steinby (1981). Thus, Σ-*rational* and Σ-*algebraic tree transformations* were defined by special tree bimorphisms that resemble in many ways (cf. Theorems 6.7.2 and 6.7.4) their string counterparts - regular translations and simple syntax-directed translations both studied in Chapter 3. Moreover, they also have some properties of their own since these tree transformations are locally finite and closed under intersection and many basic questions (e.g., equivalence), undecidable for regular translations, are decidable for them.

An immediate interest in Σ-rational and Σ-algebraic tree transformations may arise from their relation with regular translations and simple syntax-directed translations. Furthermore, Steinby (1990) put forward Σ-rational and Σ-algebraic tree transformations as error-correcting transformations required by certain errors in tree representations of patterns, and thus, perhaps certain modifications of trees can be modeled by Σ-rational tree transformations or related systems.

Before starting the exposition of the main results of Steinby (1984, 1986, 1990), we need to present some auxiliary notation and basic general algebra terminology. However, all algebraic notions and results used here can be found, for example, in Burris and Sankappanavar (1981), Gécseg and Steinby (1984) and Steinby (2004, 1981, 1977, 2005), which shall be checked for further details and references.

First of all, in a Σ-*algebra* $\mathcal{K} = (K, \Sigma)$, K is a non-empty set of elements, each $f \in \Sigma_m$ $(m \geq 1)$ is realized as an m-ary operation $f^{\mathcal{K}}\colon K^m \to K$, and any $e \in \Sigma_0$ is realized as a constant $e^{\mathcal{K}}$. The ΣX-trees form the *term algebra* $\mathcal{T}_\Sigma(X) = (T_\Sigma(X), \Sigma)$ defined so that $e^{\mathcal{T}_\Sigma(X)} = e$ for all $e \in \Sigma_0$, and $f^{\mathcal{T}_\Sigma(X)}(t_1, \ldots, t_m) = f(t_1, \ldots, t_m)$ for all $m \geq 1$, $f \in \Sigma_m$ and $t_1, \ldots, t_m \in T_\Sigma(X)$.

The set $\mathrm{Rat}_\Sigma(X)$ of Σ-*rational* ΣX-*tree languages* is defined inductively by the following conditions:

(1) every finite $T \subseteq T_\Sigma(X)$ is in $\mathrm{Rat}_\Sigma(X)$;

(2) if $R, R' \in \mathrm{Rat}_\Sigma(X)$, the $R \cup R' \in \mathrm{Rat}_\Sigma(X)$;

(3) if $m \geq 1$, $f \in \Sigma_m$ and $R_1, \ldots, R_m \in \mathrm{Rat}_\Sigma(X)$, then $f(R_1, \ldots, R_m) \in \mathrm{Rat}_\Sigma(X)$;

(4) if $R \in \mathrm{Rat}_\Sigma(X)$, then $\langle R \rangle \in \mathrm{Rat}_\Sigma(X)$, where $\langle R \rangle$ is the subalgebra generated by R in the term algebra $\mathcal{T}_\Sigma(X)$.

Unless $\Sigma = \Sigma_0$, $\mathrm{Rat}_\Sigma(X) \subset \mathrm{Rec}_\Sigma(X)$.

Next, recall that a homomorphism $\varphi\colon \mathcal{T}_\Sigma(X) \to \mathcal{T}_\Sigma(Y)$ of term algebras is a special linear non-deleting strict tree homomorphism which commutes in some sense with the operations of the two term algebras involved. Since $\mathcal{T}_\Sigma(X)$ is generated by X, φ is completely determined by the images $x\varphi \in T_\Sigma(Y)$ of the leaf symbols $x \in X$. In fact, the image $t\varphi$ of any $t \in T_\Sigma(X)$ is obtained from t by replacing every occurrence of each $x \in X$ with the ΣY-tree $x\varphi$.

A subset H of K is called an *algebraic subset* of the Σ-algebra $\mathcal{K} = (K, \Sigma)$ if there exist an alphabet X, a homomorphism of Σ-algebras $\varphi\colon \mathcal{T}_\Sigma(X) \to \mathcal{K}$ and a ΣX-tree language $R \in \mathrm{Rec}_\Sigma(X)$ such that $H = R\varphi$ (cf. Mezei and Wright, 1967, these subsets can be obtained as solutions of fixed-point equations). The set of all algebraic subsets of $\mathcal{K}$ is denoted by $\mathrm{Alg}\mathcal{K}$, and it is known that $\mathrm{Rat}\mathcal{K} \subseteq \mathrm{Alg}\mathcal{K}$. Moreover, for any $\mathcal{T}_\Sigma(X)$, $\mathrm{Alg}\mathcal{T}_\Sigma(X) = \mathrm{Rec}_\Sigma(X)$. Finally, note that in a monoid X^*, the algebraic subsets are the CFLs over X, and $\mathrm{Alg}(X^* \times Y^*)$ is the set of all simple syntax-directed translations from X^* to Y^* (Berstel, 1979).

Now, we can proceed to the formal definition of Σ-rational tree transformations.

Definition 6.7.1. A Σ-*rational tree bimorphism* $TB = (\varphi, R, \psi)$ consists of two homomorphisms of term algebras $\varphi\colon \mathcal{T}_\Sigma(Z) \to \mathcal{T}_\Sigma(X)$ and $\psi\colon \mathcal{T}_\Sigma(Z) \to \mathcal{T}_\Sigma(Y)$, and a Σ-rational tree language $R \in \mathrm{Rat}_\Sigma(Z)$. A tree transformation is called a Σ-*rational tree transformation* (Σ-RTT) if it is defined by a Σ-rational bimorphism. Furthermore, let $\mathrm{Rat}_\Sigma(X, Y)$ denote the set of all Σ-RTTs from $T_\Sigma(X)$ to $T_\Sigma(Y)$. □

Next, we list without proof some of the nice properties of Σ-rational tree transformations (cf. Steinby, 1984, 1986, 1990). Note that the tree bimorphism was the key approach to elegantly prove all of these useful properties.

Theorem 6.7.2. *The following hold.*

(i) **Alternative characterization.** *The Σ-RTTs from $T_\Sigma(X) \times T_\Sigma(Y)$ are exactly the rational subsets of the direct product $\mathcal{T}_\Sigma(X) \times \mathcal{T}_\Sigma(Y)$.*

(ii) **Closure properties.** *The class of all Σ-RTTs is closed under union, intersection, composition and inverses.*

(iii) *Any Σ-RTT τ is symmetrically locally finite.*

(iv) *If $\tau \in \mathrm{Rat}_\Sigma(X,Y)$ and $T \in \mathrm{Rat}_\Sigma(X)$, then $T\tau \in \mathrm{Rat}_\Sigma(Y)$.*

(v) *If $\tau \in \mathrm{Rat}_\Sigma(X,Y)$ and $T \in \mathrm{Rat}_\Sigma(Y)$, then $T\tau^{-1} \in \mathrm{Rat}_\Sigma(X)$.*

(vi) *If $\tau \in \mathrm{Rat}_\Sigma(X,Y)$ and $T \in \mathrm{Rec}_\Sigma(X)$, then $T\tau \in \mathrm{Rec}_\Sigma(Y)$.*

(vii) *If $\tau \in \mathrm{Rat}_\Sigma(X,Y)$ and $T \in \mathrm{Rec}_\Sigma(Y)$, then $T\tau^{-1} \in \mathrm{Rec}_\Sigma(X)$.*

(viii) **Translation power.** *The translation $\lambda(\tau)$ defined by any $\tau \in \mathrm{Rat}_\Sigma(X,Y)$ is a regular translation from X^* to Y^*. If $\Sigma_0 \neq \emptyset$ and $\Sigma_m \neq \emptyset$ for some $m \geq 2$, any regular translation $\lambda \subseteq X^* \times Y^*$ is obtained this way.*

(ix) **Tree Transducers.** *The class of Σ-RTTs is computed by BOT-transducers, but not always by TOP-transducers.*

(x) **Decidability results.** *The emptiness, the finiteness, the membership, the inclusion and the equality are all decidable for Σ-RTTs.*

As argued by Steinby (1986), the representation of Σ-RTTs by tree bimorphisms permits a direct comparison with other classes of tree transformations defined by tree bimorphisms. Since rational tree languages are recognizable and homomorphisms of term algebra are special linear non-deleting strict tree homomorphisms, Σ-RTT form a proper subclass of the class $BI = \tau[\mathrm{lnsH}, \mathrm{Rec}, \mathrm{lnsH}]$ of linear non-deleting strict tree transformations of Arnold and Dauchet (1976a, 1982) (see Section 6.7.1). In addition, the permuting tree transformations of Takahashi (1972) (see Section 6.3) form a class incomparable with Σ-RTTs because they can relabel nodes and reorder subtrees, but have a weaker subtree rewriting capability than Σ-RTTs.

Corresponding to simple syntax-directed translations (also known as algebraic transductions), Σ-algebraic tree transformations were defined by means of tree bimorphisms thus (cf. Steinby, 1986, 1984).

Definition 6.7.3. A *Σ-algebraic tree bimorphism* is a triple $TB = (\varphi, R, \psi)$, where $R \in \mathrm{Rec}_\Sigma(Z)$, and $\varphi\colon \mathcal{T}_\Sigma(Z) \to \mathcal{T}_\Sigma(X)$ and $\psi\colon \mathcal{T}_\Sigma(Z) \to \mathcal{T}_\Sigma(Y)$ are homomorphisms of term algebras. A tree transformation is called a *Σ-algebraic tree transformation* (Σ-ATT) if it is defined by a Σ-algebraic tree bimorphism. Furthermore, let $\mathrm{Alg}_\Sigma(X,Y)$ denote the set of all Σ-ATTs from $T_\Sigma(X)$ to $T_\Sigma(Y)$. □

Now, we list without proof the properties of Σ-algebraic tree transformations.

Theorem 6.7.4. *The following hold.*

(i) **Alternative characterization.** *The Σ-ATTs from $T_\Sigma(X) \times T_\Sigma(Y)$ are exactly the algebraic subsets of the direct product $\mathcal{T}_\Sigma(X) \times \mathcal{T}_\Sigma(Y)$.*

(ii) **Closure properties.** *The class of all Σ-ATTs is closed under union, intersection, composition and inverses.*

(iii) *Any $\tau \in \mathrm{Alg}_\Sigma(X,Y)$ is symmetrically locally finite.*

(iv) *If $\tau \in \mathrm{Alg}_\Sigma(X,Y)$ and $T \in \mathrm{Rec}_\Sigma(X)$, then $T\tau \in \mathrm{Rec}_\Sigma(Y)$.*

(v) *If* $\tau \in \mathrm{Alg}_\Sigma(X,Y)$ *and* $T \in \mathrm{Rec}_\Sigma(Y)$, *then* $T\tau^{-1} \in \mathrm{Rec}_\Sigma(Y)$.

(vi) **Translation power.** *The translation* $\lambda(\tau)$ *defined by any* $\tau \in \mathrm{Alg}_\Sigma(X,Y)$ *is a simple syntax-directed translation from* X^* *to* Y^*. *If* $\Sigma_0 \neq \emptyset$ *and* $\Sigma_m \neq \emptyset$ *for some* $m \geq 2$, *then for every simple syntax-directed translation* $\lambda \subseteq X^* \times Y^*$ *there exists a* $\tau \in \mathrm{Alg}_\Sigma(X,Y)$ *such that* $\lambda = \lambda(\tau)$.

(vii) **Tree Transducers.** *The class of* Σ*-algebraic tree transformations is computed by linear BOT-transducers, but not always by TOP-transducers.*

(viii) *Since any* $\tau \in \mathrm{Alg}_\Sigma(X,Y)$ *is locally finite, the questions "Is* $T_\Sigma(X) \times T_\Sigma(Y) \setminus \tau$ *in* $\mathrm{Alg}_\Sigma(X,Y)$*?" and "Is* $\tau = T_\Sigma(X) \times T_\Sigma(Y)$*?" are always answered as* `No`.

(ix) **Decidability results.** *The emptiness, the finiteness, the membership, the inclusion and the equality are all decidable for* Σ*-ATTs.*

(x) *For any two* Σ*-ATTs* τ *and* τ', *the question "Is* $\tau \cap \tau' = \emptyset$*?" is decidable.*

(xi) *For any* Σ*-ATT* τ, *it is decidable whether* τ *is a partial function.*

(xii) *For any* $\tau_1, \tau_2 \in \mathrm{Alg}_\Sigma(X,Y)$ *and any* $R \in \mathrm{Rec}_\Sigma(X)$, *the following questions are decidable:*

a. Does $r\tau_1 \cap r\tau_2 = \emptyset$ *hold for every* $r \in R$*?*

b. Does $r\tau_1 = r\tau_2$ *hold for every* $r \in R$*?*

c. Does $r\tau_1 \subseteq r\tau_2$ *hold for every* $r \in R$*?*

Note that obviously the results (ix)-(xii) of the above theorem hold for Σ-RTTs, too (cf. Steinby, 1984). Again the definition of Σ-ATTs by tree bimorphisms was essential in elegantly tackle the proofs of all the properties mentioned above. Moreover, when defined this way it is easy to show that the inclusion $\mathrm{Rat}_\Sigma(X,Y) \subseteq \mathrm{Alg}_\Sigma(X,Y)$ is proper unless $\Sigma = \Sigma_0$ (cf. Steinby, 1986). It is also obvious that Σ-ATTs form a proper subclass of the class $BI = \tau[\mathrm{lnsH}, \mathrm{Rec}, \mathrm{lnsH}]$ of linear non-deleting strict tree transformations studied by Arnold and Dauchet (1976a, 1982).

6.7.3 Extended bimorphisms: magmoids and bitransformations

An alternative way to define trees and tree languages, originating with Pair and Quéré (1968) and quite popular among French writers, involves to work with tuples of trees as basic objects rather than trees. The usual tree operations are then augmented by operations which catenate tuples of trees or form a tree from an m-tuple by creating a new root labeled by an m-ary operator. This formalism has been developed further by Arnold and Dauchet (1978b, 1979) to a theory of algebraic structures called *magmoids*, which also embodies many of the ideas of Eilenberg and Wright (1967). Arnold (1977) discusses specification methods and decidability results within the framework of magmoids.

Subsequently, Arnold and Dauchet (1982) and Bozapalidis (1988) extended tree homomorphisms and tree bimorphisms to morphisms and bimorphisms of magmoids. Actually, the class of morphisms of magmoids, also called m-morphisms, viewed as tree transformations are equivalent to the class of deterministic multi bottom-up transducers of Lilin (1978) and Fülöp et al. (2005), and an 1-morphism is our classical tree homomorphism

defined in Section 4.2. Arnold and Dauchet (1982) introduced many special classes of bimorphisms of magmoids such as linear, non-deleting and strict, for example, and studied, almost exclusively, the closure under composition of such classes as well as compositions and decompositions between them. Recently, using this theory of magmoids, Maletti (2013) showed that every tree transformation generated by a synchronous forest substitution grammar (cf. also Raoult, 1992, Zhang et al., 2008b, a, Sun et al., 2009, for the relevance of such synchronous grammars in theory and practice) is defined by a bimorphism with a recognizable center tree language and input and output homomorphisms computed by deterministic multi bottom-up tree transducers (i.e., m-morphisms). Thus, he was able to show many decomposition results of synchronous forest substitution grammars and that the tree transformation class generated by these synchronous grammars is not closed under composition.

Moreover, Estenfeld (1982) used a similar algebraic approach and described tree transformations by tree bimorphisms where the tree languages are replaced by *categories* of derivation trees of CFGs and, instead of the input and output tree homomorphisms, we have *functors* of CFGs (cf. also Schnorr, 1969, Walter, 1976). Such special bimorphisms are shown to define the same translations as simple SDTSs and share properties such as closure under composition and inverses.

On the other hand, Nederhof and Vogler (2012) introduced *bitransformations*, which extend the classical notion of tree bimorphism in the following way. The center is still a recognizable tree language, but contrary to the classical case where the input and output homomorphisms can be viewed as tree transformations computed by a one-state TOP- or BOT-transducers, they now are tree transformations defined by a very restricted type of macro tree transducers (Engelfriet, 1980, Engelfriet and Vogler, 1985, Fülöp and Vogler, 1998, Maneth, 2004): its linear, non-deleting, total and deterministic version in which there is at most one state of rank $m \geq 0$. Basically, such a machine combines a TOP-transducer and a CFTG to serve as a formal model of semantics in which context can be implicitly handled by allowing additional parameter symbols in the rules.

In the same paper, they proposed a variant of synchronous CFTGs and proved that such a formalism generates the same class of tree transformations as the one defined by the above-mentioned bitransformations. This alternative characterization of synchronous CFTG in terms if tree bimorphisms permitted a direct comparison between synchronous CFTGs and well-known tree transducers and synchronous tree grammars. For example, it becomes obvious that every synchronous tree adjoining grammar of Shieber and Schabes (1990b, a, 1991) and Abeillé et al. (1990) can be simulated by a synchronous CFTG because synchronous tree adjoining grammars are characterized by bitransformations in which the input and output homomorphisms are computed by linear non-deleting total deterministic macro tree transducers which have at most one state of rank 1 and at most one state of rank 2, no other states being allowed (Shieber, 2006). Among other properties that make synchronous CFTGs appealing for machine translation applications, Nederhof and Vogler (2012) mentioned their polynomial parsing complexity and their natural way to define tree transformations and translations, which allows a straightforward extension to weighted formalisms by assigning probabilities to rules.

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

CHAPTER 7

Synchronous Translation and Tree Transformation Generators

In this chapter, two new classes of translation and tree transformation defining devices that extend in a natural way the formalisms exhibited in Table 8.1.1 are introduced. Further related research topics that might be rewarding are proposed in Section .

7.1 Synchronous translation generators

We shall now introduce a class of translation-defining devices that generalize the synchronous context-free grammars of Satta and Peserico (2005), and hence the basic syntax-directed translation schemes of Irons (1961), Lewis II and Stearns (1968) and Aho and Ullman (1972) - see Chapter 3 for an overview. They also generalize the "syntax-connected transduction schemes" of Schreiber (1975, 1976) but, as we shall show, they seem to be essentially equivalent in power with these[1].

Definition 7.1.1. A *synchronous translation generator* (STG) is a system $SG = (N, X, Y, P, S, S')$ specified as follows.

(1) X and Y are the *input* and the *output alphabet*, respectively.

(2) N is an alphabet of *nonterminal symbols* such that $N \cap (X \cup Y) = \emptyset$.

(3) $S, S' \in N$ are the *start symbols*.

(4) P is a finite set of *productions* $A; B \to \alpha; \beta\,(\sigma)$, where $A, B \in N$, $\alpha \in (N \cup X)^*$, $\beta \in (N \cup Y)^*$ and σ is a mapping that *associates* each occurrence of a nonterminal in β with a unique occurrence of a nonterminal in α in such a way that only occurrences of the same nonterminal in β can be associated with any given occurrence of a nonterminal in α. When $\beta \in Y^*$ in a production $A; B \to \alpha; \beta\,(\sigma)$, we may omit the mapping σ and write the production simply as $A; B \to \alpha; \beta$.

The *input grammar* of SG is the CFG $SG^{\text{in}} = (N, X, P^{\text{in}}, S)$, where P^{in} consists of the productions $A \to \alpha$ such that $A; B \to \alpha; \beta\,(\sigma) \in P$ for some B, β' and σ. Similarly, the grammar $SG^{\text{out}} = (N, Y, P^{\text{out}}, S')$, where $P^{\text{out}} := \{B \to \beta \mid A; B \to \alpha; \beta\,(\sigma) \in P \text{ for some } A,\ \alpha \text{ and } \sigma\}$, is the *output grammar* of SG. □

Hence, any production of an STG $SG = (N, X, Y, P, S, S')$ can be written in the form

$$A; B \to v_0 A_1 v_1 \dots v_{m-1} A_m v_m; w_0 B_1 w_1, \dots w_{n-1} B_n w_n\,(\sigma) \tag{7.1.1}$$

[1]Since we are not quite sure about the interpretation of all parts of the definition given by Schreiber (1975, 1976), we develop our theory independently of these papers.

where $m, n \geq 0$, $v_0, \ldots, v_m \in X^*$, $w_0, \ldots, w_n \in Y^*$, $A_1, \ldots, A_m, B_1, \ldots, B_n \in N$, and $\sigma\colon [n] \to [m]$ is a mapping such that if $\sigma(i) = \sigma(j)$ for some $i, j \in [n]$, then $B_i = B_j$; if $\sigma(i) = k$, then B_i is associated with A_k. Moreover, we may give σ as the n-tuple $(\sigma(1)\,\sigma(2)\,\ldots\,\sigma(n))$ formed by its values. Let us illustrate this by an example.

Example 7.1.2. If $A, B, C \in N$, $x \in X$, $y \in Y$ and $\sigma\colon [5] \to [4]$ is a mapping, then

$$A; B \to xABxxBA; CByCByB\,(\sigma)$$

is a possible STG-production. If $\sigma(1) = \sigma(3) = 1$, $\sigma(2) = \sigma(4) = 3$ and $\sigma(5) = 4$, then we may write the production also as

$$A; B \to xABxxBA; CByCByB\,(1\,3\,1\,3\,4).$$

Note that two C's are associated with the first A on the input side while no nonterminal is associated with the leftmost B. □

The *translation forms* of SG are triples (δ, γ, π), where $\delta \in (N \cup X)^*$, $\gamma \in (N \cup Y)^*$ and π is a mapping from $[|\gamma|_N]$ to $[|\delta|_N]$ that *associates* each occurrence of a nonterminal in γ with a unique occurrence of a nonterminal in δ. Moreover, only occurrences of the same nonterminal in γ can be associated with the same occurrence of a nonterminal in δ. If $|\gamma|_N = n$, then π may be written as $(\pi(1)\,\pi(2)\,\ldots\,\pi(n))$. The strings δ and γ are called, respectively, the *input form* and the *output form* of (δ, γ, π). Moreover, we refer to the occurrences of nonterminals in δ and γ as *input nonterminals* and *output nonterminals*, respectively. A translation form (δ, γ, π) in which $\gamma \in Y^*$ is denoted $(\delta, \gamma, \oslash)$ to show that there are no output nonterminals to be associated with input nonterminals. The set of all translation forms is defined inductively as follows:

(1) $(S, S', (1))$ is a translation form in which the only S' is associated with the only S.

(2) If $(\delta A \delta', \gamma_0 B \gamma_1 \ldots \gamma_{k-1} B \gamma_k, \pi)$ is a translation form, where $A, B \in N$, $\delta, \delta' \in (N \cup X)^*$, $k \geq 0$, $\gamma_0, \gamma_1, \ldots, \gamma_k \in (N \cup Y)^*$ and the displayed Bs are all the output nonterminals associated by π with the displayed A, and $A; B \to \alpha; \beta\,(\sigma)$ is a production in P, then $(\delta\alpha\delta', \gamma_0\beta\gamma_1 \ldots \gamma_{k-1}\beta\gamma_k, \pi')$ is a translation form in which the output nonterminals appearing in $\gamma_0, \gamma_1, \ldots, \gamma_k$ are associated with input nonterminals in δ and δ' as they were associated in the original form, and the nonterminals in the new β's are associated with the nonterminals in the new α as specified by σ. To express that this is the case, we write

$$(\delta A \delta', \gamma_0 B \gamma_1 \ldots \gamma_{k-1} B \gamma_k, \pi) \Rightarrow_{SG} (\delta\alpha\delta', \gamma_0\beta\gamma_1 \ldots \gamma_{k-1}\beta\gamma_k, \pi')\ .$$

This is called a *derivation step*, and it is *leftmost* if the explicit instance of A is the leftmost occurrence of any nonterminal in $\delta A \delta'$.

Furthermore, *derivations* and the *derivation relation* are defined as usual. Two STGs SG and SG' are said to be *equivalent* if $\lambda(SG) = \lambda(SG')$. The class of translations definable by STGs is denoted by $\lambda[STG]$.

The association of output nonterminals with input nonterminals shows which output nonterminals are rewritten together with a given input nonterminal. Note that there may be input nonterminals with which no output nonterminal is associated. The rewriting of such

an input nonterminal does not affect the output form. Note also that output nonterminals associated with the same input nonterminal are always rewritten the same way and so are their corresponding successor nonterminals. Hence, the output substrings produced by them are identical. This means that not all strings generated by SG^{out} (as a CFG) necessarily appear in the range of $\lambda(SG)$ and, as we shall see, the range is not necessarily context-free.

The possible further generalization that would allow different sets of input and output nonterminals would obviously not increase the power of the STGs since we can equally well use their union as a common set of nonterminals.

Various special types of STGs are obtained by imposing different restrictions on the association maps of the productions. These we define as follows (cf. Schreiber, 1975, 1976).

Definition 7.1.3. An STG-production of the form 7.1.1 is called

- *homogeneous* if $A = B$, and $B_i = A_j$ whenever $\sigma(i) = j$ for some $i \in [n]$ and $j \in [m]$,
- *linear* if σ is injective,
- *non-deleting* if σ is surjective,
- *order-preserving* if $\sigma(i) \leq \sigma(j)$ whenever $1 \leq i \leq j \leq n$, and
- *simple* if σ is an identity mapping (and hence $m = n$).

An STG is called *homogeneous*, *linear*, *non-deleting*, *order-preserving* or *simple* if all of its productions are, respectively, homogeneous, linear, non-deleting, order-preserving or simple, and we denote by $\lambda[hSTG]$, $\lambda[lSTG]$, $\lambda[nSTG]$, $\lambda[oSTG]$ and $\lambda[sSTG]$ the respective classes of translations. Further subclasses of STGs can be obtained by combining any of these restrictions. For example, $\lambda[hlSTG]$ is the class of translations defined by homogeneous linear STGs. □

In a homogeneous production each output nonterminal is associated with an occurrence of itself in the input form and homogeneous STGs are very similar to the syntax-connected transduction schemes of Schreiber (1975). If an STG is linear, no two output nonterminals of any translation form are linked with the same input nonterminal, and therefore the subderivations starting from any two output nonterminals are independent. In any translation form of a non-deleting STG, at least one output nonterminal is associated with each input nonterminal, and hence also the output form is affected by every derivation step.

Remark 7.1.4. We may also note the following.

(a) SCFGs are precisely the linear non-deleting STGs, that is, $\lambda[lnSTG] = \lambda[SCFG] = \mathcal{SDT}$.

(b) SDTSs become by a slight formal modification the homogenous linear non-deleting STGs, i.e., $\lambda[hlnSTG] = \lambda[SDTS] = \mathcal{SDT}$.

Usually, it is simpler to consider STGs in normal form.

Definition 7.1.5. An STG $SG = (N, X, Y, P, S, S')$ is in *normal form* if all of its productions are of the form

(i) $A; B \to \alpha; \beta\,(\sigma)$ with $\alpha, \beta \in N^*$, or of the form

(ii) $A; B \to u; w$ with $u \in X^*$ and $w \in Y^*$. □

The next theorem guarantees that there is an algorithm to convert any given STG to one in normal form.

Theorem 7.1.6. *For every STG SG one can construct an equivalent STG SG' in normal form in such a way that if SG is linear, non-deleting or simple, then so is SG'.*

Proof. Let $SG = (N, X, Y, P, S, S')$ be any STG. We construct $SG' = (N', X, Y, P', S, S')$ with $N \subseteq N'$ as follows. Let

$$A; B \to v_0 A_1 v_1 \dots v_{m-1} A_m v_m; w_0 B_1 w_1, \dots w_{n-1} B_n w_n \, (\sigma) \qquad (*)$$

be a production in P which is not in normal form. If $m \geq n$, we replace it by the production

$$A; B \to C_{v_0} A_1 C_{v_1} \dots C_{v_{m-1}} A_m C_{v_m}; C_{w_0} B_1 C_{w_1} \dots C_{w_{n-1}} B_n C_{w_n} \, (\sigma') \ ,$$

where $C_{v_0}, \dots, C_{v_m}, C_{w_0}, \dots, C_{w_m}$ are new nonterminals and

$$\sigma' = (1\, 2\sigma(1)\, 3\, 2\sigma(2)\ \dots\ 2n-1\, 2\sigma(n)\, 2n+1\, 2n+2\ \dots\ 2m+1) \ ,$$

and the productions $C_{v_j}; C_{w_j} \to v_j; w_j$ $(0 \leq j \leq n)$.

If $n > m$, we replace $(*)$ by the production

$$A; B \to C_{v_0} A_1 C_{v_1} \dots C_{v_{m-1}} A_m C_{v_m} \underbrace{C_\varepsilon \dots C_\varepsilon}_{n-m \text{ times}}; C_{w_0} B_1 C_{w_1} \dots C_{w_{n-1}} B_n C_{w_n} \, (\sigma'') \ ,$$

where $C_{v_0}, \dots, C_{v_m}, C_{w_0}, \dots, C_{w_n}, C_\varepsilon$ are new nonterminals and

$$\sigma'' = (1\, 2\sigma(1)\, 3\, 2\sigma(2)\ \dots\ 2m-1\, 2\sigma(m)\, 2m+1\, 2\sigma(m+1) 2m+3\ \dots\ n+m\, 2\sigma(n)\, n+m+1) \ ,$$

and the productions $C_{v_j}; C_{w_j} \to v_j; w_j$ $(0 \leq j \leq m)$ and $C_\varepsilon; C_{w_j} \to \varepsilon; w_j$ $(m+1 \leq j \leq n)$.

It should be obvious that in both cases, the new rules together produce exactly the effect of the original rule. Hence, $\lambda(SG) = \lambda(SG')$. □

Let us illustrate the above construction by an example.

Example 7.1.7. Let us consider the STG-production of Example 7.1.2. Since $n > m$, it is replaced by the productions

$$A; B \to C_x A C_\varepsilon B C_{xx} B C_\varepsilon A C_\varepsilon C_\varepsilon; C_\varepsilon C C_\varepsilon B C_y C C_\varepsilon B C_y B C_\varepsilon (1\,2\,3\,6\,5\,2\,7\,6\,9\,8\,10)$$
$$C_x; B_\varepsilon \to x; \varepsilon \qquad C_\varepsilon; C_\varepsilon \to \varepsilon; \varepsilon \qquad C_{xx}; B_y \to xx; y \qquad C_\varepsilon; C_y \to \varepsilon; y.$$

in normal form. □

Now we study the generative power of STGs. Proposition 3.3.9 proved that the ranges of SDTS-translations are context-free languages. Hence, the following example shows that a non-linear STG may define a translation not definable by any SDTS or SCFG.

Example 7.1.8. For each alphabet X, the translation $\lambda_c = \{(w, ww) \mid w \in X^*\} (\subseteq X^* \times X^*)$ is defined by the homogenous non-deleting STG $SG = (\{S, A\}, X, X, S, S)$, where

$$P = \{S; S \to A; AA\,(1\,1)\} \cup \{A; A \to xA; xA\,(1) \mid x \in X\} \cup \{A; A \to \varepsilon; \varepsilon\} \ .$$

For example, if $X = \{0, 1\}$, then we have derivations like

$$\begin{aligned}(S, S, (1)) &\Rightarrow_{SG} (A, AA, (1\,1)) \Rightarrow_{SG} (0A, 0A0A, (1\,1)) \Rightarrow_{SG} (00A, 00A00A, (1\,1)) \\ &\Rightarrow_{SG} (001, 001001, \oslash)\ .\end{aligned}$$

If $|X| \geq 2$, then the range $\{ww \mid w \in X^*\}$ of λ_c is not a CFL. More generally, for any $n \geq 1$, we get the translation $\{(w, w^n) \mid w \in X^*\}$ by replacing $S; S \to A; AA\,(1\,1)$ with $S; S \to A; \underbrace{AA\ldots A}_{n \text{ times}}(\underbrace{1\,1\ldots 1}_{n \text{ times}})$. □

The above example depends on the parallel rewriting on the output side caused by the nonlinearity of the first production. In fact, it is easy to simulate any *Indian parallel grammar* (IPG) of Siromoney and Krithivasan (1974) (cf. also Dassow et al., 1997) by a non-linear STG; recall that an IPG is just a CFG, but at each derivation step all occurrences of one of the nonterminals is rewritten in parallel using the same production. Given a CFG $CF = (N, X, P, S)$, let $SG = (N, X, X, P', S, S)$ be the STG in which P' is obtained from P by transforming any production $A \to \alpha$ of P into the STG-production $A; A \to B_1 \ldots B_k; \alpha\,(\sigma)$, where $\{B_1, \ldots, B_k\}$ is the set of nonterminals appearing in α and σ associates each output nonterminal with its unique occurrence in $B_1 \ldots B_k$. It should be clear that $\lambda(SG) = \{(\varepsilon, w) \mid w \in L\}$, where L is the language generated by CF in the Indian parallel mode. Therefore, if we denote by PCL the class of languages generated by IPGs, then $PCL \subseteq \text{Range}(STG)$.

Next, we observe that deleting productions do not affect the rewriting on the output side. Indeed, for every STG $ST = (N, X, Y, P, S, S')$, we can construct a non-deleting STG $ST' = (N \cup \{\boxdot\}, X, Y, P', S, S')$, where $\boxdot$ is a new nonterminal and P' is obtained from P as follows. Every non-deleting production of P is in P', too. Furthermore, for every deleting production $A; B \to \alpha; \beta\,(\sigma)$ in P of the form (7.1.1), the production $A; A' \to \alpha; \alpha' \boxdot^{|[m]-\sigma([n])|}$ (σ') is in P'. Now, σ' associates each occurrence of the nonterminal $\boxdot$ with distinct occurrences of nonterminals in α that did not have an association before, and the nonterminals in α' with those in α exactly as σ. Moreover, for every $A; B \to \alpha; \beta\,(\sigma)$ in P, $A; \boxdot \to \alpha; \varepsilon$ is in P'. It should be clear that ST and ST' generate the same output language. As an immediate consequence, we also note that the range of any linear STG is a CFL since the ranges of linear non-deleting STGs are CFLs.

These observations help us show that $\{x^p y^p z^p \mid p \geq 1\}$ cannot be the language generated by the output grammar of any STG. Indeed, assume there is an STG $ST = (N, X, \{x, y, z\}, P, S, S')$ with no useless productions such that $\text{Range}(ST) = \{x^p y^p z^p \mid p \geq 1\}$. Because $\text{Range}(ST)$ is not a CFL, ST must be a non-linear STG. Hence, there are $p_0 \geq 1$, $n \geq 2$ and a derivation

$$(S, S', (1)) \Rightarrow^*_{ST} (u_0 A u_1, u'_0 A' u'_1 \ldots u'_{n-1} A' u'_n, \underbrace{(1\ 1\ldots 1)}_{n \text{ times}}) \Rightarrow^*_{ST} (u_0 u_2 u_1, x^{p_0} y^{p_0} z^{p_0}, \oslash)$$

for some $A, A' \in N$, $u_0, u_1, u_2 \in X^*$ and $u'_0, u'_1, \ldots, u'_n \in \{x, y, z\}^*$. Let

$$L(A, A') = \{u' \in \{x, y, z\}^* \mid \exists u \in X^*, (A, A', (1)) \Rightarrow^*_{ST} (u, u', \oslash)\}\ .$$

It can be proved that $L(A, A') = \{d^k\}$ for some $k \geq 0$ and $d \in \{x, y, z\}$ as follows. First, $L(A, A')$ cannot contain two different letters since $n \geq 2$. Now, if $L(A, A')$ contains two

words d^k and d^l with $k \neq l$, then Range(ST) would include at least one word in which the number of occurrences of letter d is $n|l-k|$ greater than the number of occurrences of the other two letters, which contradicts our initial assumption.

Now, we can construct the linear STG $ST' = (N, X, \{x, y, z\}, P_0, S, S')$, where the productions in P_0 are obtained from those of P as follows. First, add to P_0 all linear productions in P. Next, for every production of the form (7.1.1) and for every $i \in [m]$ such that $|\sigma^{-1}(i)| \geq 2$, let u'_i be the unique element of the set $L(A_i, B_j)$ with $j \in \sigma^{-1}(i)$. Moreover, for every $i \in [m]$, let $\alpha_i := \begin{cases} \varepsilon, \text{ if } |\sigma^{-1}(i)| \geq 2 \\ A_i, \text{ otherwise} \end{cases}$, and for every $j \in [n]$, let $\alpha'_j := \begin{cases} u'_{\sigma(j)}, \text{ if } |\sigma^{-1}(\sigma(j))| \geq 2 \\ B_j, \text{ otherwise} \end{cases}$. Add to P_0 the linear production

$$A; B \to u_0\alpha_1 u_1 \dots u_{m-1}\alpha_m u_m; u'_0\alpha'_1 u'_1 \dots u'_{n-1}\alpha'_n u'_n \ (\sigma'),$$

where σ' associates the (remaining) nonterminals exactly as σ does in the original production (7.1.1). It is easy to see that Range(ST') = Range(ST) $\notin CFL$, a contradiction. Hence, there is no STG ST such that Range(ST) = $\{x^p y^p z^p \mid n \geq 1\}$.

However, the range of any STG ST is a contex-sensitive language. Indeed, a nondeterministic algorithm recognizing the language Range(ST) in linear time can be constructed by simulating the rewriting process that takes place on the output CFG grammar of ST. A special attention shall be paid to the output nonterminals associated with an input nonterminal. Hence, Range(STG) $\subseteq CS$ (see Mateescu and Salomaa, 1997).

We summarize the previous observations regarding the ranges of STGs in the following proposition.

Proposition 7.1.9. $CFL \cup PCL \subset$ Range(STG) $\subset CS$.

The inclusions must be proper. Note that $\{x^p y^p z^p \mid p \geq 1\}$ is a context-sensitive language (Mateescu and Salomaa, 1997, Example 2.1) that cannot be in Range(STG). Also, the range of the STG $ST = (S, A, \{x, y\}, \{x, y\}, P, S, S)$, where $P = \{S; S \to A; AA \ (1\,1), A; A \to AA; AA \ (1\ 2), A; A \to A; xAy \ (1), A; A \to \varepsilon; \varepsilon\}$, that is, $\{ww \mid w$ is a balanced string in $\{x, y\}^*\}$ is neither context-free nor parallel context-free.

Also the following property of STGs holds.

Theorem 7.1.10. *For every STG there effectively exists an equivalent homogenous STG.*

Proof. Given any STG $SG = (N, X, Y, P, S, S')$, we define a homogenous STG $SG' = (N', X, Y, P', (S, S'), (S, S'))$, where $N' := N \times (N \cup \{\boxdot\})$ ($\boxdot \notin N$) and P' is obtained from P as follows.

(1) For each production of the form (7.1.1) in P, we include in P' the production $(A, B); (A, B) \to \delta; \gamma \ (\sigma)$ with

$$\delta = v_0(A_1, C_1)u_1 \dots v_{m-1}(A_m, C_m)v_m$$

and

$$\gamma = w_0(A_{\sigma(1)}, B_1)w_1 \dots w_{n-1}(A_{\sigma(n)}, B_n)w_n \ ,$$

and where for each $k \in [m]$, $C_k = \begin{cases} B_j, & \text{if } \sigma(j) = k \ (j \in [n]), \\ \boxdot, & \text{if } k \text{ is not in the range of } \sigma. \end{cases}$

(2) For every $A; B \to \alpha; \beta\,(\sigma)$ in P, we add $(A, \boxdot); (A, \boxdot) \to \bar{\alpha}; \varepsilon$ to P', where $\bar{\alpha}$ is obtained from α by replacing each input nonterminal C with $(C, \boxdot)$.

Note that in (1), the value of C_k is always uniquely defined because $\sigma(i) = \sigma(j)$ implies $B_i = B_j$ $(i, j \in [n])$. The productions of type (2) are obtained from the productions of the input grammar SG^{in} and they can only be applied in situations in which there would be an independent input nonterminal in a derivation in SG. (Hence, they could be omitted if SG is non-deleting.)

Let $\mu\colon (N' \cup X)^* \to (N \cup X)^*$ and $\nu\colon (N' \cup Y)^* \to (N \cup Y)^*$ be homomorphisms such that $x\mu = x$ for $x \in X$, $(A, B)\mu = A$ for $(A, B) \in N'$, $y\nu = y$ for $y \in Y$ and $(A, B)\nu = B$ if $(A, B) \in N \times N$, and $(A, B)\nu = \varepsilon$ if $B = \boxdot$. Now, one can verify by induction on $n \geq 0$ that for each derivation

$$((S, S'), (S, S'), (1)) \Rightarrow_{SG'} (\delta_1, \gamma_1, \pi_1) \Rightarrow_{SG'} \ldots \Rightarrow_{SG'} (\delta_n, \gamma_n, \pi_n) \qquad (a)$$

in SG',

$$(S, S', (1)) \Rightarrow_{SG} (\delta_1\mu, \gamma_1\nu, \pi_1) \Rightarrow_{SG} \ldots \Rightarrow_{SG} (\delta_n\mu, \gamma_n\nu, \pi_n) \qquad (b)$$

is a valid derivation in SG, and that every derivation in SG is obtained this way from a derivation in SG'. From this it immediately follows that SG and SG' are equivalent. □

Corollary 7.1.11. *For every STG SG one can construct an equivalent homogenous STG SG' in normal form in such a way that if SG is linear, non-deleting, order-preserving or simple, then so is SG'.*

7.2 Synchronous tree transformation generators

We shall now consider a class of tree transformation generating devices that in a natural way correspond to synchronous translation generators. They generalize the STSGs and GSTSGs studied by Eisner (2003), Fülöp et al. (2010) and Shieber (2004) and presented in Section 5.3. This generalization follows the lead of (Schreiber, 1975, 1976), who extended SDTSs to syntax-connected transduction schemes (cf. Chapter 7.1) by linking each occurrence of a nonterminal in the output grammar with a unique occurrence of a nonterminal in the input grammar in such a way that only occurrences of the same nonterminal in the output can be associated with any given occurrence of a nonterminal in the input.

Definition 7.2.1. A *synchronous tree transformation generator* (STTG) is a system $TG = (N, \Sigma, X, \Omega, Y, P, S, S')$ specified as follows.

(1) Σ and Ω are ranked alphabets, and X and Y are leaf alphabets; Σ and X are the *input alphabets*, and Ω and Y are the *output alphabets*.

(2) N is a finite set of *nonterminals* such that $N \cap (\Sigma \cup X \cup \Omega \cup Y) = \emptyset$.

(3) $S, S' \in N$ are the *start symbols*.

(4) P is a finite set of *productions* $A; A' \to r; r'\,(\sigma)$ in which $A, A' \in N$, $r \in T_\Sigma(X \cup N)$, $r' \in T_\Omega(Y \cup N)$, and σ is a mapping that *associates* each occurrence of a nonterminal in r' with a unique occurrence of a nonterminal in r in such a way that only occurrences of the same nonterminal in r' can be associated with the same

occurrence of a nonterminal in r. If $r' \in T_\Omega(Y)$, we may omit σ and write the production as $A; A' \to r; r'$.

The *input grammar* of TG is the regular tree grammar $TG^{\text{in}} = (N, \Sigma, X, P^{\text{in}}, S)$, where P^{in} consists of the productions $A \to r$ such that $A; A' \to r; r'\,(\sigma)$ for some A', r' and σ. Similarly, $TG^{\text{out}} = (N, \Omega, Y, P^{\text{out}}, S')$, where P^{out} consists of those productions $A' \to r'$ for which $A; A' \to r; r'\,(\sigma) \in P$ for some A, r and σ, is the *output grammar* of TG. □

Much of what was said about STGs in Section 7.1 can be said, in an appropriately modified form, about STTGs. Let us first note that an STTG-production $A; A' \to r; r'\,(\sigma)$ can be written in the more explicit form

$$A; A' \to q[A_1, \ldots, A_m]; q'[A'_1, \ldots, A'_n]\,(\sigma) \ , \tag{7.2.1}$$

where $m, n \geq 0$, $A_1, \ldots, A_m, A'_1, \ldots, A'_n \in N$, $q \in \widetilde{T}_\Sigma(X \cup \Xi_m)$, $q' \in \widetilde{T}_\Omega(Y \cup \Xi_n)$, and $\sigma\colon [n] \to [m]$ is a mapping such that if $\sigma(i) = \sigma(j)$ for some $i, j \in [n]$, then $A'_i = A'_j$; if $\sigma(i) = k$, then A'_i is associated with A_k. Again, we may give σ as the n-tuple $(\sigma(1)\,\sigma(2)\,\ldots\,\sigma(n))$ formed by its values. Obviously, that (7.2.1) equals the production $A; A' \to r; r'\,(\sigma)$, means that $\text{yd}_N(r) = A_1 \ldots A_m$, $\text{yd}_N(r') = A'_1 \ldots A'_n$, $r = q[A_1, \ldots, A_m]$ and $r' = q'[A'_1, \ldots, A'_n]$.

For any given STTG $TG = (N, \Sigma, X, \Omega, Y, P, S, S')$, its *translation forms*, *input forms*, *output forms*, *input nonterminals*, *output nonterminals*, *(leftmost) derivation steps* and *(leftmost) derivations* are defined the same way as for an STG with the obvious modifications. In particular, a translation form of TG is a triple (s, s', π), where $s \in T_\Sigma(X \cup N)$, $s' \in T_\Omega(Y \cup N)$ and π is a mapping from $[|\text{yd}|_N(s')]$ to $|\text{yd}|_N(s)]$ that associates each output nonterminal with a unique input nonterminal in such a way that any two output nonterminal associated with the same input nonterminal are equal. A derivation step is of the form

$$(c(A), c'[A', \ldots, A'], \pi) \Rightarrow_{TG} (c(r), c'[r', \ldots, r'], \pi') \ ,$$

where $c \in C_\Sigma(X \cup N)$, $c' \in \widetilde{T}_\Omega(Y \cup N \cup \Xi_k)$ for some $k \geq 0$, $A; A' \to r; r'\,(\sigma)$ is a production in P, and the displayed k occurrences of A' are all the output nonterminals associated with the displayed input nonterminal A. Then, the *tree transformation defined* by TG is the relation

$$\tau(TG) := \{(t, t') \mid t \in T_\Sigma(X), t' \in T_\Omega(Y), (S, S', (1)) \Rightarrow^*_{TG} (t, t', \oslash)\} \ ,$$

and the *translation defined* by TG is

$$\lambda(TG) := \text{yd}(\tau(TG)) = \{(\text{yd}(t), \text{yd}(t')) \mid (t, t') \in \tau(TG)\} \ .$$

Let us denote the classes of tree transformations and translations definable by an STTG by $\tau[STTG]$ and $\lambda[STTG]$, respectively.

Homogeneous, *linear*, *non-deleting*, *order-preserving* and *simple* STTGs are also defined by the same conditions as the respective classes of STGs. The classes of tree transformations and translations definable by such special STTGs are naturally denoted by $\tau[hSTTG]$, $\lambda[hSTTG]$, $\tau[lSTTG]$, etc.

Furthermore, we say that an STTG $TG = (N, \Sigma, X, \Omega, Y, P, S, S')$ is in *normal form* if every production in it is of the form

(1) $A; A' \to r; r'\,(\sigma)$, where $r \in T_\Sigma(N)$ and $r' \in T_\Omega(N)$, or

(2) $A; A' \to s; s'$ with $s \in T_\Sigma(X)$ and $s' \in T_\Omega(Y)$.

Obviously, at the tree level, we cannot find an equivalent STTG in normal form for every STTG. We conjecture that the following weaker fact can be proven.

Theorem 7.2.2. *For every STTG TG one can construct an STTG TG' in normal form such that $\lambda(TG) = \lambda(TG')$.*

The next result can be obtained simply by imitating the proof of Theorem 7.1.10.

Theorem 7.2.3. *For every STTG there effectively exists an equivalent homogeneous STTG.*

As one could expect, STTGs are related to STGs as follows.

Theorem 7.2.4. $\lambda[STTG] = \lambda[STG]$.

Proof. For any STTG $TG = (N, \Sigma, X, \Omega, Y, P, S, S')$, let $ST = (N, X, Y, P', S, S')$ be the STG in which P' consists of all productions $A; A' \to \mathrm{yd}(r); \mathrm{yd}(r')\,(\sigma)$ obtained from a production $A; A' \to r; r'\,(\sigma)$ in P. To show that $\lambda(ST) = \lambda(TG)$, it suffices to verify the following facts by induction on the length of the given derivation.

Claim 1 For any $A, A' \in N$, $t \in T_\Sigma(X)$ and $t' \in T_\Omega(Y)$, if $(A, A', (1)) \Rightarrow^*_{TG} (t, t', \oslash)$, then $(A, A', (1)) \Rightarrow_{SG} *(\mathrm{yd}(t), \mathrm{yd}(t'), \oslash)$.

Claim 2 For any $A, A' \in N$, $v \in X^*$ and $w \in Y^*$, if $(A, A', (1)) \Rightarrow_{SG} (v, w, \oslash)$, then there are trees $t \in T_\Sigma(X)$ and $t' \in T_\Omega(Y)$ such that $\mathrm{yd}(t) = v$, $\mathrm{yd}(t') = w$ and $(A, A', (1)) \Rightarrow^*_{TG} (t, t', \oslash)$.

The other inclusion is also straightforward, and can be obtained by simply imitating the proof of Proposition 5.3.3.

□

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

CHAPTER 8

Concluding Remarks, Further Topics and Bibliographic Notes

In this chapter, we exhibit in Table 8.1.1 a summary of the main results presented in this thesis, along with several related topics and references that are proposed for further study.

8.1 A summary of the main results

In this section, we present a summary of the classes of translations and tree transformations naturally defined by means of tree bimorphisms that were studied in this monograph, along with some of their most useful properties. Also their connection with well-known synchronous grammars and tree transducers from the literature is outlined. Note that $\bigcup$ and $\bigcap$ denote the closure under union and intersection, respectively.

The inclusion relation between the classes of tree transformations of Table 8.1.1 and corresponding translations are shown in Proposition 6.4.2, Section 6.7.2 and the HASSE diagrams of Figures 3.3.1, 5.2.2, 5.2.3, 6.2.3, and 6.6.1.

When defined and connected in such an uniform manner, the formalisms (especially synchronous grammars) can be straightforwardly extended to assign probabilities to rules, whereby probability distributions can be defined, both on the translations and on tree transformations (Nederhof and Vogler, 2012).

8.2 Topics to be considered

Next, several ideas for further study are proposed.

8.2.1 Future work on Section 5.3.2

Already in Corollary 6.4.9, we exemplified how the connection between GSTSGs and the tree bimorphism formalism detailed in Chapter 6 can be used to improve the mathematical foundations of such grammars by showing various closure properties and decidability results. Next, we propose different research topics that might be of practical interest in view of the linguistic relevance of GSTSGs and their link with linear non-deleting XTT-transducers.

In the spirit of Theorems 3.3.11 and 2.4.6, one can consider *proper* GSTSGs without 'ε-rules' or unit rules, i.e., without rules of the forms $A; A' \to B; r$, $A; A' \to r; B'$ and $A; A' \to B; B'$. Could these rules be safely eliminated without affecting the generative/translation capacity? Also normal forms (cf. Theorem 3.3.12) can be defined and studied. Furthermore, a *simple* GSTSGs could be defined as a GSTSG GS in which in every production $A; A' \to r; r'\,(\sigma)$ in P, the permutation σ is the identity. How do they relate to simple SDTSs? What theoretical properties do they have? Are they practically relevant?

Bimorphisms	Translations	Transducers	Properties	Decidability
Alphabetic tree transformations $\tau[aTB]$ (Takahashi, 1972)				
Alphabetic Definitions 6.1.3 and 6.3.1	$\lambda[sSDTS]$ Theorem 6.3.7(ii)	QREL Theorem 6.3.9	COMP, SYM, PRES, PRES^{-1}, $\bigcup$ Theorem 6.3.3	equality Theorem 6.3.3
Permuting tree transformations $\tau[pTB]$ (Takahashi, 1972)				
Permuting Definitions 6.1.3 and 6.3.1	$\lambda[SDTS]$ Theorem 6.3.7(i)	p-TOP Theorem 6.3.11	COMP, SYM, PRES, PRES^{-1}, $\bigcup$ Theorem 6.3.3	equality Theorem 6.3.3
Σ-rational tree transformations Σ-RTTs (Steinby, 1984, 1986, 1990)				
Σ-rational Definition 6.7.1	$\lambda[rSDTS]$,$\lambda[FST]$ Theorems 6.7.2(viii) and 3.3.19	BOT Theorem 6.7.2(ix)	COMP, SYM, PRES, PRES^{-1}, $\bigcup$, $\bigcap$ Theorem 6.7.2	emptiness, finiteness, membership, inclusion, equality Theorem 6.7.2(x)
Σ-algebraic tree transformations Σ-ATTs (Steinby, 1984, 1986, 1990)				
Σ-algebraic Definition 6.7.3	$\lambda[sSDTS]$ Theorem 6.7.4(vi)	l-BOT Theorem 6.7.4(vii)	COMP, SYM, PRES, PRES^{-1}, $\bigcup$, $\bigcap$ Theorem 6.7.4	emptiness, finiteness, membership, inclusion, equality Theorem 6.7.4(ix)
Linear non-deleting tree transformations $\tau[lnTB]$ (Arnold and Dauchet, 1976a, 1982)				
Linear non-deleting Definition 6.1.3	$\lambda[SDTS]$, $\lambda[STSG]$, $\lambda[GSTSG]$ Theorem 6.4.8(ii), Corollary 5.3.9	ln-XTT Theorem 6.4.8(i)	SYM, PRES, PRES^{-1}, $\bigcup$ Theorem 6.4.1	emptiness, finiteness Corollary 6.4.9
Fine tree transformations $\tau[fTB]$ (Bozapalidis, 1992)				
Fine Definitions 6.1.3 and 6.5.1	$\lambda[SDTS]$ Theorem 6.5.4	Rahonis (2001) Theorem 6.5.4	COMP, SYM, PRES, PRES^{-1}, $\bigcup$ Theorem 6.5.4	equality Theorem 6.5.4
Quasi-alphabetic tree transformations $\tau[qaTB]$ (Steinby and Tîrnăucă, 2007, 2009)				
Quasi-alphabetic Definitions 6.1.3 and 6.2.1	$\lambda[SDTS]$, $\lambda[SCFG]$ Theorem 6.2.25	q-XTT Theorem 6.2.34, Proposition 6.2.23	COMP, SYM, PRES, PRES^{-1}, $\bigcup$ Theorems 6.2.17 and 6.2.6, Corollary 6.2.10, Proposition 6.2.11	emptiness, finiteness, membership Theorem 6.2.7

Table 8.1.1: Weak and strong equivalences of classes of tree transformations and translations defined by tree bimorphisms, synchronous grammars or tree transducers. Properties of such classes – a summary of results.

As for the closure properties of GSTSGs, it remains an open question if $\tau_1 \cap \tau_2$ is in $\tau[GSTSG]$ for any $\tau_1, \tau_2 \in \tau[GSTSG]$. Moreover, $\tau[GSTSG]$ is not closed under composition, but compositions with tree transformations belonging to subclasses of $\tau[GSTSG]$ may be worth studying. For example, if $\tau_1 \in \tau[GSTSG]$ and $\tau_2 \in \tau[STSG]$, one could verify if $\tau_1 \circ \tau_2$ and $\tau_2 \circ \tau_1$ are in $\tau[GSTSG]$. As for the decidability results for GSTSGs, the following questions are still open:

- Inclusion: Is $\tau(GS) \subseteq \tau(GS')$?
- Equivalence: Is $\tau(GS) = \tau(GS')$?
- Composition: Is $\tau(GS) \circ \tau(GS') \in \tau[GSTSG]$?
- Intersection: Is $\tau(GS) \cap \tau(GS') \neq \emptyset$?

8.2.2 Future work on Chapter 7

The following topics could be further explored for STGs. For example, the Hasse diagram showing the inclusion relations between the classes $\lambda[STG]$, $\lambda[lSTG]$, $\lambda[nSTG]$, $\lambda[oSTG]$, $\lambda[sSTG]$, $\lambda[lnSTG]$, $\lambda[loSTG]$, etc. could be explored. Also other known classes of translations may be placed into the diagram. In addition, one could verify if the composition of two STG-translations is an STG-translation. But how about compositions of, or with, special STG-translations?

Moreover, it could be interesting to investigate other closure properties of the classes $\lambda[xxSTG]$, where xxSTG is any type of STG introduced in Definition 7.1.3. In particular,

- $\lambda_1, \lambda_2 \in \lambda[xxSTG] \Rightarrow \lambda_1 \cup \lambda_2 \in \lambda[xxSTG]$?
- $\lambda_1, \lambda_2 \in \lambda[xxSTG] \Rightarrow \lambda_1 \cap \lambda_2 \in \lambda[xxSTG]$?
- $\lambda_1, \lambda_2 \in \lambda[xxSTG] \Rightarrow \lambda_1 \setminus \lambda_2 \in \lambda[xxSTG]$?

Also to characterize STG-translations in terms of tree bimorphisms may be quite rewarding in view of the results presented so far in Chapter 6. Furthermore, special classes of STGs could be characterized by such devices, and one shall relate them to known classes of tree bimorphims that were studied in Chapter 6.

The construction follows the lines of Section 6.2.2. Every derivation in an STG can obviously be replaced with a leftmost one that yields the same result and in which exactly the same productions are used as in the original derivation. Hence, we may view leftmost derivations as normal forms of derivations, and again we will represent them by production trees. These are defined as follows.

Let $SG = (N, X, Y, P, S, S')$ be an STG. We associate with SG a ranked alphabet Σ^{SG} that contains for each production $A; B \to \alpha; \beta\,(\sigma)$ in P a symbol $[A; B \to \alpha; \beta\,(\sigma)]$ of rank $|\alpha|_N$, i.e., for every $m \geq 0$,

$$\Sigma^{SG}_m = \{[A; B \to \alpha; \beta\,(\sigma)] \mid A; B \to \alpha; \beta\,(\sigma) \in P, |\alpha|_N = m\} \ .$$

Now, the sets $P(SG, A)$ of Σ^{SG}-trees associated with the nonterminals $A \in N$ are defined inductively as usual. The set of *production trees* of SG is the Σ^{SG}-tree language $P(SG) := P(SG, S)$.

Clearly, the production trees can be converted to production trees of the input grammar SG^{in} by the alphabetic tree homomorphism that maps $[A; B \to \alpha; \beta\,(\sigma)]$ to $[A \to \alpha]$. The following fact is also obvious.

Lemma 8.2.1. $P(SG) \in \text{Loc} \cap \text{DRec}^{\text{vf}}$. □

To see how a production tree represents a derivation in SG, we split it into a tree that represents the generated input and a tree that represents the generated output. This way a converse can be obtained by focusing on tree bimorphisms $(\varphi, P(SG), \psi)$ with φ and ψ of some type (a quasi-symbol-to-symbol tree homomorphism could be defined similarly as quasi-alphabetic tree homorphisms extended alphabetic and permuting tree homomorphisms; such a definition shall do the trick).

Now using the representation by tree bimorphisms, one can show that the membership problem "$(u, v) \in \lambda(ST)$?" is decidable for synchronous translation generators. If it is not, how about the various special types (linear, non-deleting, etc.)? How about SDTSs?

On the other hand, to describe a class of (extended) tree transducers that characterizes $\tau[STG]$ and find the appropriate special types corresponding to our subclasses of $\tau[STG]$ means to improve knowledge and applicability of such synchronous generators.

Moreover, let $\mathcal{S}_{stg}$ denote the STG-surface language operator, i.e., if $\mathcal{L}$ is a family of languages, then $\mathcal{S}_{stg}(\mathcal{L})$ is the family of all the languages $L\lambda\,(:= \{v \mid (u, v) \in \lambda, u \in L\})$, where $L \in \mathcal{L}$ and $\lambda \in \lambda[STG]$. The following topics on surface languages then might be worth investigating:

(a) Study/characterize/bound $\mathcal{S}_{stg}(CFL)$.

(b) Study/characterize/bound $\mathcal{S}_{stg}(Reg)$.

(c) Are $CFL \subseteq \mathcal{S}_{stg}(CFL) \subseteq \mathcal{S}^2_{stg}(CFL) \subseteq \ldots$ and $Reg \subseteq \mathcal{S}_{stg}(Reg) \subseteq \mathcal{S}^2_{stg}(Reg) \subseteq \ldots$ properly ascending chains?

(d) Are there some natural upper bounds or limits of the above chains?

Of course, the same questions can be considered for all the special classes of STGs (and perhaps for other families of languages).

As for the STTGs, the following directions could be further explored. One could exhibit and verify the HASSE diagram that will show the inclusion relations between the classes $\tau[STTG]$, $\tau[lSTTG]$, $\tau[nSTTG]$, $\tau[oSTTG]$, $\tau[sSTTG]$, $\tau[lnSTTG]$, and $\tau[loSTTG]$, for example. Also, other known classes of translations may be placed into the diagram. Furthermore, one could check if the composition of two STTG-tree transformation is an STTG-tree transformation. How about compositions of, or with, special STTG-transformations?

In addition, one could study other closure properties of the classes $\tau[xxSTTG]$. In particular,

- $\tau_1, \tau_2 \in \tau[xxSTTG] \;\Rightarrow\; \tau_1 \cup \tau_2 \in \tau[xxSTTG]$?
- $\tau_1, \tau_2 \in \tau[xxSTTG] \;\Rightarrow\; \tau_1 \cap \tau_2 \in \tau[xxSTTG]$?
- $\tau_1, \tau_2 \in \tau[xxSTTG] \;\Rightarrow\; \tau_1 \setminus \tau_2 \in \tau[xxSTTG]$?

STTG-tree transformations could be characterized in terms of tree bimorphisms in order to improve their mathematical background. Moreover, one could characterize also the special classes of STTGs and relate them to known classes of tree bimorphims. Again one could construct a tree bimorphism for any STTG by using the production trees (see also Section 6.2.2). The production trees are local, and also a converse can be obtained by focusing on tree bimorphisms (φ, R, ψ) of some type, where R is local. Then the following question could be addressed: assuming that the membership problem "$(s,t) \in \tau(TG)$?" is decidable, device a practical decision algorithm.

In order to improve the applicability of STTGs, one should describe a class of (extended) tree transducers that characterizes $\tau[STTG]$, and find the appropriate special types corresponding to our subclasses of $\tau[STTG]$.

Furthermore, let $\mathcal{S}_{sttg}$ denote the STTG-surface language operator, i.e., if $\mathcal{F}$ is a family of languages, then $\mathcal{S}_{sttg}(\mathcal{L})$ is the family of all the languages $T\tau$ $(:= \{s \mid (s,t) \in \tau, u \in T\})$, where $T \in \mathcal{F}$ and $\tau \in \tau[STTG]$. The following topics on surface languages could be investigated:

(a) Study/characterize/bound $\mathcal{S}_{sttg}(Rec)$.

(b) Is $Rec \subseteq \mathcal{S}_{sttg}(Rec) \subseteq \mathcal{S}^2_{sttg}(Rec) \subseteq \ldots$ a properly ascending chain?

(d) Are there some natural upper bounds of the above chain? Can we characterize the limit family?

Of course, the same questions can be revisited for all the special classes of STTGs (and perhaps for other families of languages).

8.3 Bibliographic notes

This section hopefully provide inspiration for further reading. Each subsection mostly contains the very first papers on the topic of the mentioned chapter and recent advances (up to 2014) of related formalisms that could not be covered by the thesis. The idea was also to point out possible applications of the notions introduced.

8.3.1 On Chapter 2

The theory of formal languages emerged in the 1950s when four different types of grammars were introduced as devices for defining natural languages (Chomsky, 1956, 1957, 1959b, a). The classification of these classes of grammars and languages into the first Chomsky hierarchy became clearer in Chomsky (1963) and Bar-Hillel (1964), and, since then, a standard reference in the field.

The concept of a finite-state system originates in the work of McCulloch and Pitts (1943). Huffman (1954), Mealy (1955) and Moore (1956) independently established finite automata as they are customary described and used nowadays. A first comprehensive study of finite automata (characterization, closure properties, and decidability results) was done by Rabin and Scott (1959).

The idea of a pushdown automaton was independently introduced by Oettinger (1961) and by Schützenberger (1963). The equivalence between context-free languages and pushdown machines was formally shown by Chomsky (1962) and Evey (1963). Deterministic

pushdown automata were introduced by Fischer (1963). One-turn pushdown automata are due to in Ginsburg and Spanier (1966).

Turing machines were defined by Turing (1936). Linear bounded automata were first studied by Myhill (1960), and later on their equivalence with context-sensitive languages was established by Landweber (1963) and Kuroda (1964).

Among the great variety of textbooks dedicated to formal languages and their applications one can find Aho and Ullman (1972), Aho and Ullman (1973), Bar-Hillel (1964), Eilenberg (1974), Ginsburg (1966), Harrison (1978), Hopcroft and Ullman (1969), Hopcroft et al. (2001), Hopcroft et al. (2006), Kozen (1997), Linz (2001), Martín-Vide et al. (2004), Rozenberg and Salomaa (1997), Salomaa (1973), Sipser (1997), Sudkamp (1997), Sudkamp (2006), Taylor (1998), Beesley and Karttunen (2003) and Roche and Schabes (1997).

8.3.2 On Chapter 3

An automatic translation means to mechanically transform strings of one language into another, and the concept is even older than the computing machines. With it, the theoretical framework for describing devices that define translations, generally called *translation theory*, developed quickly in the 1950s to be applied in linguistics, especially in machine translation (Weaver, 1955, Chomsky, 1957, Locke and Booth, 1955, Bar-Hillel, 1960), and in computer science such as in compiler design (Wilkes (1951), Sheridan (1959), Backus (1959), Backus et al. (1960, 1963), Irons (1961), Dijkstra et al. (1959); for overviews see Wilkes (1968), Sammet (1969), Wexelblat (1981), O'Hearn and Tennent (1997), and Priestley (2011), for example). Since then many translation classes have been introduced and studied theoretically and experimentally, but a perfect candidate to handle all ambiguities and syntactic differences encountered in natural languages is yet to be found.

Regular translations were first studied by Elgot and Mezei (1965) and numerous methods to define them were investigated since then: finite-state transducers (Elgot and Mezei, 1965, Ginsburg, 1962, 1966), right-linear syntax-directed translation schemata (Aho and Ullman, 1972), string bimorphisms (Nivat, 1968), matrix representations (Berstel, 1979), formal power series (Eilenberg, 1974, Salomaa and Soittola, 1978), and tree bimorphisms (Steinby (1986); cf. also Tîrnăucă (2011) and Proposition 6.2.28). In the literature, regular translations are known under different names: regular transductions, finite state transducer mappings, finite transductions, rational translations, rational transductions or rational relations (Aho and Ullman, 1972, Berstel, 1979, Choffrut, 1978, Mohri, 1997, Yu, 1997, Elgot and Mezei, 1965, Ginsburg, 1966, Eilenberg, 1974, Wintner, 2001, Roche and Schabes, 1997). For good overviews on their properties and numerous applications, especially in NLP, computer science and biology, the interested reader can consult, for example, Choffrut (1978), Berstel (1979), Cortes and Mohri (2005), Aho et al. (2006), Rozenberg and Salomaa (1997), Jurafsky and Martin (2009), Roche and Schabes (1997), Beesley and Karttunen (2003), Mohri (1996, 1997), and the references therein.

The concept of *syntax-directed translation* (see Aho and Ullman, 1972, Fülöp and Vogler, 1998, Maneth, 2004, for detailed explanations) appeared for the first time in the works of Irons (1961, 1963), and Barnett and Futrelle (1962) as a simple model of a compiler. Immediately after, their properties were systematically investigated when syntax-directed translations were defined by *syntax-directed translations schemata* (Čulík, 1965, 1966, 1968, Lewis II and Stearns, 1966, 1968, Younger, 1966, 1967, Paull, 1967, Aho and Ullman, 1969b) and (k-register) *pushdown transducers* (Aho and Ullman, 1968, 1969a). Later on, the math-

ematical foundations of syntax-directed translations (see Vere, 1970, Aho and Ullman, 1972, for overviews) were improved when they were characterized by means of *tree bimorphisms* by Steinby and Tîrnăucă (2007, 2009), and Maletti and Tîrnăucă (2009, 2010) (see Section 6.2 for a complete exposition).

Good introductory textbooks of applications of syntax-directed translations in computer science are Aho and Ullman (1972, 1973), Crespi-Reghizzi (2009) and Aho et al. (2006). On the other hand, the usefulness of this well-known translation class in NLP, especially in machine translation, is surveyed in the works of Yamada and Knight (2001), Chiang and Knight (2006), Jurafsky and Martin (2009), Huang (2008) and Chiang (2006, 2007).

Simple syntax-directed translations were systematically studied by Aho and Ullman (1969a), where they were defined by means of string bimorphisms and simple syntax-directed translation schemata. Their application in compiler design is surveyed by Aho and Ullman (1972) and Aho et al. (2006).

Translations defined by *inversion transduction grammars* are due to Wu (1995, 1997) and their usefulness, especially in alignment, was further studied by Wu and Fung (2005), Wu et al. (2006), Saers and Wu (2009), Saers et al. (2009) and Saers et al. (2012).

Linear syntax-directed translations were first investigated by Aho and Ullman (1972), and recently they have received an increased interest in the NLP community because of their utility in word alignment (see Saers, 2011, for an overview). Moreover, different ways to define them were studied: linear syntax-directed translation schemata (Saers et al., 2010), zipper finite-state transducers (Saers and Wu, 2011) and string bimorphisms (Theorem 3.3.15(iii)).

Extended versions of syntax-directed translation schemata were mostly considered by linguists in the recent years as an attempt of modeling more complicated natural language phenomena. *Synchronous context-free grammars* were introduced by Satta and Peserico (2005), and further investigated by Zhang et al. (2006), Zhang and Gildea (2007), Satta (2007), Huang (2008), Tîrnăucă (2011), and Huang et al. (2009), for example. They generate exactly the syntax-directed translations as it was suggested by Huang et al. (2009, p. 565) and Zhang et al. (2006, p. 258); this result is mentioned at least by Satta and Peserico (2005), Chiang and Knight (2006), Satta (2007), and Maletti and Tîrnăucă (2010). All the details are formally checked in Proposition 3.3.21.

To implement independent rewriting by means of partial deletion of syntactic structures, Melamed (2003) introduced *multitext grammars*. They simultaneously generate arbitrary n strings that are translations of each other via production rules of arbitrary length. Moreover, in the productions of such synchronous grammars both terminals as well as nonterminals can be linked except the empty string. However, multitext grammars of dimension 2 (Melamed, 2003, p. 81) are exactly the syntax-directed translation schemata of Aho and Ullman (1969a, 1972).

Schreiber (1975, 1976) extended syntax-directed translation schemata by introducing the *syntax-connected transduction schemes* as an attempt to better model the more complicated phases of a compiler such as the semantic analysis, code generation and optimization. These synchronous grammars were further extended and systematically studied in Chapter 7.1.

8.3.3 On Chapter 4

The theory of tree automata and tree languages emerged in the 1960s when J.R. Büchi and J.B. Wright observed that finite-state automata may be defined as unary algebras (cf.

also Mezei and Wright, 1967, Thatcher, 1973). Then, the generalization to tree automata was suggested independently by Doner (1965, 1970) and Thatcher and Wright (1965, 1968). Moreover, Thatcher (1973) and Rounds (1970b) expressed for the first time the idea to give elegant proofs using tree language theory for results on strings.

Many results presented in this chapter were obtained in various forms in early contributions by various authors. Even then many of them defined trees as terms and nowadays this formalism is the most common one. However, most authors do not use a separate frontier alphabet, but we followed Gécseg and Steinby (1984), Gécseg and Steinby (1997), Steinby (2004, 2005), for example, where the reasons (originating in universal algebra) are explained. Also, we felt that the tree bimorphism exposition (of Chapter 6) as formalism for describing natural language phenomena gains by using separate leaf alphabets (clearly representing the vocabularies). Moreover, in many presentations, symbols from the ranked alphabets may have more than one rank. This occurs especially in examples from linguistics where a symbol represent the same grammatical category in different sentences.

Alphabetic tree homomorphisms, often called *projections*, are due to Thatcher and Wright (1968), and general *tree homomorphisms* arose as special cases of tree transformations computed by tree transducers (see Thatcher, 1970, 1973, Engelfriet, 1975a, cf. also Section 5.2.3 for details). Other special types of tree homomorphisms, such as linear, nondeleting, strict, normalized, permuting, fine, quasi-alphabetic and symbol-to-symbol, can be found in the literature, cf., for example, Thatcher (1973), Arnold and Dauchet (1978a, 1982), Steinby (1986), Takahashi (1972, 1977), Steinby and Tîrnăucă (2007, 2009), Comon et al. (2007), and Bozapalidis (1992).

Deterministic and nondeterministic *bottom-up tree recognizers* were invented, and their equivalence was established, in the mid-1960s independently by Thatcher and Wright (1968), Doner (1965, 1970), and Magidor and Moran (1969). Nondeterministic *top-down tree recognizers* were introduced by Rabin (1969) and Magidor and Moran (1969). The latter paper investigated deterministic top-down tree recognizers and also showed the equivalence between nondeterministic top-down and bottom-up tree recognizers.

Regular tree grammars are due to Brainerd (1969), and their restriction motivated by linguistics - *tree substitution grammars*, appeared twenty years later in the work of Schabes (1990).

Also, alternative ways to specify *recognizable tree languages* were investigated (see Gécseg and Steinby, 1984, Gécseg and Steinby, 1997, for expositions): fixed point equations (Mezei and Wright, 1967, Eilenberg and Wright, 1967), congruences (Brainerd, 1968, Arbib and Give'on, 1968, Magidor and Moran, 1969, Kozen, 1977, Fülöp and Vágvölgyi, 1989a), logic formulae (Thatcher and Wright, 1968, Doner, 1970), regular expressions (Arbib and Give'on, 1968, Thatcher and Wright, 1968, Magidor and Moran, 1969) and elementary operations on tree languages in the Medvedev style (Costich, 1972).

The basic properties of recognizable tree languages were shown in the very first papers in tree language theory: Mezei and Wright (1967), Eilenberg and Wright (1967), Thatcher (1970), Magidor and Moran (1969), Thatcher and Wright (1968), Arbib and Give'on (1968), Doner (1970) and Brainerd (1968, 1969).

Local tree languages were investigated by Thatcher (1967, 1970), Magidor and Moran (1969), and Doner (1970).

The connection between context-free languages and recognizable tree languages was established using equations (Mezei and Wright, 1967, Magidor and Moran, 1969) and derivation trees (Thatcher, 1967, 1970, Doner, 1970). Engelfriet (1975c) and Steinby (1977) further

used various forms of production trees.

Context-free tree grammars are due to Rounds (1969, 1970a, b) and originate in the idea of Fischer (1968). Immediately, they were systematically studied by Engelfriet and Schmidt (1977a, b), Arnold and Dauchet (1976c, 1977, 1978a), Downey (1974) and Maibaum (1974).

Several handbooks (Gécseg and Steinby, 1984, Gécseg and Steinby, 2015, Gécseg and Steinby, 1997, Comon et al., 2007) were dedicated to survey the basic results related to trees and tree language theory. Also, Thatcher (1973), Engelfriet (1975c, 2015), and Steinby (2004) offer good introductions into the field.

8.3.4 On Chapter 5

To formally define and study how a (parse) tree is transformed into another (parse) tree has become customary since 1960s when the theory of program schemata, syntax-directed translations, attributed grammars and semantic interpretation emerged. Among the first papers dealing with tree transformations and their applications we mention: Thatcher (1970, 1973), Rounds (1968, 1970a, 1973), Kopřiva (1970), Engelfriet (1971, 1975a, c), Baker (1973a, b), Rosen (1971), Ogden and Rounds (1972), Martin and Vere (1970), Joshi et al. (1972, 1975), Kosaraju (1973), Alagić (1973, 1975), and Bertsch (1973). Recently, the computational linguistics community has extensively turned to tree-based approaches (many times enriched with weights) for NLP applications such as: syntax-based machine translation (Abeillé et al., 1990, Shieber and Schabes, 1990b, Yamada and Knight, 2001, Graehl and Knight, 2004, Knight and Graehl, 2005, Eisner, 2003, Melamed, 2003, Alshawi et al., 2000, Gildea, 2003, Shieber, 2004, 2006, May, 2010, DeNeefe, 2011), summarization (Knight and Marcu, 2002), question answering (Echihabi and Marcu, 2003), language modeling (Charniak, 2001), paraphrasing (Pang et al., 2003), and natural language generation (Shieber and Schabes, 1990a, 1991, Langkilde and Knight, 1998, Bangalore and Rambow, 2000, Corston-Oliver et al., 2002).

Top-down tree transducers were introduced by Rounds (1968, 1970a) and Thatcher (1970), as models of transformational grammars of Chomsky (1957). *Bottom-up tree transducers* are due to Thatcher (1973). Schreiber (1975, 1976) defined the *generalized finite tree transducer*, which combines both devices: a move applies a top-down tree transducer rule followed by a bottom-up tree transducer rule. As it was mentioned in the case of tree languages and tree recognizers, many of the authors dealing with tree transducers do not use a separate leaf alphabet. Also, some of them allow symbols from the ranked alphabets to have more than one rank.

The basic properties of such tree transducers and their variants (deterministic, total, linear, non-deleting, relabeling, etc.), inclusion relations between various classes and many composition and decomposition results were first studied by Engelfriet (1971, 1975a, c) and Baker (1973a, b, 1978b, 1979), and later by Vágvölgyi (1986), Fülöp and Vágvölgyi (1987, 1991), Engelfriet (1982) and Fülöp (1991).

Top-down tree transducers with regular look-ahead are due to Engelfriet (1975b, 1977), and this notion has been developed further in various ways: Bloem and Engelfriet (2000), Fülöp and Vágvölgyi (1989b, c, d), Engelfriet and Vogler (1985, 1986, 1987, 1988), Maletti et al. (2009), Vágvölgyi (1992), Maletti (2007, 2008), and Engelfriet et al. (2016).

Extended top-down tree transducers were first considered by Dauchet (1975), Arnold and Dauchet (1976a, 1982), and Lilin (1978), and later by Graehl and Knight (2004). Extended top-down transducers with regular look-ahead were introduced by Maletti et al. (2009).

Variants of such tree transducers, useful closure properties, composition and decomposition results, and relations with the classical (top-down and bottom-up) tree transducers were studied since then, also by Knight and Graehl (2005), Maletti (2007, 2008, 2010b, 2012), Maletti et al. (2009), Maletti and Vogler (2009), Fülöp and Maletti (2013), Graehl et al. (2008), Lagoutte et al. (2012), and Tîrnăucă (2009) (cf. Knight, 2007, Maletti, 2010a, Razmara, 2011, for overviews and further references).

In a similar manner, bottom-up tree transducers were generalized to (*extended*) *multi bottom-up tree transducers* (Lilin, 1978, Fülöp et al., 2004, 2005, Maletti, 2007, Engelfriet et al., 2008) by allowing the states to have an arbitrary rank ("multi") and by permitting trees of arbitrary height on both sides of the rules ("extended"). Variants of them, properties appealing for NLP applications including composition and decomposition results, and connections with other well-known classes of tree transformations were systematically studied by Fülöp et al. (2004, 2005), Maletti (2008, 2010c, 2011a, b, 2012), Engelfriet et al. (2009), and surveyed by Maletti (2010a).

Generalized syntax-directed translators were defined by Aho and Ullman (1969c), Aho and Ullman (1971), Martin and Vere (1970) and Baker (1978a) to transform trees into strings in a top-down fashion. As also surveyed by Gécseg and Steinby (1984), Gécseg and Steinby (2015) and Gécseg and Steinby (1997), they are technically useful for studying certain properties of tree transformations classes computed by classical tree transducers and, in compiler design, as mathematical models of syntax-directed translations of context-free grammars. Recently, such transducers were called (*top-down*) *tree-to-string transducers* in the computational linguistics community (Knight and Graehl, 2005), and used in several machine translation models, especially when there does not exist a good parser for the output language or trees are not available on the output side of a training corpus: Yamada and Knight (2001), Galley et al. (2004), Galley et al. (2006), and Marcu et al. (2006).

Even more powerful tree transducer models were introduced and studied over time. *Macro tree transducers* were introduced by Engelfriet (1980), and a comprehensive discussion of them can be found in Engelfriet and Vogler (1985), Fülöp and Vogler (1998), and Maneth (2004), for example. Restricted versions of such transducers were recently applied in machine translation by Shieber (2006), Maletti (2012), and Nederhof and Vogler (2012). Other very powerful transducer devices are, for example: *alphabetic tree transducers* and *synchronized tree transducers* (Rahonis, 2001), *attributed tree transducers* (Fülöp, 1981, Engelfriet, 1980, 1981), *modular tree transducers* (Engelfriet and Vogler, 1991), and *ground tree transducers* (Dauchet and Tison, 1990, Dauchet et al., 1990). However, our list is far from comprehensive, and the interested reader can find more models and further references in the works of Gécseg and Steinby (1997), Maneth (2004), Fülöp and Vogler (1998), Comon et al. (2007), and Fülöp (2004), for example.

Gécseg and Steinby (1984), Gécseg and Steinby (1997), Gécseg and Steinby (2015), Fülöp and Vogler (1998) and Comon et al. (2007) are the classical textbooks covering the tree transducer theory, but also Thatcher (1973), Engelfriet (1975a, c), Fülöp (2004), Steinby (2005), Knight (2007), Maletti (2010a) and Razmara (2011) survey the basic tree transducers discussed in this chapter, as well as their applications.

The process of simultaneous generation of pairs of strings and trees, and its applications in compiler design and NLP is reviewed by Satta (2004, 2009), and Chiang and Knight (2006), Chiang (2006) and Razmara (2011) offer good and brief introductions into the features of the most well-known synchronous grammars and their usefulness in representing natural language phenomena.

Synchronous tree-substitution grammars were mentioned for the first time in the work of Schabes (1990), and Shieber (2004) and Eisner (2003) started to use them to naturally model local rotations in the form of subject-object swapping, a very common phenomenon occurring in translations between natural languages. Enriched with weights, synchronous tree-substitution grammars further found their use in NLP in alignment (Gildea, 2003), statistical (syntax-based) machine translation (DeNero et al., 2009, Graehl et al., 2008, Liu et al., 2009, Chiang, 2010, Zhang et al., 2006, 2007, 2013), annotation (Kato and Matsubara, 2010b, a, Krasnowska et al., 2012) and sentence compression (Cohn and Lapata, 2009, Yamangil and Shieber, 2010, Feblowitz and Kauchak, 2013), for example.

Generalized synchronous tree-substitution grammars can be found in the literature under the name of synchronous tree-substitution grammars with states, and their weighted version was defined and systematically studied for solving various tasks in statistical machine translation by Maletti (2010c, 2011a) and Fülöp et al. (2010).

Synchronous tree-adjoining grammars were independently introduced by Shieber and Schabes (1990b, a) and Abeillé et al. (1990) as a more powerful type of synchronous tree substitution grammar that also allows tree adjunction (Joshi and Schabes, 1997) as operation at its nodes, besides substitution. Used to explicitly model even the most complicated natural language phenomena encountered, for example, in semantic interpretation, natural language generation, paraphrasing and machine translation, the synchronous tree-adjoining grammars were further studied by Schabes and Joshi (1990), Shieber and Schabes (1991), Shieber (1994), Abeillé (1994), TAG+3 (1994), Park (1995), Dras (1997, 1999), Schuler (1999), Nesson et al. (2008), Shieber (2006), and Maletti (2010b), for example. Their stochastic version is due to Shieber (2007) and the applicability of the weighted model to statistical machine translation is surveyed by DeNeefe (2011) (see also the references therein). However, it is generally argued that such generating devices are algorithmically expensive for practical use in machine translation.

Therefore, in the last decade, many other devices that generate pairs of trees in a synchronized manner were proposed having as motivation various problems in NLP, especially in (statistical) machine translation. We mention *synchronous context-free tree grammars* (Nederhof and Vogler, 2012), *generalized multitext grammars* (Melamed et al., 2004), *syntax-directed translations with extended domain of locality* (Huang et al., 2006), *synchronous description tree grammars* (Rambow et al., 1995, Rambow and Satta, 1996), *synchronous dependency insertion grammars* (Ding and Palmer, 2005), *synchronous tree-insertion grammars* (Nesson et al., 2006, DeNeefe and Knight, 2009), *synchronous tree sequence substitution grammars* (Raoult, 1992, Zhang et al., 2008b, a, Sun et al., 2009, Maletti, 2011c), and *synchronous forest substitution grammars* (Maletti, 2013), for example.

Finally, bear in mind that very often tree transducers and synchronous grammars are connected via tree bimorphisms, the formalism that we studied in detail in Chapter 6, in an attempt to improve the mathematical foundations of both devices and subsequently, their relevance in practice. Most of these relations are proved or referred to throughout this chapter and especially in Section 6.7 (cf. Tîrnăucă, 2008, for a brief exposition).

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

Bibliography

Abeillé, A. (1994). Syntax or semantics? Handling nonlocal dependencies with MCTAGs or synchronous TAGs. *Computational Intelligence*, 10(4):471–485. (Cited on page 169.)

Abeillé, A., Schabes, Y., and Joshi, A. K. (1990). Using lexicalized tags for machine translation. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING 1990)*, volume 3, pages 1–6. University of Helsinki, Finland. (Cited on pages 18, 37, 91, 93, 147, 167 and 169.)

Aho, A. V. (1967). Indexed grammars - an extension of context free grammars. In *Conference Record of 1967 Eighth Annual Symposium on Switching and Automata Theory (SWAT-FOCS)*, pages 21–31. IEEE Computer Society. (Cited on page 75.)

Aho, A. V. (1968). Indexed grammars - an extension of context-free grammars. *Journal of the ACM*, 15(4):647–671. (Cited on page 75.)

Aho, A. V. (1969). Nested stack automata. *Journal of the ACM*, 16(3):383–406. (Cited on page 75.)

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1969). A general theory of translation. *Mathematical Systems Theory*, 3(3):193–221. (Cited on page 50.)

Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools*. Pearson Education, Inc., New Jersey, 2nd edition. (Cited on pages 31, 35, 39, 40, 42, 164 and 165.)

Aho, A. V. and Ullman, J. D. (1968). Automaton analogs of syntax directed translation schemata. In *Conference Record of 1968 Ninth Annual Symposium on Switching and Automata Theory (SWAT-FOCS)*, pages 143–159. IEEE Computer Society. (Cited on pages 50 and 164.)

Aho, A. V. and Ullman, J. D. (1969a). Properties of syntax directed translations. *Journal of Computer and System Sciences*, 3(3):319–334. (Cited on pages 41, 47, 49, 50, 164 and 165.)

Aho, A. V. and Ullman, J. D. (1969b). Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3(1):37–56. (Cited on pages 45, 46, 47, 48, 50 and 164.)

Aho, A. V. and Ullman, J. D. (1969c). Translations on a context free grammar. In *Proceedings of the First Annual ACM Symposium on Theory of Computing (STOC '69)*, pages 93–112. ACM, New York, NY, USA. (Cited on pages 142 and 168.)

Aho, A. V. and Ullman, J. D. (1971). Translations on a context-free grammar. *Information and Control*, 19(5):439–475. (Cited on pages 142 and 168.)

Aho, A. V. and Ullman, J. D. (1972). *Parsing*, volume I of *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, New Jersey. (Cited on pages 16, 17, 23, 24, 26, 31, 35, 36, 37, 38, 39, 40, 42, 45, 46, 47, 48, 49, 50, 51, 56, 72, 149, 164 and 165.)

Aho, A. V. and Ullman, J. D. (1973). *Compiling*, volume II of *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, New Jersey. (Cited on pages 35, 40, 42, 164 and 165.)

Alagić, S. (1973). Natural state transformations. Technical Report 73B-2, University of Massachusetts at Amherst, Amherst, MA. (Cited on page 167.)

Alagić, S. (1975). Natural state transformations. *Journal of Computer and System Sciences*, 10(2):266–307. (Cited on page 167.)

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). OpenFst: A general and efficient weighted finite-state transducer library. In Holub, J. and Žďárek, J., editors, *Proceedings of the 12th International Conference on Implementation and Application of Automata (CIAA)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer, Berlin. (Cited on page 39.)

Alshawi, H., Bangalore, S., and Douglas, S. (2000). Learning dependency translation models as collections of finite state head transducers. *Computational Linguistics*, 26(1):45–60. (Cited on page 167.)

Arbib, M. A. and Give'on, Y. (1968). Algebra automata I: Parallel programming as a prolegomena to the categorical approach. *Information and Control*, 12(4):331–345. (Cited on page 166.)

Arnold, A. (1977). *Systèmes d'equations dans le magmoide. Ensembles rationnels et algébriques d'arbres*. Ph.D. thesis, Université de Lille 1, Lille, France. (Cited on page 146.)

Arnold, A. and Dauchet, M. (1976a). Bi-transductions de forêts. In Michaelson, S. and Milner, R., editors, *Proceedings of the Third International Colloquium on Automata, Languages and Programming (ICALP)*, pages 74–86. Edinburgh University Press. (Cited on pages 20, 86, 101, 103, 131, 140, 141, 142, 145, 146, 160 and 167.)

Arnold, A. and Dauchet, M. (1976b). Transductions de forêts reconnaissables monadiques. forêts corégulières. *RAIRO Informatique Théorique et Applications*, 10(1):5–28. (Cited on page 131.)

Arnold, A. and Dauchet, M. (1976c). Un théorème de duplication pour les forêts algébriques. *Journal of Computer and System Sciences*, 13(2):223–244. (Cited on page 167.)

Arnold, A. and Dauchet, M. (1977). Un théorème de Chomsky-Schützenberger pour les forêts algébriques. *CALCOLO*, 14(2):161–184. (Cited on page 167.)

Arnold, A. and Dauchet, M. (1978a). Forêts algébriques et homomorphismes inverses. *Information and Control*, 37(2):182–196. (Cited on pages 59, 135, 166 and 167.)

Arnold, A. and Dauchet, M. (1978b). Théorie des magmoïdes. *Informatique Théorique et Applications*, 12(3):235–257. (Cited on pages 141 and 146.)

Arnold, A. and Dauchet, M. (1979). Théorie des magmoïdes (II). *Informatique Théorique et Applications*, 13(2):135–154. (Cited on pages 141 and 146.)

Arnold, A. and Dauchet, M. (1982). Morphismes et bimorphismes d'arbres. *Theoretical Computer Science*, 20(1):33–93. (Cited on pages 19, 20, 59, 101, 103, 131, 137, 140, 141, 142, 145, 146, 147, 160, 166 and 167.)

Baader, F. and Nipkow, T. (1998). *Term Rewriting and All That.* Cambridge University Press, New York, NY. (Cited on page 55.)

Backus, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zürich ACM-GAMM conference. In *Proceedings of the International Conference on Information Processing (IFIP)*, pages 125–131. UNESCO (Paris), R. Oldenbourg (München) and Bütterworths (London). (Cited on page 164.)

Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., van Wijngaarden, A., Woodger, M., and Naur, P. (1960). Report on the algorithmic language ALGOL 60. *Communications of the ACM*, 3(5):299–314. (Cited on pages 17 and 164.)

Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., van Wijngaarden, A., Woodger, M., and Naur, P. (1963). Revised report on the algorithm language ALGOL 60. *Communications of the ACM*, 6(1):1–17. (Cited on pages 17 and 164.)

Baker, B. S. (1973a). *Tree transductions and families of tree languages.* Ph.D. thesis, Harvard University, Cambridge, MA, USA. (Cited on page 167.)

Baker, B. S. (1973b). Tree transductions and families of tree languges. In Aho, A. V., Borodin, A., Constable, R. L., Floyd, R. W., Harrison, M. A., Karp, R. M., and Strong, H. R., editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC)*, pages 200–206. ACM. (Cited on page 167.)

Baker, B. S. (1978a). Generalized syntax directed translation, tree transducers, and linear space. *SIAM Journal on Computing*, 7(3):376–391. (Cited on pages 142 and 168.)

Baker, B. S. (1978b). Tree transducers and tree languages. *Information and Control*, 37(3):241–266. (Cited on pages 89 and 167.)

Baker, B. S. (1979). Composition of top-down and bottom-up tree transductions. *Information and Control*, 41(2):186–213. (Cited on pages 19, 89 and 167.)

Bangalore, S. and Rambow, O. (2000). Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING), 2 Volumes*, pages 42–48. Morgan Kaufmann. (Cited on page 167.)

Bar-Hillel, Y. (1960). The present status of automatic translation of languages. *Advances in Computers*, 1:91–163. (Cited on page 164.)

Bar-Hillel, Y. (1964). *Language and Information.* Addison-Wesley, Reading, MA. (Cited on pages 18, 163 and 164.)

Barnett, M. P. and Futrelle, R. P. (1962). Syntactic analysis by digital computer. *Communication of the ACM*, 5(10):515–526. (Cited on pages 17, 37, 40 and 164.)

Beesley, K. R. and Karttunen, L. (2003). *Finite State Morphology.* Center for the Study of Language and Information (CSLI Publications), Stanford. (Cited on pages 39 and 164.)

Berstel, J. (1979). *Transductions and Context-Free Languages.* B.G. Teubner, Stuttgart. (Cited on pages 35, 37, 38, 39, 101, 120, 144 and 164.)

Bertsch, E. (1973). Some considerations about classes of mappings between context-free derivation systems. In Böhling, K.-H. and Indermark, K., editors, *Gesellschaft für Informatik, 1. Fachtagung über Automatentheorie und Formale Sprachen*, volume 2 of *Lecture Notes in Computer Science*, pages 278–283. Springer. (Cited on page 167.)

Bloem, R. and Engelfriet, J. (2000). A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *Journal of Computer and System Sciences*, 61(1):1–50. (Cited on page 167.)

Bozapalidis, S. (1988). Sur une classe de transformations d'arbres. *Informatique théorique et applications*, 22(1):43–47. (Cited on page 146.)

Bozapalidis, S. (1992). Alphabetic tree relations. *Theoretical Computer Science*, 99(2):177–211. (Cited on pages 19, 21, 59, 101, 103, 106, 108, 110, 120, 122, 128, 135, 137, 160 and 166.)

Bozapalidis, S. and Rahonis, G. (1994). On two families of forests. *Acta Informatica*, 31(3):235–260. (Cited on pages 108, 135 and 137.)

Brainerd, W. S. (1968). The minimalization of tree automata. *Information and Control*, 13(5):484–491. (Cited on page 166.)

Brainerd, W. S. (1969). Tree generating regular systems. *Information and Control*, 14(2):217–231. (Cited on pages 66 and 166.)

Burck, G. (1965). *The computer age and its potential for management.* New York: Harper & Row. (Cited on page 16.)

Burris, S. and Sankappanavar, H. (1981). *A Course in Universal Algebra.* Springer, New York. (Cited on pages 55 and 143.)

Charniak, E. (2001). Immediate-head parsing for language models. In *Association for Computational Linguistic, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference (ACL 2001)*, pages 116–123. Morgan Kaufmann Publishers. (Cited on page 167.)

Chiang, D. (2006). An introduction to synchronous grammars. Personal communication: Notes of the tutorial given at the Coling/ACL 2006 (Joint conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics) with Kevin Knight, `http://www.isi.edu/~chiang/papers/synchtut.pdf`. (Cited on pages 18, 44, 77, 92, 93, 103, 165 and 168.)

Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228. (Cited on pages 40, 44, 103 and 165.)

Chiang, D. (2010). Learning to translate with source and target syntax. In Hajic, J., Carberry, S., and Clark, S., editors, *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010)*, pages 1443–1452. The Association for Computer Linguistics. (Cited on page 169.)

Chiang, D. and Knight, K. (2006). An introduction to synchronous grammars. Tutorial given at *Coling/ACL 2006* (Joint conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics), `http://www.isi.edu/~chiang/papers/synchtut-slides.pdf`. (Cited on pages 18, 40, 52, 77, 92, 93, 103, 165 and 168.)

Choffrut, C. (1978). *Contribution á l'étude de quelques familles remarquables de fonctions rationnelles*. Ph.D. thesis, Université Paris 7, Paris, France. (Cited on pages 35, 37, 38 and 164.)

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, IT2(3):113–124. (Cited on page 163.)

Chomsky, N. (1957). *Syntactic Structures*. Mouton and Co., The Hague. (Cited on pages 15, 16, 17, 163, 164 and 167.)

Chomsky, N. (1959a). A note on phrase structure grammars. *Information and Control*, 2(4):393–395. (Cited on pages 29 and 163.)

Chomsky, N. (1959b). On certain formal properties of grammars. *Information and Control*, 2(2):137–167. (Cited on pages 29 and 163.)

Chomsky, N. (1962). Context-free grammars and pushdown storage. Quarterly Progress Report 65: pp. 187–194, MIT Research Laboratories of Electronics, Cambridge, MA. (Cited on page 163.)

Chomsky, N. (1963). Formal properties of grammars. In Luce, R. D., Bush, R. R., and Galanter, E., editors, *Handbook of Mathematical Psychology*, volume 2, pages 323–418. Wiley, New York. (Cited on page 163.)

Cohn, T. and Lapata, M. (2009). Sentence compression as tree transduction. *Journal of Artificial Intelligence Research*, 34:637–674. (Cited on page 169.)

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Löding, C., Tison, S., and Tommasi, M. (2007). *Tree Automata—Techniques and Applications*. Available at `http://tata.gforge.inria.fr/`. (Cited on pages 18, 55, 59, 63, 77, 80, 81, 91, 123, 140, 142, 143, 166, 167 and 168.)

Coquidé, J.-L., Dauchet, M., Gilleron, R., and Vágvölgyi, S. (1994). Bottom-up tree pushdown automata: Classification and connection with rewrite systems. *Theoretical Computer Science*, 127(1):69–98. (Cited on page 75.)

Corston-Oliver, S., Gamon, M., Ringger, E. K., and Moore, R. C. (2002). An overview of Amalgam: A machine-learned generation module. In *Proceedings of the International Natural Language Generation Conference (INLG 2002)*, pages 33–40. Association for Computational Linguistics. (Cited on page 167.)

Cortes, C. and Mohri, M. (2005). Weighted finite-state transducers in computational biology. Tutorial given at the 13th International Conference on Intelligent Systems for Computational Biology (*ISMB 2005*). (Cited on pages 39 and 164.)

Costich, O. L. (1972). A Medvedev characterization of sets recognized by generalized finite automata. *Mathematical Systems Theory*, 6(1-2):263–267. (Cited on page 166.)

Crespi-Reghizzi, S. (2009). *Formal Languages and Compilation*. Texts in Computer Science. Springer, London. (Cited on pages 35, 39, 40 and 165.)

Culik II, K. and Cohen, R. S. (1973). LR-regular grammars–an extension of LR(k) grammars. *Journal of Computer and System Sciences*, 7:66–96. (Cited on page 82.)

Culik II, K. and Kari, J. (1997). Digital images and formal languages. In Salomaa, A. and Rozenberg, G., editors, *Handbook of Formal Languages*, volume 3: Beyond Words, pages 599–616. Springer, Berlin. (Cited on page 39.)

Dassow, J., Păun, G., and Salomaa, A. (1997). Grammars with controlled derivations. In Salomaa, A. and Rozenberg, G., editors, *Handbook of Formal Languages*, volume 2: Linear Modeling Background and Applications, pages 101–114. Springer, Berlin. (Cited on page 153.)

Dauchet, M. (1975). *Transductions inversibles de forêts*. Master thesis, Université de Lille, Lille, France. (Cited on pages 86, 131 and 167.)

Dauchet, M., Heuillard, T., Lescanne, P., and Tison, S. (1990). Decidability of the confluence of finite ground term rewrite systems and of other related term rewrite systems. *Information and Computation*, 88(2):187–201. (Cited on page 168.)

Dauchet, M. and Mongy, J. (1979). Transformations de noyaux reconnaissables. In Budach, L., editor, *Fundamentals of Computation Theory, FCT'79, Proceedings of the Conference on Algebraic, Arthmetic, and Categorial Methods in Computation Theory*, pages 92–98. Akademie-Verlag, Berlin. (Cited on page 131.)

Dauchet, M. and Tison, S. (1990). The theory of ground rewrite systems is decidable. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90)*, pages 242–248. IEEE Computer Society. (Cited on pages 141 and 168.)

Dauchet, M. and Tison, S. (1992). Structural complexity of classes of tree languages. In Nivat, M. and Podelski, A., editors, *Tree Automata and Languages*, pages 327–354. North-Holland, Amsterdam. (Cited on pages 19, 101, 103 and 140.)

De Mori, R., Giordana, A., and Laface, P. (1982). Speech segmentation and interpretation using a semantic syntax-directed translation. *Pattern Recognition Letters*, 1(2):121–124. (Cited on page 40.)

DeNeefe, S. (2011). *Tree-Adjoining Machine Translation*. Ph.D. thesis, University of Southern California, Los Angeles, CA, USA. (Cited on pages 167 and 169.)

DeNeefe, S. and Knight, K. (2009). Synchronous tree adjoining machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP 2009)*, pages 727–736. The Association for Computational Linguistics. (Cited on page 169.)

DeNero, J., Bansal, M., Pauls, A., and Klein, D. (2009). Efficient parsing for transducer grammars. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL'09)*, pages 227–235. The Association for Computational Linguistics. (Cited on page 169.)

Dijkstra, E. W., Heise, W., Perlis, A. J., and Samelson, K. (1959). ALGOL sub-committee report - extensions. *Communications of the ACM*, 2(9):24. (Cited on page 164.)

Ding, Y. and Palmer, M. (2005). Machine translation using probabilistic synchronous dependency insertion grammars. In Knight, K., Ng, H. T., and Oflazer, K., editors, *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*. The Association for Computer Linguistics. (Cited on page 169.)

Doner, J. (1965). Decidability of the weak second-order theory of two successors. *Notices of the American Mathematical Society*, 12:819. (Cited on page 166.)

Doner, J. (1970). Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4(5):406–451. (Cited on pages 17, 55, 63, 71, 73 and 166.)

Downey, P. J. (1974). *Formal languages and recursion schemes*. Ph.D. thesis, Harvard University, Cambridge, MA, USA. (Cited on page 167.)

Dras, M. (1997). Representing paraphrases using synchronous TAGs. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL '98) and Eighth Conference of the European Chapter of the Association for Computational Linguistics (EACL '97)*, pages 516–518. Association for Computational Linguistics. (Cited on page 169.)

Dras, M. (1999). A meta-level grammar: Redefining synchronous TAG for translation and paraphrase. In Dale, R. and Church, K. W., editors, *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 80–87. ACL. (Cited on page 169.)

Duske, J. and Parchmann, R. (1984). Linear indexed languages. *Theoretical Computer Science*, 32(1-2):47–60. (Cited on page 75.)

Echihabi, A. and Marcu, D. (2003). A noisy-channel approach to question answering. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 16–23, Morristown. Association for Computational Linguistics. (Cited on page 167.)

Eilenberg, S. (1974). *Automata, Languages, and Machines*, volume A. Academic Press Inc., New York. (Cited on pages 35, 37, 38 and 164.)

Eilenberg, S. and Wright, J. B. (1967). Automata in general algebras. *Information and Control*, 11(4):452–470. (Cited on pages 146 and 166.)

Eisner, J. (2003). Learning non-isomorphic tree mappings for machine translation. In Funakoshi, K., Kübler, S., and Otterbacher, J., editors, *ACL 2003, 41st Annual Meeting of the Association for Computational Linguistics, Companion Volume to the Proceedings*, pages 205–208. Association of Computational Linguistics. (Cited on pages 37, 68, 92, 155, 167 and 169.)

Elgot, C. C. and Mezei, J. (1965). On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68. (Cited on pages 35, 37, 106 and 164.)

Engelfriet, J. (1971). Bottom-up and top-down tree transformations - a comparison. Memorandum 19, Technische Hogeschool Twente, Twente, Holland. (Cited on page 167.)

Engelfriet, J. (1975a). Bottom-up and top-down tree transformations - a comparison. *Mathematical Systems Theory*, 9(3):198–231. (Cited on pages 59, 80, 81, 85, 86, 89, 103, 105, 123, 137, 138, 166, 167 and 168.)

Engelfriet, J. (1977). Top-down tree transducers with regular look-ahead. *Mathematical Systems Theory*, 10(1):289–303. (Cited on pages 82, 137 and 167.)

Engelfriet, J. (1980). Some open questions and recent results on tree transducers and tree languages. In Book, R., editor, *Formal language theory: perspectives and open problems*, pages 241–286. Academic Press, New York, NY. (Cited on pages 147 and 168.)

Engelfriet, J. (1981). Tree transducers and syntax-directed semantics. Memorandum 363, Technische Hogeschool Twente, Twente, Holland. (Cited on page 168.)

Engelfriet, J. (1982). Three hierarchies of transducers. *Mathematical Systems Theory*, 15(2):95–125. (Cited on pages 91 and 167.)

Engelfriet, J. (2015). Tree automata and tree grammars. *CoRR*, abs/1510.02036. (Cited on pages 85, 89 and 167.)

Engelfriet, J. (April 1975c). Tree automata and tree grammars. Lecture Notes DAIMI FN-10, Aarhus University, Aarhus, Denmark. (Cited on pages 17, 19, 55, 56, 59, 63, 65, 66, 70, 73, 85, 89, 91, 103, 137, 166, 167 and 168.)

Engelfriet, J. (July 1975b). Top-down tree transducers with regular look-ahead. Technical Report DAIMI PB-49, Aarhus University, Aarhus, Denmark. (Cited on page 167.)

Engelfriet, J., Lilin, E., and Maletti, A. (2008). Extended multi bottom-up tree transducers. In Ito, M. and Toyama, M., editors, *Proceedings of Developments in Language Theory, 12th International Conference (DLT 2008)*, volume 5257 of *Lecture Notes in Computer Science*, pages 289–300. Springer. (Cited on page 168.)

Engelfriet, J., Lilin, E., and Maletti, A. (2009). Extended multi bottom-up tree transducers. *Acta Informatica*, 46(8):561–590. (Cited on page 168.)

Engelfriet, J., Maneth, S., and Seidl, H. (2016). Look-ahead removal for total deterministic top-down tree transducers. *Theoretical Computer Science*, 616:18–58. (Cited on page 167.)

Engelfriet, J. and Schmidt, E. M. (1977a). IO and OI. I. *Journal of Computer and System Sciences*, 15(3):328–353. (Cited on page 167.)

Engelfriet, J. and Schmidt, E. M. (1977b). IO and OI. II. *Journal of Computer and System Sciences*, 16(1):67–99. (Cited on page 167.)

Engelfriet, J. and Vogler, H. (1985). Macro tree transducers. *Journal of Computer and Systems Sciences*, 31(1):71–146. (Cited on pages 147, 167 and 168.)

Engelfriet, J. and Vogler, H. (1986). Pushdown machines for the macro tree transducer. *Theoretical Computer Science*, 42:251–368. (Cited on page 167.)

Engelfriet, J. and Vogler, H. (1987). Look-ahead on pushdowns. *Information and Computation*, 73(3):245–279. (Cited on page 167.)

Engelfriet, J. and Vogler, H. (1988). High level tree transducers and iterated pushdown tree transducers. *Acta Informatica*, 26(1/2):131–192. (Cited on page 167.)

Engelfriet, J. and Vogler, H. (1991). Modular tree transducers. *Theoretical Computer Science*, 78(2):267–303. (Cited on page 168.)

Estenfeld, K. (1982). A new characterization theorem of tree-transductions. *Journal of Information Processing and Cybernetics (EIK)*, 18(4/5):187–204. (Cited on pages 141 and 147.)

Evey, R. J. (1963). Application of pushdown-store machines. In *Proceedings of the AFIPS Fall Joint Computer Conference*, pages 215–227. American Federation of Information Processing Societies (AFIPS) Press, Montvale, NJ. (Cited on page 163.)

Feblowitz, D. and Kauchak, D. (2013). Sentence simplification as tree transduction. In Williams, S., Siddharthan, A., and Nenkova, A., editors, *Proceedings of the Second Workshop on Predicting and Improving Text Readability for Target Reader Populations*, pages 1–10. Association for Computational Linguistics. (Cited on page 169.)

Fischer, M. J. (1968). Grammars with macro-like productions. In *Proceedings of the 9th Annual Symposium on Switching and Automata Theory (SWAT-FOCS)*, pages 131–142. IEEE Computer Society. (Cited on page 167.)

Fischer, P. C. (1963). On computability by certain classes of restricted Turing machines. In *Proceedings of the 4th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT-FOCS)*, pages 23–32. IEEE Computer Society. (Cited on page 164.)

Frank, R. (2000). From regular to context-free to mildly context-sensitive tree rewriting systems: The path of child language acquisition. In Abeillé, A. and Rambow, O., editors, *Tree Adjoining Grammars: Formalisms, Linguistic Analysis and Processing*, pages 101–120. CSLI Publications, Stanford, CA. (Cited on pages 68 and 69.)

Fülöp, Z. (1981). On attributed tree transducers. *Acta Cybernetica*, 5(3):261–279. (Cited on page 168.)

Fülöp, Z. (1991). A complete description for a monoid of deterministic bottom-up tree transformation classes. *Theoretical Computer Science*, 88(2):253–268. (Cited on pages 89 and 167.)

Fülöp, Z. (2004). A short introduction to tree transducers. GRLMC Reports 30/04, Universitat Rovira i Virgili, Tarragona, Spain. (Cited on pages 80, 88, 105, 123, 137 and 168.)

Fülöp, Z. and Gazdag, Z. (2003). Shape preserving top-down tree transducers. *Theoretical Computer Science*, 1-3(304):315–339. (Cited on page 86.)

Fülöp, Z., Kühnemann, A., and Vogler, H. (2004). A bottom-up characterization of deterministic top-down tree transducers with regular look-ahead. *Information Processing Letters*, 91(2):57–67. (Cited on page 168.)

Fülöp, Z., Kühnemann, A., and Vogler, H. (2005). Linear deterministic multi bottom-up tree transducers. *Theoretical Computer Science*, 347(1-2):276–287. (Cited on pages 146 and 168.)

Fülöp, Z. and Maletti, A. (2013). Composition closure of ε-free linear extended top-down tree transducers. In Béal, M.-P. and Carton, O., editors, *Proceedings of Developments in Language Theory - 17th International Conference (DLT 2013)*, volume 7907 of *Lecture Notes in Computer Science*, pages 239–251. Springer. (Cited on page 168.)

Fülöp, Z., Maletti, A., and Vogler, H. (2010). Preservation of recognizability for synchronous tree substitution grammars. In Drewes, F. and Kuhlmann, M., editors, *Proceedings of the 2010 Workshop on Applications of Tree Automata in Natural Language Processing (ATANLP)*, pages 1–9. Association for Computational Linguistics. (Cited on pages 92, 96, 97, 155 and 169.)

Fülöp, Z. and Vágvölgyi, S. (1987). Results on compositions of deterministic root-to-frontier tree transformations. *Acta Cybernetica*, 8(1):49–61. (Cited on pages 89 and 167.)

Fülöp, Z. and Vágvölgyi, S. (1989a). Congruential tree languages are the same as recognizable tree languages - a proof for a theorem of D. Kozen. *Bulletin of the EATCS*, 39:175–184. (Cited on page 166.)

Fülöp, Z. and Vágvölgyi, S. (1989b). Iterated deterministic top-down look-ahead. In Csirik, J., Demetrovics, J., and Gécseg, F., editors, *Proceedings of the International Conference on Fundamentals of Computation Theory (FCT'89)*, volume 380 of *Lecture Notes in Computer Science*, pages 175–184. Springer. (Cited on page 167.)

Fülöp, Z. and Vágvölgyi, S. (1989c). Top-down tree transducers with deterministic top-down look-ahead. *Information Processing Letters*, 33(1):3–5. (Cited on page 167.)

Fülöp, Z. and Vágvölgyi, S. (1989d). Variants of top-down tree transducers with look-ahead. *Mathematical Systems Theory*, 21(3):125–145. (Cited on page 167.)

Fülöp, Z. and Vágvölgyi, S. (1991). A complete classification of deterministic root-to-frontier tree transformation classes. *Theoretical Computer Science*, 81(1):1–15. (Cited on page 167.)

Fülöp, Z. and Vogler, H. (1998). *Syntax-Directed Semantics. Formal Models Based on Tree Transducers*. Springer, Berlin. (Cited on pages 18, 79, 80, 89, 123, 147, 164 and 168.)

Galley, M., Graehl, J., Knight, K., Marcu, D., DeNeefe, S., Wang, W., and Thayer, I. (2006). Scalable inference and training of context-rich syntactic translation models. In Calzolari, N., Cardie, C., and Isabelle, P., editors, *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (ACL 2006)*. The Association for Computer Linguistics. (Cited on page 168.)

Galley, M., Hopkins, M., Knight, K., and Marcu, D. (2004). What's in a translation rule? In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL-04)*, pages 273–280. Association for Computational Linguistics. (Cited on page 168.)

Gazdag, Z. (2006a). Decidability of the shape preserving property of bottom-up tree transducers. *International Journal of Foundations in Computer Science*, 17(2):395–414. (Cited on page 86.)

Gazdag, Z. (2006b). *Shape preserving tree transducers*. Ph.D. thesis, University of Szeged, Szeged, Hungary. (Cited on pages 86, 128 and 130.)

Gazdar, G. (1988). Applicability of indexed grammars to natural languages. In Reyle, U. and Rohrer, C., editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. D. Reidel Publishing Company, Dordrecht. (Cited on page 75.)

Gécseg, F. and Steinby, M. (1984). *Tree Automata*. Akadémiai Kiadó, Budapest. (Cited on pages 17, 18, 19, 55, 57, 59, 63, 65, 66, 67, 68, 70, 71, 73, 77, 80, 81, 89, 91, 109, 118, 120, 142, 143, 166, 167 and 168.)

Gécseg, F. and Steinby, M. (1997). Tree languages. In Salomaa, A. and Rozenberg, G., editors, *Handbook of Formal Languages*, volume 3: Beyond Words, pages 1–68. Springer, Berlin. (Cited on pages 18, 55, 57, 59, 63, 65, 68, 70, 73, 74, 75, 77, 80, 109, 142, 166, 167 and 168.)

Gécseg, F. and Steinby, M. (2015). Tree automata. *CoRR*, abs/1509.06233. (Cited on pages 89, 91, 167 and 168.)

Georgeff, M. P. (1981). Interdependent translation schemes. *Journal of Computer and System Sciences*, 22(2):198–219. (Cited on page 37.)

Gildea, D. (2003). Loosely tree-based alignment for machine translation. In Hinrichs, E. W. and Roth, D., editors, *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, pages 80–87. Association for Computational Linguistics. (Cited on pages 167 and 169.)

Ginsburg, S. (1962). *Introduction to Mathematical Machine Theory*. McGraw-Hill Book Co., Reading, MA. (Cited on page 164.)

Ginsburg, S. (1966). *The Mathematical Theory of Context-Free Languages*. Addison-Wesley, New York. (Cited on pages 35, 37, 38, 39 and 164.)

Ginsburg, S. (1975). *Algebraic and Automata-theoretic Properties of Formal Languages*. North-Holland, Amsterdam. (Cited on pages 37 and 137.)

Ginsburg, S. and Spanier, E. H. (1966). Finite-turn pushdown automata. *SIAM Journal on Control*, 4(3):429–453. (Cited on page 164.)

Gorn, S. (1967). Explicit definitions and linguistic dominoes. In Hart, J. F. and Takasu, S., editors, *Proceedings of the Conference on Systems and Computer Science*, pages 77–115, Toronto, ON, Canada. University of Toronto Press. (Cited on page 57.)

Graehl, J. (1997). CARMEL. `http://www.isi.edu/licensed-sw/carmel/`. (Cited on page 39.)

Graehl, J. and Knight, K. (2004). Training tree transducers. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL-04)*, pages 105–112. Association for Computational Linguistics. (Cited on pages 86 and 167.)

Graehl, J., Knight, K., and May, J. (2008). Training tree transducers. *Computational Linguistics*, 34(3):391–427. (Cited on pages 19, 67, 86, 122, 168 and 169.)

Guessarian, I. (1983). Pushdown tree automata. *Mathematical Systems Theory*, 16(4):237–263. (Cited on page 75.)

Harrison, M. A. (1978). *Introduction to Formal Language Theory*. Addison Wesley, Reading, MA. (Cited on pages 23, 29, 32 and 164.)

Hayashi, T. (1973). On derivation trees of indexed grammars – An extension of the uvwxy - theorem. *Publications of the Research Institute for Mathematical Sciences*, 9:61–92. (Cited on page 75.)

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 2nd edition. (Cited on pages 23, 30, 31, 50, 70 and 164.)

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 3rd edition. (Cited on pages 29 and 164.)

Hopcroft, J. E. and Ullman, J. D. (1969). *Formal Languages and Their Relation to Automata*. Addison-Wesley, Reading, MA. (Cited on page 164.)

Huang, L. (2008). *Forest-based Algorithms in Natural Laguage Processing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA. (Cited on pages 35, 40 and 165.)

Huang, L., Knight, K., and Joshi, A. (2006). Statistical syntax-directed translation with extended domain of locality. In *Proceedings of the 7th Biennial Conference of the Association for Machine Translation in the Americas (AMTA 2006)*. Association for Computational Linguistics. (Cited on pages 19, 141, 142 and 169.)

Huang, L., Zhang, H., Gildea, D., and Knight, K. (2009). Binarization of synchronous context-free grammars. *Computational Linguistics*, 35(4):559–595. (Cited on pages 43, 44, 48, 52 and 165.)

Huffman, D. A. (1954). The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3-4):161–190 and 275–303. (Cited on page 163.)

Irons, E. T. (1961). A syntax directed compiler for ALGOL 60. *Communications of the ACM*, 4(1):51–55. (Cited on pages 17, 37, 40, 149 and 164.)

Irons, E. T. (1963). The structure and use of the syntax directed compiler. *Annual Review in Automatic Programming*, 3:207–227. (Cited on page 164.)

Joshi, A. K., Levy, L. S., and Takahashi, M. (1972). A tree generating system. In Nivat, M., editor, *Proceedings of the Automata, Languages and Programming, Colloquium (ICALP 1972)*, pages 453–465. North-Holland, Amsterdam. (Cited on page 167.)

Joshi, A. K., Levy, L. S., and Takahashi, M. (1975). Tree adjunct grammars. *Journal Computer and System Sciences*, 10(1):136–163. (Cited on page 167.)

Joshi, A. K. and Schabes, Y. (1997). Tree-adjoining grammars. In Salomaa, A. and Rozenberg, G., editors, *Handbook of Formal Languages*, volume 3: Beyond Words, pages 1–68. Springer, Berlin. (Cited on page 169.)

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice-Hall, Upper Saddle River, NJ, 2nd edition. (Cited on pages 35, 39, 164 and 165.)

Kaplan, R. M. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378. (Cited on page 39.)

Karp, D., Schabes, Y., Zaidel, M., and Egedi, D. (1992). A freely available wide coverage morphological analyzer for English. In *Proceedings of the 14th Conference on Computational Linguistics (COLING '92)*, volume vol. 3, pages 950–955. Association for Computational Linguistics. (Cited on page 38.)

Karttunen, L., Gaál, T., and Kempe, A. (June 1997). Xerox finite-state tool. Technical report, Xerox Research Centre Europe, Grenoble, France. (Cited on page 39.)

Karttunen, L. and Wittenburg, K. (1983). A two-level morphological analysis of English. *Texas Linguistic Forum*, 22:217–228. (Cited on page 39.)

Kato, Y. and Matsubara, S. (2010a). Correcting errors in a treebank based on synchronous tree substitution grammar. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010), Short Papers*, pages 74–79. (Cited on page 169.)

Kato, Y. and Matsubara, S. (2010b). Correcting syntactic annotation errors using a synchronous tree substitution grammar. *IEICE Transactions*, 93-D(9):2660–2663. (Cited on page 169.)

Knight, K. (2007). Capturing practical natural language transformations. *Machine Learning*, 21(2):121–133. (Cited on pages 18, 19, 88, 91, 92, 98, 103, 132 and 168.)

Knight, K. and Graehl, J. (2005). An overview of probabilistic tree transducers for natural language processing. In Gelbukh, A. F., editor, *Proceedings 6th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing)*, volume 3406 of *Lecture Notes in Computer Science*, pages 1–24, Berlin. Springer. (Cited on pages 18, 19, 86, 88, 167 and 168.)

Knight, K. and Marcu, D. (2002). Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1):91–107. (Cited on page 167.)

Knuth, D. E. (1968). *The Art of Computer Programming, Volume I: Fundamental Algorithms*. Addison-Wesley. (Cited on page 141.)

Kopřiva, J. (1970). Decomposition translations and syntax directed translation schemata. *Kybernetika*, 6(6):403–417. (Cited on page 167.)

Kosaraju, S. R. (1973). Context-sensitiveness of translational languages. In *Proceedings of the Seventh Annual Princeton Conference on Information Sciences and Systems*. Princeton University Press. (Cited on page 167.)

Koskenniemi, K. (1983). Two-level morphology. A general computational model for word-form recognition and production. Publications 12, University of Helsinki, Deptartment of General Linguistics, Helsinki, Finland. (Cited on pages 38 and 39.)

Kozen, D. (1977). Complexity of finitely presented algebras. In Hopcroft, J. E., Friedman, E. P., and Harrison, M. A., editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC)*, pages 164–177. ACM. (Cited on page 166.)

Kozen, D. C. (1997). *Automata and Computability*. Springer, New York. (Cited on page 164.)

Krasnowska, K., Kieras, W., Wolinski, M., and Przepiórkowski, A. (2012). Using tree transducers for detecting errors in a treebank of Polish. In Sojka, P., Horák, A., Kopecek, I., and Pala, K., editors, *Proceedings of the Text, Speech and Dialogue - 15th International Conference (TSD 2012)*, volume 7499 of *Lecture Notes in Computer Science*, pages 119–126. Springer. (Cited on page 169.)

Kuroda, S.-Y. (1964). Classes of languages and linear-bounded automata. *Information and Control*, 7(2):207–223. (Cited on page 164.)

Lagoutte, A., Braune, F., Quernheim, D., and Maletti, A. (2012). Composing extended top-down tree transducers. In Daelemans, W., Lapata, M., and Màrquez, L., editors, *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2012)*, pages 808–817. The Association for Computational Linguistics. (Cited on page 168.)

Landweber, P. S. (1963). Three theorems on phrase structure grammars of Type 1. *Information and Control*, 6(2):131–136. (Cited on page 164.)

Langkilde, I. and Knight, K. (1998). Generation that exploits corpus-based statistical knowledge. In Boitet, C. and Whitelock, P., editors, *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL '98)*, pages 704–710. Morgan Kaufmann Publishers / ACL. (Cited on page 167.)

Lewis II, P. M. and Stearns, R. E. (1966). Syntax directed transduction. In *Conference Record 7th Annual Symposium on Switching and Automata Theory (FOCS)*, pages 7–20. IEEE Computer Society. (Cited on page 164.)

Lewis II, P. M. and Stearns, R. E. (1968). Syntax-directed transduction. *Journal of the ACM*, 15(3):465–488. (Cited on pages 17, 40, 45, 149 and 164.)

Lilin, E. (1978). *Une généralisation des transducteurs d'états finis d'arbres: les S-transducteurs.* Phd thesis, Université de Lille, Lille, France. (Cited on pages 86, 146, 167 and 168.)

Linz, P. (2001). *An Introduction to Formal Languages and Automata.* Jones and Bartlett Publishers, Inc., Sudbury, MA, 3 edition. (Cited on pages 23, 26, 29, 31 and 164.)

Liu, Y., Lü, Y., and Liu, Q. (2009). Improving tree-to-tree translation with packed forests. In Su, K.-Y., Su, J., and Wiebe, J., editors, *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL/IJCNLP)*, pages 558–566. The Association for Computer Linguistics. (Cited on page 169.)

Locke, W. N. and Booth, A. D. (1955). *Machine Translation of Languages.* Technology Press of the MIT, Cambridge, Mass., and John Wiley and Sons, Inc., Wiley. (Cited on page 164.)

Magidor, M. and Moran, G. (1969). Finite automata over finite trees. Technical Report 30, Hebrew University, Jerusalem, Israel. (Cited on pages 63, 65 and 166.)

Maibaum, T. S. E. (1974). A generalized approach to formal languages. *Journal of Computer and System Sciences*, 8(3):409–439. (Cited on page 167.)

Maletti, A. (2007). Compositions of extended top-down tree transducers. In Loos, R., Fazekas, S. Z., and Martín-Vide, C., editors, *Proceedings of the 1st International Conference on Language and Automata Theory and Applications (LATA 2007)*, volume 35/07 of *GRLMC Reports*, pages 379–390. Universitat Rovira i Virgili, Tarragona, Spain. (Cited on pages 19, 131, 142, 167 and 168.)

Maletti, A. (2008). Compositions of extended top-down tree transducers. *Information and Computation*, 206(9–10):1187–1196. (Cited on pages 19, 20, 78, 86, 92, 131, 132, 139, 142, 167 and 168.)

Maletti, A. (2010a). Survey: tree transducers in machine translation. In Bordihn, H., Freund, R., Hinze, T., Holzer, M., Kutrib, M., and Otto, F., editors, *Proceedings of the 2nd International Workshop on Non-Classical Models of Automata and Applications (NCMA 2010)*, volume 263 of `books@ocg.at`, pages 11–32. Österreichische Computer Gesellschaft. (Cited on pages 18, 19, 77, 86, 88, 91, 103, 131, 132, 137, 140, 142 and 168.)

Maletti, A. (2010b). A tree transducer model for synchronous tree-adjoining grammars. In Hajic, J., Carberry, S., and Clark, S., editors, *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010)*, pages 1067–1076. The Association for Computer Linguistics. (Cited on pages 168 and 169.)

Maletti, A. (2010c). Why synchronous tree substitution grammars? In *Proceedings of the Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL 2010)*, pages 876–884. The Association for Computational Linguistics. (Cited on pages 92, 96, 168 and 169.)

Maletti, A. (2011a). An alternative to synchronous tree substitution grammars. *Natural Language Engineering*, 17(2):221–242. (Cited on pages 92, 96, 97, 168 and 169.)

Maletti, A. (2011b). How to train your multi bottom-up tree transducer. In Lin, D., Matsumoto, Y., and Mihalcea, R., editors, *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL HLT 2011)*, pages 825–834. The Association for Computer Linguistics. (Cited on page 168.)

Maletti, A. (2011c). Tree transformations and dependencies. In Kanazawa, M., Kornai, A., Kracht, M., and Seki, H., editors, *The Mathematics of Language - 12th Biennial Conference, MOL 12, Proceedings*, volume 6878 of *Lecture Notes in Computer Science*, pages 1–20. Springer. (Cited on page 169.)

Maletti, A. (2012). Every sensible extended top-down tree transducer is a multi bottom-up tree transducer. In *Proceedings of the Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*, pages 263–273. The Association for Computational Linguistics. (Cited on page 168.)

Maletti, A. (2013). Synchronous forest substitution grammars. In Muntean, T., Poulakis, D., and Rolland, R., editors, *Proceedings of the 5th International Conference on Algebraic Informatics (CAI)*, volume 8080 of *Lecture Notes in Computer Science*, pages 235–246. Springer. (Cited on pages 37, 141, 147 and 169.)

Maletti, A. (2014). Applications of automata theory in parsing and machine translation. Lecture I: Tree automata. Tutorial given at *CSEDays. Theory 2014*, Ural Federal University, Ekaterinburg, Russia, August 23 - August 25 , `http://www.ims.uni-stuttgart.de/institut/mitarbeiter/maletti/pub/slides/2014-08-23.pdf`. (Cited on page 69.)

Maletti, A., Graehl, J., Hopkins, M., and Knight, K. (2009). The power of extended top-down tree transducers. *SIAM Journal of Computing*, 39(2):410–430. (Cited on pages 18, 19, 86, 87, 88, 89, 91, 167 and 168.)

Maletti, A. and Tîrnăucă, C. I. (2009). Syntax-directed translations and quasi-alphabetic tree bimorphisms - revisited. In Bozapalidis, S. and Rahonis, G., editors, *Proceedings of the Third International Conference on Algebraic Informatics (CAI 2009)*, volume 5725 of *Lecture Notes in Computer Science*, pages 305–317. Springer. (Cited on pages 19, 47, 103, 106, 110, 120, 128, 129 and 165.)

Maletti, A. and Tîrnăucă, C. I. (2010). Properties of quasi-relabeling tree bimorphisms. *International Journal of Foundations in Computer Science*, 21(3):257–276. (Cited on pages 19, 52, 103, 106, 108, 135, 138 and 165.)

Maletti, A. and Vogler, H. (2009). Compositions of top-down tree transducers with *epsilon*-rules. In Yli-Jyrä, A., Kornai, A., Sakarovitch, J., and Watson, B. W., editors, *Proceedings of the 8th International Workshop on Finite-State Methods and Natural Language Processing (FSMNLP)*, volume 6062 of *Lecture Notes in Computer Science*, pages 69–80. Springer. (Cited on page 168.)

Maneth, S. (2004). *Models of Tree Translation*. Ph.D. thesis, Universiteit Leiden, Leiden, The Netherlands. (Cited on pages 17, 147, 164 and 168.)

Marcu, D., Wang, W., Echihabi, A., and Knight, K. (2006). SPMT: Statistical machine translation with syntactified target language phrases. In Jurafsky, D. and Gaussier, É., ed-

itors, *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 44–52. The Association for Computational Linguistics. (Cited on page 168.)

Martin, D. F. and Vere, S. A. (1970). On syntax-directed transduction and tree transducers. In Fischer, P. C., Fabian, R., Ullman, J. D., and Karp, R. M., editors, *Proceedings of the 2nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 129–135. ACM. (Cited on pages 142, 167 and 168.)

Martín-Vide, C., Mitrana, V., and Păun, G., editors (2004). *Formal Languages and Applications*, volume 148 of *Studies in Fuzzyness and Soft Computing*. Springer, Berlin. (Cited on page 164.)

Mateescu, A. and Salomaa, A. (1997). Aspects of classical language theory. In Salomaa, A. and Rozenberg, G., editors, *Handbook of Formal Languages*, volume 1: Word, Language, Grammar, pages 175–252. Springer, Berlin. (Cited on page 154.)

May, J. (2010). *Weighted Tree Automata and Transducers for Syntactic Natural Language Processing*. Ph.D. thesis, University of Southern California, Los Angeles, CA, USA. (Cited on pages 18, 67, 103, 122 and 167.)

May, J. and Knight, K. (2006). Tiburon: A weighted tree automata toolkit. In Ibarra, O. H. and Yen, H.-C., editors, *Proceedings of the 11th International Conference on Implementation and Application of Automata (CIAA)*, volume 4094 of *Lecture Notes in Computer Science*, pages 102–113. Springer. (Cited on pages 103 and 122.)

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133. (Cited on page 163.)

Mealy, G. H. (1955). A method for synthesizing sequential circuits. *Bell Systems Technical Journal*, 34(5):1045–1079. (Cited on page 163.)

Melamed, D. I. (2003). Multitext grammars and synchronous parsers. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2003)*, pages 79–86. Association for Computational Linguistics. (Cited on pages 37, 116, 165 and 167.)

Melamed, D. I., Satta, G., and Wellington, B. (2004). Generalized multitext grammars. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (ACL 2004)*, pages 563–569. Association for Computational Linguistics. (Cited on pages 37 and 169.)

Mezei, J. and Wright, J. B. (1967). Algebraic automata and context-free sets. *Information and Control*, 11(1/2):3–29. (Cited on pages 73, 144 and 166.)

Mohri, M. (1996). On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2:61–80. (Cited on pages 35, 37, 38, 39 and 164.)

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311. (Cited on pages 35, 37, 38, 39 and 164.)

Moore, E. F. (1956). Gedanken experiments on sequential machines. In Shannon, C. E. and McCarthy, J., editors, *Automata Studies*, pages 129–153. Princeton University Press, New York. (Cited on page 163.)

Myhill, J. (1960). Linear bounded automata. Technical Note WADD TR-60-165, Wright Patterson Air Force Base, Dayton, OH, USA. (Cited on page 164.)

Nederhof, M.-J. and Vogler, H. (2012). Synchronous context-free tree grammars. In Satta, G., Han, C.-H., Éric Villemonte de la Clergerie, Seddah, D., and Danlos, L., editors, *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, pages 55–63. (Cited on pages 19, 37, 141, 147, 159, 168 and 169.)

Nesson, R., Satta, G., and Shieber, S. M. (2008). Optimal k-arization of synchronous tree-adjoining grammar. In McKeown, K., Moore, J. D., Teufel, S., Allan, J., and Furui, S., editors, *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-08: HLT)*, pages 604–612. The Association for Computer Linguistics. (Cited on page 169.)

Nesson, R., Shieber, S. M., and Rush, A. (2006). Induction of probabilistic synchronous tree-insertion grammars for machine translation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA 2006)*, pages 128–137. The Association for Machine Translation in the Americas. (Cited on page 169.)

Nivat, M. (1968). Transductions des langages de chomsky. *Annales de l'Institut Fourier*, 18(1):339–455. (Cited on pages 38, 39, 49, 101, 103, 110, 141, 142 and 164.)

Oettinger, A. G. (1961). Automatic syntactic analysis and the pushdown store. In *Structure of Language and Its Mathematical Aspects*, volume 12 of *Proc. 12th Symposium on Applied Mathematics*, pages 104–129. American Mathematical Society, Providence, R.I. (Cited on page 163.)

Ogden, W. F. and Rounds, W. C. (1972). Compositions of n tree transducers. In Fischer, P. C., Zeiger, H. P., Ullman, J. D., and Rosenberg, A. L., editors, *Proceedings of the 4th Annual ACM Symposium on Theory of Computing (STOC)*, pages 198–206. ACM. (Cited on page 167.)

O'Hearn, P. and Tennent, R., editors (1997). *Algol-like Languages*. Progress in Theoretical Computer Science. Birkhäuser, Boston. (Cited on page 164.)

Pair, C. and Quéré, A. (1968). Définition et etude des bilangages réguliers. *Information and Control*, 13(6):565–593. (Cited on page 146.)

Pang, B., Knight, K., and Marcu, D. (2003). Syntax-based alignment of multiple translations extracting paraphrases and generating new sentences. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 102–109. Association for Computational Linguistics. (Cited on page 167.)

Park, H. S. (1995). Mapping scrambled korean senteces into english using synchronous tags. In Uszkoreit, H., editor, *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 317–319. Morgan Kaufmann Publishers / ACL. (Cited on page 169.)

Partee, B. H., ter Meulen, A., and Wall, R. E. (1990). *Mathematical Methods in Linguistics*, volume 30 of *Studies in Linguistics and Philosophy*. Kluwer Academic, Dordrecht. (Cited on page 75.)

Paull, M. (1967). Bilateral descriptions of syntactic mappings. In *Proceedings of the 1st Annual Princeton Conference on Information Sciences and Systems*, pages 76–81. Princeton University Press, Princeton, NJ. (Cited on page 164.)

Priestley, M. (2011). *A Science of Operations: Machines, Logic and the Invention of Programming*. Springer, London. (Cited on page 164.)

Rabin, M. O. (1969). Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141(2):1–35. (Cited on page 166.)

Rabin, M. O. and Scott, D. (1959). Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):115–125. (Cited on page 163.)

Rahonis, G. (2001). Alphabetic and synchronized tree transducers. *Theoretical Computer Science*, 255(1-2):377–399. (Cited on pages 107, 108, 126, 130, 135, 136, 137, 141, 143, 160 and 168.)

Rambow, O. and Satta, G. (1996). Synchronous models of language. In Joshi, A. K. and Palmer, M., editors, *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 116–123. Morgan Kaufmann Publishers / ACL. (Cited on page 169.)

Rambow, O., Vijay-Shanker, K., and Weir, D. J. (1995). D-Tree grammars. In Uszkoreit, H., editor, *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 151–158. ACL. (Cited on page 169.)

Raoult, J.-C. (1992). A survey of tree transductions. In Nivat, M. and Podelski, A., editors, *Tree Automata and Languages*, pages 311–326. North-Holland, Amsterdam. (Cited on pages 19, 101, 103, 140, 141, 142, 147 and 169.)

Razmara, M. (March 2011). Application of tree transducers in statistical machine translation. Depth report, Simon Fraser University, British Columbia, Canada. (Cited on pages 77, 88, 92 and 168.)

Roche, E. and Schabes, Y. (1995). Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, 21(2):227–253. (Cited on page 35.)

Roche, E. and Schabes, Y., editors (1997). *Finite-State Language Processing*. The MIT Press, Cambridge, MA. (Cited on pages 35, 39 and 164.)

Rosen, B. K. (1971). *Subtree replacement systems*. Ph.D. thesis, Harvard University, Cambridge, MA, USA. (Cited on page 167.)

Rounds, W. C. (1968). *Trees, transducers and transformations*. Ph.D. thesis, Stanford University, Standford, CA, USA. (Cited on page 167.)

Rounds, W. C. (1969). Context-free grammars on trees. In Fischer, P. C., Ginsburg, S., and Harrison, M. A., editors, *Proceedings of the 1st Annual ACM Symposium on Theory of Computing (STOC'69)*, pages 143–148. ACM. (Cited on pages 74, 75 and 167.)

Rounds, W. C. (1970a). Mappings and grammars on trees. *Mathematical Systems Theory*, 4(3):257–287. (Cited on pages 17, 80, 86 and 167.)

Rounds, W. C. (1970b). Tree-oriented proofs of some theorems on context-free and indexed languages. In Fischer, P. C., Fabian, R., Ullman, J. D., and Karp, R. M., editors, *Proceedings of the 2nd Annual ACM Symposium on Theory of Computing (STOC'70)*, pages 109–116, New York, NY, USA. ACM. (Cited on pages 74, 75, 166 and 167.)

Rounds, W. C. (1973). Complexity of recognition in intermediate-level languages. In *Proceedings of the 14th Annual Symposium on Switching and Automata Theory (SWAT-FOCS)*, pages 145–158. IEEE Computer Society. (Cited on page 167.)

Rozenberg, G. and Salomaa, A., editors (1997). *Handbook of Formal Languages*. Springer, Berlin. (Cited on pages 23, 25, 29 and 164.)

Saers, M. (2011). *Translation as linear transduction: Models and algorithms for efficient learning in statistical machine translation*. Ph.D. thesis, Upssala University, Uppsala, Sweden. (Cited on pages 37, 42, 48 and 165.)

Saers, M., Addanki, K., and Wu, D. (2012). From finite-state to inversion transductions: toward unsupervised bilingual grammar induction. In Kay, M. and Boitet, C., editors, *COLING 2012, 24th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, 8-15 December 2012, Mumbai, India*, pages 2325–2340. Indian Institute of Technology Bombay. (Cited on pages 142 and 165.)

Saers, M., Nivre, J., and Wu, D. (2009). Learning stochastic bracketing inversion transduction grammars with a cubic time biparsing algorithm. In *Proc. of the 11th International Workshop on Parsing Technologies (IWPT'09)*, pages 29–32. The Association for Computational Linguistics. (Cited on pages 142 and 165.)

Saers, M., Nivre, J., and Wu, D. (2010). Word alignment with stochastic bracketing linear inversion transduction grammar. In *Proc. of the Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*, pages 341–344. The Association for Computational Linguistics. (Cited on page 165.)

Saers, M. and Wu, D. (2009). Improving phrase-based translation via word alignments from stochastic inversion transduction grammars. In *Proc. Third Workshop on Syntax and Structure in Statistical Translation*, pages 28–36, Stroudsburg. Association for Computational Linguistics. (Cited on pages 142 and 165.)

Saers, M. and Wu, D. (2011). Linear transduction grammars and zipper finite-state transducers. In Angelova, G., Bontcheva, K., Mitkov, R., and Nicolov, N., editors, *Proc. of the Recent Advances in Natural Language Processing (RANLP'11)*, pages 640–647. RANLP 2011 Organising Committee. (Cited on page 165.)

Salomaa, A. (1973). *Formal Languages*. Academic Press, New York. (Cited on pages 23, 29 and 164.)

Salomaa, A. and Soittola, M. (1978). *Automata-theoretic Aspects of Formal Power Series*. Texts and monographs in computer science. Springer. (Cited on page 164.)

Salomaa, K. (1994). Synchronized tree automata. *Theoretical Computer Science*, 127(1):25–51. (Cited on page 143.)

Sammet, J. E. (1969). *Programming Languages: History and Fundamentals*. Prentice-Hall, Inc., Upper Saddle River, NJ. (Cited on page 164.)

Satta, G. (2004). Local parallel rewriting: Theory and practice. Tutorial given at *FG04* (9th Conference on Formal Grammar), `http://www.dei.unipd.it/~satta/`. (Cited on pages 18, 91 and 168.)

Satta, G. (2007). Translation algorithms by means of language intersection. Personal communication, `http://www.dei.unipd.it/~gsatta/publ/paper/inters.pdf`. (Cited on pages 42, 43, 44, 52 and 165.)

Satta, G. (2009). Synchronous rewriting for natural language processing. Tutorial given at *MOL-11* (11th Meeting on Mathematics of Language). (Cited on pages 18, 91 and 168.)

Satta, G. and Peserico, E. (2005). Some computational complexity results for synchronous context-free grammars. In *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP 2005)*, pages 803–810. Association for Computational Linguistics. (Cited on pages 37, 42, 44, 52, 149 and 165.)

Schabes, Y. (1990). *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA. (Cited on pages 68, 69, 91, 92, 166 and 169.)

Schabes, Y. and Joshi, A. K. (1990). Two recent developments in tree adjoining grammars: Semantics and efficient processing. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*, pages 48–53. Morgan Kaufmann Publishers, Inc. (Cited on page 169.)

Schimpf, K. M. and Gallier, J. H. (1985). Tree pushdown automata. *Journal of Computer and System Sciences*, 30(1):25–40. (Cited on page 74.)

Schnorr, C.-P. (1969). Transformational classes of grammars. *Information and Control*, 14(3):252–277. (Cited on page 147.)

Schreiber, P. P. (1975). Tree-transducers and syntax-connected transductions. In Barkhage, H., editor, *Automata Theory and Formal Languages, 2nd GI Conference*, volume 33 of *Lecture Notes in Computer Science*, pages 202–208. Springer. (Cited on pages 21, 37, 149, 151, 155, 165 and 167.)

Schreiber, P. P. (1976). Tree-transducers and syntax-connected transductions. In Arnold, A., Dauchet, M., and Jacob, G., editors, *1er Colloque sur les Arbres en Algèbre et en Programmation*, pages 217–238. Université de Lille 1. (Cited on pages 21, 37, 149, 151, 155, 165 and 167.)

Schuler, W. (1999). Preserving semantic dependencies in synchronous tree adjoining grammar. In Dale, R. and Church, K. W., editors, *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 88–95. ACL. (Cited on page 169.)

Schützenberger, M.-P. (1963). On context-free languages and pushdown automata. *Information and Control*, 6(3):246–264. (Cited on page 163.)

Schützenberger, M.-P. (1977). Sur une variante des fonctions séquentielles. *Theoretical Computer Science*, 4(1):47–57. (Cited on page 38.)

Searls, D. B. (1992). The linguistics of DNA. *American Scientist*, 80(6):579–591. (Cited on page 31.)

Sheridan, P. B. (1959). The arithmetic translator-compiler of the IBM FORTRAN automatic coding system. *Communications of the ACM*, 2(2):9–21. (Cited on page 164.)

Shieber, S. M. (1994). Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10(4):371–385. (Cited on page 169.)

Shieber, S. M. (2004). Synchronous grammars as tree transducers. In *Proceedings of TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms*, pages 88–95. (Cited on pages 18, 19, 37, 68, 87, 88, 92, 96, 131, 132, 142, 155, 167 and 169.)

Shieber, S. M. (2006). Unifying synchronous tree-adjoining grammars and tree transducers via bimorphisms. In McCarthy, D. and Wintner, S., editors, *Proceedings of the 11st Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006)*, pages 377–384. The Association for Computational Linguistics. (Cited on pages 19, 37, 141, 147, 167, 168 and 169.)

Shieber, S. M. (2007). Probabilistic synchronous tree-adjoining grammars for machine translation: The argument from bilingual dictionaries. In Chiang, D. and Wu, D., editors, *Proceedings of SSST: NAACL-HLT 2007 / AMTA Workshop on Syntax and Structure in Statistical Translation*, pages 377–384. The Association for Computational Linguistics. (Cited on page 169.)

Shieber, S. M. and Schabes, Y. (1990a). Generation and synchronous tree-adjoining grammars. In *Proceedings of the Fifth International Workshop on Natural Language Generation*, pages 9–14. (Cited on pages 37, 92, 147, 167 and 169.)

Shieber, S. M. and Schabes, Y. (1990b). Synchronous tree-adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING 1990)*, pages 253–258. University of Helsinki, Finland. (Cited on pages 37, 92, 147, 167 and 169.)

Shieber, S. M. and Schabes, Y. (1991). Generation and synchronous tree-adjoining grammars. *Computational Intelligence*, 4(7):220–228. (Cited on pages 92, 147, 167 and 169.)

Sipser, M. (1997). *Introduction to the Theory of Computation*. PWS Publishing Company, Boston, MA. (Cited on pages 23 and 164.)

Siromoney, R. and Krithivasan, K. (1974). Parallel context-free languages. *Information and Control*, 24(2):155–162. (Cited on page 153.)

Stamer, H. (2009). *Restarting Tree Automata. Formal Properties and Possible Variations*. Ph.D. thesis, University of Kassel, Kassel, Germany. (Cited on page 74.)

Steinby, M. (1977). On algebras as tree automata. In *Contributions to Universal Algebra - Proceedings of the Colloquim on Universal Algebra held in Szeged, 1975*, volume 17 of *Colloquia Mathematica Societatis János Bolyai*, pages 441–455. North-Holland, Amsterdam. (Cited on pages 143 and 166.)

Steinby, M. (1981). Some algebraic aspects of recognizability and rationality. In Gécseg, F., editor, *Proceedings of Fundamentals of Computation Theory (FCT'81)*, volume 117 of *Lecture Notes in Computer Science*, pages 360–372. Springer. (Cited on page 143.)

Steinby, M. (1984). Some decidable properties of Σ-rational and Σ-algebraic tree transformations. *Annales Universitatis Turkuensis. Series A I*, 186:102–109. (Cited on pages 106, 141, 143, 144, 145, 146 and 160.)

Steinby, M. (1986). On certain algebraically defined tree transformations. In Demetrovics, J., Budach, L., and Salomaa, A., editors, *Algebra, Combinatorics and Logic in Computer Science, Györ, Hungary, 1983*, volume 42 of *Colloquia Mathematica Societatis János Bolyai*, pages 745–764. North-Holland, Amsterdam. (Cited on pages 19, 59, 141, 143, 144, 145, 146, 160, 164 and 166.)

Steinby, M. (1990). A formal theory of errors in tree representations of patterns. *Journal of Information Processing and Cybernetics (EIK)*, 26(1/2):19–32. (Cited on pages 141, 143, 144 and 160.)

Steinby, M. (2004). Finite tree automata and regular tree automata. An introduction. In Martín-Vide, C., Mitrana, V., and Păun, G., editors, *Formal Languages and Applications*, volume 148 of *Studies in Fuzzyness and Soft Computing*, pages 411–426. Springer, Berlin. (Cited on pages 55, 143, 166 and 167.)

Steinby, M. (2005). Tree automata and tree languages. An introduction. Lecture notes given at Information Science Institute, Marina del Rey, CA, USA. (Cited on pages 55, 80, 143, 166 and 168.)

Steinby, M. and Tîrnăucă, C. I. (2007). Syntax-directed translations and quasi-alphabetic tree bimorphisms. In Holub, J. and Žďárek, J., editors, *Proceedings of the 12th International Conference on Implementation and Application of Automata (CIAA)*, volume 4783 of *Lecture Notes in Computer Science*, pages 265–276. Springer. (Cited on pages 19, 20, 59, 101, 103, 106, 110, 111, 160, 165 and 166.)

Steinby, M. and Tîrnăucă, C. I. (2009). Defining syntax-directed translations by tree bimorphisms. *Theoretical Computer Science*, 410(37):3495–3503. (Cited on pages 18, 19, 20, 59, 101, 103, 104, 106, 110, 111, 112, 160, 165 and 166.)

Sudkamp, T. A. (1997). *Languages and Machines: An Introduction to the Theory of Computer Science*. Addison Wesley, Reading, MA, 2nd edition. (Cited on pages 23, 69 and 164.)

Sudkamp, T. A. (2006). *Languages and Machines: An Introduction to the Theory of Computer Science*. Addison Wesley, Reading, MA, 3rd edition. (Cited on page 164.)

Sun, J., Zhang, M., and Tan, C. L. (2009). A non-contiguous tree sequence alignment-based model for statistical machine translation. In Su, K.-Y., Su, J., and Wiebe, J., editors,

Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL/IJCNLP), pages 914–922. The Association for Computer Linguistics. (Cited on pages 147 and 169.)

TAG+3, editor (1994). *Proceedings of the 3éme Colloque International sur les Grammaires d'Arbres Adjoints (TAG+3)*, volume TALANA-RT-94-01 of *Rapport Technique TALANA*. Université Paris 7. (Cited on page 169.)

Takahashi, M. (1972). Primitive transformations of regular sets and recognizable sets. In Nivat, M., editor, *Proceedings of the Automata, Languages and Programming, Colloquium (ICALP 1972)*, pages 475–480. North-Holland, Amsterdam. (Cited on pages 19, 20, 101, 103, 108, 128, 145, 160 and 166.)

Takahashi, M. (1977). Rational relations of binary trees. In Salomaa, A. and Steinby, M., editors, *Proceedings of the Automata, Languages and Programming, Fourth Colloquium (ICALP 1977)*, volume 52 of *Lecture Notes in Computer Science*, pages 524–538. Springer. (Cited on pages 13, 19, 140, 141, 142 and 166.)

Taylor, R. G. (1998). *Models of Computation and Formal Languages*. Oxford University Press, New York. (Cited on pages 23 and 164.)

Thatcher, J. W. (1967). Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences*, 1(4):317–322. (Cited on pages 17, 71, 73 and 166.)

Thatcher, J. W. (1970). Generalized^2 sequential machine maps. *Journal of Computer and System Sciences*, 4(4):339–367. (Cited on pages 80, 166 and 167.)

Thatcher, J. W. (1973). Tree automata: An informal survey. In Aho, A. V., editor, *Currents in the Theory of Computing*, pages 143–172. Prentice Hall. (Cited on pages 17, 59, 83, 166, 167 and 168.)

Thatcher, J. W. and Wright, J. B. (1965). Generalized finite automaton theory with an application to a decision problem of second-order logic. *Notices of the American Mathematical Society*, 12:820. (Cited on page 166.)

Thatcher, J. W. and Wright, J. B. (1968). Generalized finite automaton theory with an application to a decision problem of second-order logic. *Journal of Computer and System Sciences*, 2(1):57–81. (Cited on pages 55, 63 and 166.)

Tîrnăucă, C. I. (2008). Tree bimorphisms and their relevance in the theory of translations. In Sandoval, A. M., editor, *Actas del VIII Congreso de Lingüística General*, pages 1893–1912. (Cited on pages 18, 19, 42, 101, 103, 117, 140 and 169.)

Tîrnăucă, C. I. (2011). Synchronous context-free grammars and their tree transformations. *Fundamenta Informaticae*. accepted. (Cited on pages 19, 44, 103, 111, 113, 116, 118, 142, 164 and 165.)

Tîrnăucă, C. I. (2007). Synchronous context-free grammar translations by means of tree bimorphisms. In Bel Enguix, G. and Jiménez Lopez, M. D., editors, *Proceedings of the*

1st International Workshop on Non-Classical Formal Languages in Linguistics,, volume 36/07 of *GRLMC Reports*, pages 97–107. Universitat Rovira i Virgili, Tarragona, Spain. (Cited on pages 19, 113 and 116.)

Tîrnăucă, C. I. (2009). Model syntax-directed translations by tree transducers. In Bordihn, H., Freund, R., Holzer, M., Kutrib, M., and Otto, F., editors, *Proceedings of the 1st Workshop on Non-Classical Models for Automata and Applications (NCMA)*, pages 209–220. Austrian Computer Society. (Cited on pages 19, 103, 107, 122, 123 and 168.)

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265. (Cited on page 164.)

Vágvölgyi, S. (1986). On compositions of root-to-frontier tree transformations. *Acta Cybernetica*, 7(4):443–480. (Cited on page 167.)

Vágvölgyi, S. (1992). Top-down tree transducers with two-way tree walking look-ahead. *Theoretical Computer Science*, 93(1):43–74. (Cited on page 167.)

Čulík, K. (1965). Semantics and translation of grammars and ALGOL-like languages. *Kybernetika*, 1(1):47–49. (Cited on pages 17, 37, 40, 77 and 164.)

Čulík, K. (1966). Well-translatable grammars and ALGOL-like languages. In T.B. Steel, J., editor, *Proceedings of the IFIP Working Conference on Formal Language Description of Languages*, pages 76–85. North-Holland, Amsterdam. (Cited on page 164.)

Čulík, K. (1968). Well-translatable grammars and ALGOL-like languages. Technical Report AD-683 105, National Technical Information Service, Alexandria, VA. (Cited on page 164.)

Vere, S. (1970). *Syntax-Directed Translation of Context-free Languages*. Ph.D. thesis, UCLA, Los Angeles, CA, USA. (Cited on pages 35, 40, 45 and 165.)

Vijay-Shanker, K. and Weir, D. J. (1994). The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27(6):511–546. (Cited on page 75.)

Walter, H. K.-G. (1976). Grammarforms and grammarhomomorphisms. *Acta Informatica*, 7:75–93. (Cited on page 147.)

Weaver, W. (1955). Translation. In Locke, W. N. and Booth, A. D., editors, *Machine Translation of Languages*, pages 15–23. Technology Press of the MIT, Cambridge, Mass., and John Wiley and Sons, Inc., Wiley. (Cited on page 164.)

Wexelblat, R. L., editor (1981). *History of Programming Languages I.* ACM, New York, NY, USA. (Cited on page 164.)

Wilkes, M. V. (1951). The EDSAC computer. In *Proceedings of the International Workshop on Managing Requirements Knowledge (AFIPS)*, pages 79–79. (Cited on page 164.)

Wilkes, M. V. (1968). Computers then and now. *Journal of the ACM*, 15(1):1–7. (Cited on page 164.)

Wintner, S. (2002). Formal language theory for natural language processing. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics (ETMTNLP '02)*, volume vol. 1, pages 71–76. Association for Computational Linguistics. (Cited on page 38.)

Wintner, S. (October 2001). Formal language theory for natural language processing. Technical report, University of Haifa, Haifa, Israel. (Cited on pages 30, 35, 38, 39 and 164.)

Wu, D. (1995). Stochastic inversion transduction grammars, with application to segmentation, bracketing, and alignment of parallel corpora. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95), 2 Volumes*, pages 1328–1337. Morgan Kaufmann. (Cited on page 165.)

Wu, D. (1997). Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403. (Cited on pages 37, 40, 42, 47, 48, 142 and 165.)

Wu, D., Carpuat, M., and Shen, Y. (2006). Inversion transduction grammar coverage of arabic-english word alignment for tree-structured statistical machine translation. In Gilbert, M. and Ney, H., editors, *2006 IEEE ACL Spoken Language Technology Workshop, SLT 2006, Palm Beach, Aruba, December 10-13, 2006*, pages 234–237. (Cited on pages 142 and 165.)

Wu, D. and Fung, P. (2005). Inversion transduction grammar constraints for mining parallel sentences from quasi-comparable corpora. In Dale, R., Wong, K.-F., Su, J., and Kwong, O. Y., editors, *Natural Language Processing - IJCNLP 2005, Second International Joint Conference, Jeju Island, Korea, October 11-13, 2005, Proceedings*, volume 3651 of *Lecture Notes in Computer Science*, pages 257–268. Springer. (Cited on pages 142 and 165.)

Yamada, K. and Knight, K. (2001). A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics (ACL 2001)*, pages 523–530. Asociation of Computational Linguistics. (Cited on pages 18, 40, 42, 103, 142, 165, 167 and 168.)

Yamangil, E. and Shieber, S. M. (2010). Bayesian synchronous tree-substitution grammar induction and its application to sentence compression. In Hajic, J., Carberry, S., and Clark, S., editors, *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010)*, pages 937–947. The Association for Computer Linguistics. (Cited on page 169.)

Younger, D. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208. (Cited on page 164.)

Younger, D. H. (1966). Context-free language processing in time n^3. In *Conference Record 7th Annual Symposium on Switching and Automata Theory (FOCS)*, pages 7–20. IEEE Computer Society. (Cited on page 164.)

Yu, S. (1997). Regular languages. In Salomaa, A. and Rozenberg, G., editors, *Handbook of Formal Languages*, volume 1: Word, Language, Grammar, pages 41–110. Springer, Berlin. (Cited on pages 35, 37 and 164.)

Zhang, H. and Gildea, D. (2007). Factorization of synchronous context-free grammars in linear time. In *Proceedings of the NAACL-HLT 2007/AMTA Workshop on Syntax and Structure in Statistical Translation*, IEEE System Theory 2007 (SSST '07), pages 25–32. Association for Computational Linguistics. (Cited on pages 44 and 165.)

Zhang, H., Huang, L., Gildea, D., and Knight, K. (2006). Synchronous binarization for machine translation. In Moore, R. C., Bilmes, J. A., Chu-Carroll, J., and Sanderson, M., editors, *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL 2006)*, pages 256–263. Association for Computational Linguistics. (Cited on pages 44, 52, 165 and 169.)

Zhang, J., Zhai, F., and Zong, C. (2013). Syntax-based translation with bilingually lexicalized synchronous tree substitution grammars. *IEEE Transactions on Audio, Speech & Language Processing*, 21(8):1586–1597. (Cited on page 169.)

Zhang, M., Jiang, H., Aw, A., Li, H., Tan, C. L., and Li, S. (2008a). A tree sequence alignment-based tree-to-tree translation model. In McKeown, K., Moore, J. D., Teufel, S., Allan, J., and Furui, S., editors, *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL 2008)*, pages 559–567. The Association for Computer Linguistics. (Cited on pages 147 and 169.)

Zhang, M., Jiang, H., Aw, A. T., Sun, J., Li, S., and Tan, C. L. (2007). A tree-to-tree alignment-based model for statistical machine translation. In Maegaard, B., editor, *Proceedings of the Machine Translation Summit XI*, pages 535–542. European Association for Machine Translation. (Cited on page 169.)

Zhang, M., Jiang, H., Li, H., Aw, A., and Li, S. (2008b). Grammar comparison study for translational equivalence modeling and statistical machine translation. In Scott, D. and Uszkoreit, H., editors, *Proceedings of the 22nd International Conference on Computational Linguistics (COLING 2008)*, pages 1097–1104. (Cited on pages 147 and 169.)

UNIVERSITAT ROVIRA I VIRGILI
SYNTAX-DIRECTED TRANSLATIONS, TREE TRANSFORMATIONS AND BIMORPHISMS
Catalin Ionut Tirnauca

Index

ΣX-context, 56
ΣX-tree, 55
Σ-terms, 55
Σ-tree, 56
Σ-context, 56
ΣX-tree
    linear, 56
    non-deleting, 56

alphabet, 24
    leaf, 55
    rank
        supremum, 136
    ranked, 55
        binary, 55
        constant, 55
        unary, 55
    subalphabet, 25
    terminal, 26

bimorphism
    string, 38
    tree, 101

closure under composition, 36, 111, 128, 132, 135, 136, 141–145, 147
context, 56
context-free grammar, 31
    ε-free, 32
    Chomsky normal form for, 33
    cycle-free, 33
    derivation tree of a, 31
    leftmost derivation, 31
    proper, 33
    rightmost derivation, 31
    without useless symbols, 33
context-free language, 31
    generated by a context-free grammar, 31
context-free tree grammar, 74
context-sensitive grammar, 28

decision problem, 30
    decidable, 30
    emptiness, 30
    equivalence, 30
    finiteness, 30
    inclusion, 30
    membership, 30
    undecidable, 30
decomposition, 36
derivation, 27
    leftmost, 31
    rightmost, 31
derivation tree, 31

Dewey notation, 57

effective, 24, 33, 45, 46, 51, 52, 67, 75, 89, 120, 124, 130, 134, 137

f-catenation, 62
finite automaton
    with output, 37
finite-state transducer, 37
    deterministic, 38
    translation computed by a, 38

generalized synchronous tree-substitution grammar, 97
generating device, 26
grammar, 26
    context-free, 31
    context-sensitive, 28
    derivation, 27
    directly derives, 27
    equivalent, 27
    language generated by, 27
    linear, 31
    nonterminals of a, 26
    phrase structure, 26
    production of a, 27
        left-hand side, 27
        right-hand side, 27
    right-linear, 28
    sentential form, 27
    start symbol of a, 27
    terminals of a, 26
    tree-substitution, 68

Hasse diagram, 29, 53, 75, 88–90, 127, 139
    classes of tree transformations defined by tree bimorphisms and tree transducers, 139
    original Chomsky hierarchy, 29
homomorphism, 25
    image under a, 25
    inverse, 25
    inverse image under a, 25
    projection, 25
    tree, 58

inversion transduction grammar, 43

language, 25
    ε-free, 25
    closure under an operation, 25
    complement, 25
    concatenation, 25
    difference, 25

intersection, 25
Kleene closure, 25
linear, 32
positive closure, 25
union, 25
yield, 63
linear language
accepted by a one-turn pushdown automaton, 32
generated by a linear grammar, 32
linear syntax-directed translation schema, 43
local tree language, 70

magmoids, 147
mapping, 24
bijective, 24
composition, 24
extension of a, 24
homomorphism
string, 25
tree, 58
identity, 24
image under a, 24
injective, 24
inverse, 24
partial, 24
permutation, 24
pre-image, 24
rank, 55
restriction of a, 24
surjective, 24
total, 24
yield
generalized, 56

next-configuration relation, 38
defined by a generalized sequential machine, 38

parsing, 26
phrase structure grammar, 26
context-free, 31
context-sensitive, 28
linear, 31
right-linear, 28
production tree, 73, 112
in a syntax-directed translation schema, 112
of a context-free grammar, 73
pushdown automaton, 31
one-turn, 32
pushdown transducer, 44
k-register, 45
deterministic, 45
nondeterministic, 45

quasi-alphabetic
canonical representation, 107
comblike tree bimorphism, 118
decidability results, 106
production tree, 112
translation, 104, 112
tree bimorphism, 103
tree transducer, 124
tree transformation, 103
closure under composition, 111
closure under inverses, 109
closure under union, 108
locally finite, 106
not closed under intersection, 109
preservation of recognizability, 106

recognizable tree language, 63
recognizer, 26
deterministic, 26
language accepted by a, 26
nondeterministic, 26
tree, 63
regular language
generated by a right-linear grammar, 28
regular translation, 37
defined by a finite-state transducer, 37
regular tree grammar, 67
normal form of a, 67
tree-substitution grammar, 68
relabeling, 85
finite-state, 86
total, 85
relation, 23
composition, 24
converse, 24
derivation, 27
domain of a, 24
identity, 24
range of a, 24
reflexive transitive closure of a, 24
total, 24
transitive closure, 24
right-linear syntax-directed translation schema, 43

set theory, 23
cardinality, 23
Cartesian product, 23
complement, 23
difference, 23
empty set, 23
incomparability, 23
indexed family, 23
intersection, 23
power set, 23
finite, 23
proper inclusion, 23
subset, 23
superset, 23
union, 23
string, 25

concatenation, 25
empty, 25
homomorphism, 25
length of a, 25
non-empty, 25
prefix, 25
reverse, 25
substring, 25
suffix, 25
string bimorphism, 38
syntax-directed translation characterized by a language, 48
syntax-directed translation strongly characterized by a language, 48
translation defined by, 39
substitution, 58
symmetrically locally finite, 106
synchronous context-free grammar, 44
translation defined by , 44
tree transformation defined by, 113
typical production in a, 44
synchronous grammar, 37
generalized synchronous tree-substitution grammar, 97
inversion transduction grammar, 43
linear syntax-directed translation schema, 43
right-linear syntax-directed translation schema, 43
simple syntax-directed translation schema, 43
synchronous context-free grammar, 44
synchronous translation generator, 149
synchronous tree transformation generator, 155
synchronous tree-substitution grammar, 92
syntax-directed translation schema, 41
synchronous translation generator, 149
(leftmost) derivation step, 150
associated nonterminals, 150
homogenous, 151
input grammar of„ 149
input nonterminal, 150
linear, 151
non-deleting, 151
normal form of a, 151
order-preserving, 151
output grammar of„ 149
output nonterminal, 150
simple, 151
translation form, 150
synchronous tree transformation generator, 155
synchronous tree-substitution grammar, 92
syntax-directed translation, 42
characterized by a language, 48
strongly characterized by a language, 48
syntax-directed translation schema, 41
a typical production in an, 41
associated nonterminals in a, 41
characterized by a language, 48
derivation in a, 41
in strong normal form, 47
in weak normal form, 47
input alphabet, 41
input grammar of a, 41
inversion transduction grammar, 43
leftmost derivation in a, 42
length of a production, 95
linear, 43
nonterminal symbols of a, 41
order of a, 43
output alphabet, 41
output grammar of a, 41
production set of a, 41
proper, 46
right-linear, 43
simple, 43
strongly characterized by a language, 48
syntax-directed translation defined by a, 42
translation forms in a, 41
tree transformation defined by, 113

translation, 35
alphabetic, 128
closed under composition, 36
composition, 36
converse, 36
decomposed, 36
defined by a string bimorphism, 39
domain of a, 36
identity, 37
input alphabet of a, 35
output alphabet, 35
permuting, 128
pre-image of a, 36
quasi-alphabetic, 104
range of a, 36
regular, 37
set of translated elements in a, 36
syntax-directed, 42
tree, 56
subtree rooted at a node in, 57
branch in a, 58
domain, 57
fork of a, 56
height of a, 56
homomorphism, 58
induction, 56
instance of a, 58
label of a node in a, 57
node of a, 57
position in a, 57
replacement in a, 57
representation, 80
root label of a, 56
shape of a, 58
subtrees of a, 56

yield of a, 56
tree bimorphism, 101
Σ-algebraic, 145
Σ-rational, 144
almost variable-free, 102
alphabetic, 103
bitransformation, 147
center language of a, 101
fine, 103
input tree homomorphism of a, 101
linear non-deleting, 103
linear non-deleting strict, 142
output tree homomorphism of a, 101
permuting, 103
pictorial representation of a, 102
quasi-alphabetic, 103
rational relation of binary trees, 141
synchronized transduction, 143
translation defined by a, 102
tree transformation defined by a, 101
variable-free, 102
tree grammar
context-free, 74
regular, 67
tree-substitution, 68
tree homomorphism, 58
alphabetic, 59
deleting, 59
fine, 60
linear, 59
non-deleting, 59
non-linear, 59
normalized, 59
permuting, 59
quasi-alphabetic, 59
strict, 59
symbol-to-symbol, 60
tree language, 62
x-iteration, 62
almost variable-free, 62
Cartesian product, 62
complement, 62
context-free, 74
deterministic recognizable, 65
difference, 62
f-concatenation, 62
intersection, 62
local, 70
recognizable, 63
union, 62
variable-free, 62
x-product, 62
x-quotient, 62
tree recognizer
deterministic top-down, 65
nondeterministic bottom-up, 65
nondeterministic top-down, 63
tree transducer
bottom-up, 83
deterministic, 84
linear, 84
non-deleting, 84
total, 84
extended top-down, 86
epsilon-free, 88
linear, 88
non-deleting, 88
non-erasing, 88
quasi-alphabetic, 124
with finite look-ahead, 88
with regular look-ahead, 88
extended top-down tree-to-string, 142
finite-state relabeling, 85
permuting, 130
quasi-alphabetic, 124
relabeling, 85
top-down, 80
deterministic, 82
extended, 86
linear, 82
non-deleting, 82
total, 82
with finite look-ahead, 82
with regular look-ahead, 82
tree transformation, 78
Σ-algebraic, 145
Σ-rational, 144
alphabetic, 128
composition, 79
converse, 78
fine, 135
finite-state relabeling, 86
permuting, 128
quasi-alphabetic, 103
rational relation of binary trees, 141
relabeling, 85
symetrically locally finite, 106
synchronized transduction, 143
tree-substitution grammar, 68

x-iteration, 62
x-product, 62
x-quotient, 62

yield, 56
generalized yield mapping, 56
yield language, 63

List of Notation

Acronyms

BOT	Bottom-up tree transducer	83
CFG	Context-free grammar	31
CFL	Context-free language	30
CFTG	Context-free tree grammar	74
CFTL	Context-free tree language	74
CNF	Chomsky normal form	33
ITG	Inversion transduction grammar	43
FST	Finite-state transducer	37
GSTSG	Generalized synchronous tree-substitution grammar	97
ndT	Nondeterministic top-down tree recognizer	63
SCFG	Synchronous context-free grammar	44
SDT	Syntax-directed translation	42
SDTS	Syntax-directed translation schema	41
STG	Synchronous translation generator	149
STSG	Synchronous tree-substitution grammar	92
STTG	Synchronous tree transformation generator	155
TOP	Top-down tree transducer	80
TOP^{R}	Top-down tree transducer with regular look-ahead	82
TSG	Tree substitution grammar	68
XTT	Extended top-down tree transducer	86

Devices

BR	A bottom-up tree recognizer (Q, Σ, X, P, F)	65
BU	A bottom-up tree transducer $(Q, \Sigma, X, \Omega, Y, P, F)$	83
CF	A context-free grammar (N, X, P, S)	31
CT	A context-free tree grammar (N, Σ, X, P, S)	74
G	A phrase structure grammar (N, X, P, S)	26
FT	A finite-state transducer $(Q, X, Y, \kappa, q_0, F)$	37
GS	A generalized synchronous tree-substitution grammar $(N, \Sigma, X, N', \Omega, Y, P, S, S')$	97
LG	A linear grammar (N, X, P, S)	31
RT	A regular tree grammar (N, Σ, X, P, S)	67
SB	A string bimorphism (μ, L, ν)	38
SC	A synchronous context-free grammar (N, X, Y, P, S, S')	44
SD	A syntax-directed translation schema (N, X, Y, P, S)	41
SG	A synchronous translation generator (N, X, Y, P, S, S')	149
ST	A synchronous tree-substitution grammar $(N, \Sigma, X, N', \Omega, Y, P, S, S')$	92
TD	A top-down tree transducer $(Q, \Sigma, X, \Omega, Y, P, I)$	80
TG	A synchronous tree transformation generator $(N, \Sigma, X, \Omega, Y, P, S, S')$	155
TR	A top-down tree recognizer (Q, Σ, X, P, I)	63
TS	A tree substitution grammar (N, Σ, X, P, S)	68
TB	A tree bimorphism (φ, R, ψ)	101
XT	An extended top-down tree transducer $(Q, \Sigma, X, \Omega, Y, P, I)$	86

Greek alphabet

α	A sentential form in a (synchronous) grammar	27

β	A sentential form in a (synchronous) grammar	27
δ	A sentential form in a (synchronous) grammar	27
γ	A sentential form in a (synchronous) grammar	27
ε	The empty string	25
η	A relabeling	85
θ	A substitution	58
κ	The transition mapping of a finite-state transducer	37
λ	A translation	37
μ	A (string) homomorphism	25
ν	A (string) homomorphism	25
ξ	A special symbol used to define contexts	56
π	A projection (tree) homomorphism	107
ρ	A binary relation	23
σ	A permutation	24
τ	A tree transformation	78
υ	A tree homomorphism	60
ϕ	A bijective mapping	24
φ	A tree homomorphism	60
ψ	A tree homomorphism	60
ω	A position or node in a tree	57
Γ	A ranked alphabet	55
Ξ	The set of auxiliary variables, which identifies the subtrees of a tree	58
Σ	A ranked alphabet	55
Ω	A ranked alphabet	55

Language classes

CFL	All context-free languages	31
$CFTL$	All context-free tree languages	74
CSL	All context-sensitive languages	29
Fam	Any family of tree languages	62
$\mathrm{Fam}_\Sigma(X)$	The set of all ΣX-tree languages in Fam	62
Fam_Σ	The set of all Σ-tree languages in Fam	62
$\mathrm{Fam}^{\mathrm{vf}}$	The family of all variable-free tree languages in Fam	62
$\mathrm{Fam}^{\mathrm{avf}}$	The family of all almost variable-free tree languages in Fam	62
LIN	All linear languages	32
Loc	All local tree languages	70
RE	All recursively enumerable languages	29
Rec	All recognizable tree languages	64
DRec	All deterministic recognizable tree languages	65
REG	All regular languages	29
$T[TSG]$	All tree languages generated by tree substitution grammars	69

Miscellaneous

$\cap$	Intersection of sets	23
$\cup$	Union of sets	23
$\setminus$	Difference between sets	23
$\times$	Cartesian product of sets	23
$\subseteq$	Inclusion between sets	23
$\subset$	Proper inclusion between sets	23
$\parallel$	Incomparability between sets	23
$\vdash$	The next-configuration relation of a recognizer	38
$\Rightarrow$	The derivation relation of a generator or a tree machine	27

$\circ$	Composition operator	24
$\emptyset$	The empty set	23
$\mathbb{N}$	The set of all natural numbers, i.e., of all non-negative integers	23
$\mathbb{N}_+$	The set of all positive integers	23

Operations, relations and sets related to strings

$\|v\|$	The length of the string v	25
$\|v\|_x$	The number of occurrences of the letter x in the string v	25
$\|v\|_Y$	The length of the string obtained by erasing from v all letters not in Y	25
v^n	The n-times concatenation of the string v with itself	25
v^R	The reversal of the string v	25
L^*	The closure of the language L	25
L^+	The positive closure of the language L	25
L^n	The n-times concatenation of the language L with itself	25
ρ^{-1}	The inverse of a relation ρ	24
$\mathrm{Dom}(\rho)$	The domain of a relation ρ	24
$\mathrm{Range}(\rho)$	The range of a relation ρ	24
ρ^*	The reflexive and transitive closure of a relation ρ	24
ρ^+	The transitive closure of a relation ρ	24
id_X	The identity (diagonal) relation of a set X	24
t_X	The total relation of a set X	24
1_X	The identity mapping for the set X	24
$\phi\|_Z$	The restriction of the mapping ϕ to the set Z	24
pr_Y	The projection homomorphism on a subalphabet Y	25
$\|X\|$	The cardinality of the set X	23
X^C	The complement of the set X	23
X^n	The Cartesian power of the set X	23
$\wp(X)$	The power set of the set X	23
$\wp_F(X)$	The set of finite subsets of X	23
X^*	The set of all strings over the alphabet X	25
X^+	The set of all non-empty strings over the alphabet X	25
Y^X	The set of all mappings from X to Y	24
$[n]$	All positive integers between 1 and n	23

Operations, relations and sets related to trees

$\|t\|_f$	The number of nodes labeled with the symbol f in the tree t	56
$\|t\|_x$	The number of leaves in t labeled with the letter x	56
$\mathrm{hg}(t)$	The height of a tree t	56
$\mathrm{br}(t)$	The set of all paths from the root to the leaves of the tree t	58
$\mathrm{root}(t)$	The label of the root of the tree t	56
$\mathrm{sub}(t)$	The set of subtrees of the tree t	56
$\mathrm{dom}(t)$	The domain of a tree, i.e., its set of nodes	57
$\mathrm{dom}_D(t)$	The positions in the domain of t labeled with symbols in D	57
$\mathrm{yd}(t)$	The yield of the tree t	56
$\mathrm{yd}(T)$	The yield language of the tree language T	63
$c(t)$	The ΣX-tree obtained when the ξ in c is replaced with t	56
$c'(c)$	The ΣX-context obtained from c' by replacing the ξ in it with c	56
$g^n(t)$	A tree $g(g^{n-1}(t))$	56
$t(\omega)$	The label of a tree t at position $\omega \in \mathrm{dom}(t)$	57
$t\|_\omega$	The subtree of t rooted at node ω	57
$t[u]_\omega$	The replacement of the subtree of t rooted at node ω by the tree u	57
$t[t_1, \ldots, t_n]$	Simultaneously replace in t each occurrence of ξ_i with t_i $(i \in [n])$	58

rk	The rank mapping	55
fork(t)	The set of forks of the tree t	56
fork(Σ, X)	The set of all possible forks of ΣX-trees	56
$\mathrm{yd}^{-1}(L)$	The set of all ΣX-trees t such that $\mathrm{yd}(t) \in L$	63
Ξ_n	The set of the variables $\xi_1, \ldots, \xi_n$	58
var(t)	The set of all variables $\xi_i \in \Xi_n$ appearing in t	58
$c(T)$	The set of all trees $c(t)$, with $t \in T$	63
$c^{-1}(T)$	The set of all ΣX-trees t such that $c(t) \in T$	63
$T[T_1, \ldots, T_n]$	The set of all trees $t[t_1, \ldots, t_n]$ with $t \in T$, $t_1 \in T_1$ and $t_n \in T_n$	62
$f(T_1, \ldots, T_m)$	The set of all trees $f(t_1, \ldots, t_m)$ with $t_1 \in T_1, \ldots, t_m \in T_m$	62
$T \bullet_x U$	The set of trees $u[x \leftarrow (t_1, \ldots, t_n)]$ ($u \in U$, $n = \lvert u \rvert_x$, $t_1, \ldots, t_n \in T$)	62
$T /_x U$	The set of all ΣX-trees t such that $(T \bullet_x \{t\}) \cap U \neq \emptyset$	62
T^{*x}	The x-iteration of the ΣX-tree language T	62
Σ_m	The set of all symbols of rank m in Σ	55
$\vee$	The supremum operator over two ranked alphabets	136
$\Sigma^{[n]}$	A special ranked alphabet with trees as symbols	136
$\Sigma(U)$	The set of all trees with root in Σ and its direct subtrees in U	55
$Q(H)$	The set of trees $q(t)$ with $t \in H$ and q a unary symbol in Q (see $\Sigma(U)$)	80
$T_\Sigma(X)$	The set of all Σ-terms indexed by variables in X (i.e., of all ΣX-trees)	55
T_Σ	The set of all Σ-trees	56
$\widetilde{T}_\Sigma(X \cup \Xi_n)$	The set of $\Sigma(X \cup \Xi_n)$-trees with $\mathrm{yd}_{\Xi_n}(t) = \xi_1 \ldots \xi_n$	58
$T_\Omega(Y \cup Q(H))$	A special set of trees that have a unique representation	80
$C_\Sigma(X)$	The set of all ΣX-contexts	56
C_Σ	The set of all Σ-contexts	56
$C^n_\Sigma(X)$	The set of $\Sigma(X \cup \Xi_n)$-trees in which each $\xi_i (i \in [n]$ appears exactly once	58

Roman alphabet

c	A context	56
d	A leaf letter or a ranked symbol, usually a constant	63
e	A constant, i.e., nullary symbol	55
f	A ranked symbol, usually binary	55
g	A ranked symbol, usually unary	55
h	A ranked symbol	55
i	An index or counter, usually from 1 to l, m or n	23
j	An index or counter, usually from 1 to l, m or n	24
k	Any natural number such as the order of an SDTS	43
l	Any natural number	113
m	Any natural number such as the rank of a symbol	55
n	Any natural number such as the number of auxiliary variables	23
p	A production in a grammar or a rule in a transducer or recognizer	113
q	An internal state of a (tree) recognizer or (tree) transducer	63
r	A tree	56
s	A tree	56
t	A tree	56
u	A tree	56
v	A string	25
w	A string	25
x	A letter, usually in X	24
y	A letter, usually in Y	24
z	A letter, usually in Z	24
A	A nonterminal in a (synchronous) grammar	27
B	A nonterminal in a (synchronous) grammar	27
C	A nonterminal in a (synchronous) grammar	27
D	A set of letters and ranked symbols	56
E	A set of forks	70

F The set of final states of a tree recognizer or tree transducer 65
G A phrase structure grammar (N, X, P, S) 26
I The set of initial states of a (tree) recognizer or (tree) transducer 63
J An index set 23
K A string language 25
L A string language 25
M The look-ahead facility of a tree transducer 82
N The set of nonterminals of a (synchronous) grammar 26
P The set of productions of a grammar or of rules of a transducer 27
Q The set of internal states of a (tree) recognizer or (tree) transducer 63
R A recognizable tree language 64
S The start symbol of a (synchronous) grammar 27
T A tree language or a set of trees 62
U A tree language or a set of trees 62
X A finite (leaf) alphabet 24
Y A finite (leaf) alphabet 24
Z A finite (leaf) alphabet 24

Translation classes

λ[BOT] All translations definable by BOT-transducers 83
$\lambda[ITG]$ All translations definable by ITGs 43
$\lambda[GSTSG]$ All translations definable by GSTSGs 97
$\lambda[FST]$ All translations definable by FSTs, i.e., regular translations 38
$\lambda[aTB]$ All translations definable by alphabetic tree bimorphisms 128
$\lambda[fTB]$ All translations definable by fine tree bimorphisms 135
$\lambda[lnTB]$ All translations definable by linear non-deleting tree bimorphisms 104
$\lambda[pTB]$ All translations definable by permuting tree bimorphisms 128
$\lambda[qTB]$ All translations definable by quasi-alphabetic tree bimorphisms 104
$\lambda[SCFG]$ All translations definable by SCFGs 44
$\mathcal{SDT}$ All syntax-directed translations 42
$\lambda[SDTS]$ All syntax-directed translations (defined by SDTSs) 42
$\lambda[k\text{-}SDTS]$ All syntax-directed translations of order k 43
$\lambda[lSDTS]$ All linear syntax-directed translations 43
$\lambda[rSDTS]$ All right-linear syntax-directed translations 43
$\lambda[sSDTS]$ All translations definable by simple SDTSs 43
$\lambda[STG]$ All translations definable by STGs 150
$\lambda[STSG]$ All translations definable by STSGs 93
$\lambda[STTG]$ All translations definable by STTGs 156
λ[TOP] All translations definable by TOP-transducers 81
$\lambda[\text{TOP}^{\text{R}}]$ All translations definable by TOP^{R}-transducers 82
$\lambda[\text{TOP}^{\text{F}}]$ All translations definable by TOP-transducers with finite look-ahead 82
λ[l-TOP] All translations definable by linear TOP-transducers 82
λ[ln-TOP] All translations definable by linear non-deleting TOP-transducers 82
λ[XTT] All translations definable by XTT-transducers 87
λ[q-XTT] All translations definable by quasi-alphabetic tree transducers 124
λ[ln-XTT] All translations definable by linear non-deleting XTT-transducers 88

Tree bimorphism classes

$\mathbf{B}[\mathcal{H}_1, \text{Fam}, \mathcal{H}_2]$ All tree bimorphisms (φ, R, ψ) with $\varphi \in \mathcal{H}_1$, $R \in \text{Fam}$ and $\psi \in \mathcal{H}_2$ 102
$\tau[\mathcal{H}_1, \text{Fam}, \mathcal{H}_2]$ All tree transformations defined by $\mathbf{B}[\mathcal{H}_1, \text{Fam}, \mathcal{H}_2]$ 102
$\lambda[\mathcal{H}_1, \text{Fam}, \mathcal{H}_2]$ All translations defined by $\mathbf{B}[\mathcal{H}_1, \text{Fam}, \mathcal{H}_2]$ 102
$\mathbf{B}[\text{aH}, \text{Rec}, \text{aH}]$ All alphabetic tree bimorphisms 103

B[fH, Rec, fH] All fine tree bimorphisms 103
B[pH, Rec, pH] All permuting tree bimorphisms 103
B[qH, Loc, qH] All quasi-alphabetic tree bimorphisms 103
B[lnH, Rec, lnH] All linear non-deleting tree bimorphisms 103

Tree homomorphism classes

aH The class of all alphabetic tree homomorphisms 60
fH The class of all fine tree homomorphisms 60
lH The class of all linear tree homomorphisms 60
nH The class of all non-deleting tree homomorphisms 60
qH The class of all quasi-alphabetic tree homomorphisms 60
pH The class of all reordering tree homomorphisms 60
sH The class of all strict tree homomorphisms 60
ssH The class of all symbol-to-symbol tree homomorphisms 60
lnH The class of all linear non-deleting tree homomorphisms 60

Tree transformation classes

τ[BOT] All tree transformations computed by BOT-transducers 83
τ[l-BOT] All tree transformations computed by linear BOT-transducers 84
τ[ln-BOT] All tree transformations computed by linear non-deleting BOT-transducers 84
$\tau[aTB]$ All tree transformations defined by alphabetic tree bimorphisms 128
$\tau[fTB]$ All tree transformations defined by fine tree bimorphisms 135
$\tau[lnTB]$ All tree transformations defined by linear non-deleting tree bimorphisms 104
$\tau[pTB]$ All tree transformations defined by permuting tree bimorphisms 128
$\tau[qTB]$ All tree transformations defined by quasi-alphabetic tree bimorphisms 104
FTA All recognizable tree languages viewed as tree transformations 86
HOM All tree homomorphisms viewed as tree transformations 85
QREL All finite-state relabellings 86
REL All relabellings 85
$\tau[SCFG]$ All tree transformations defined by SCFGs 113
$\tau[SDTS]$ All tree transformations defined by SDTSs 113
$\tau[GSTSG]$ All tree transformations defined by GSTSGs 97
$\tau[STSG]$ All tree transformations defined by STSGs 93
$\tau[STTG]$ All tree transformations defined by STTGs 156
τ[TOP] All tree transformations computed by TOP-transducers 81
τ[l-TOP] All tree transformations computed by linear TOP-transducers 82
τ[ln-TOP] All tree transformations computed by linear non-deleting TOP-transducers 82
$\tau[\text{TOP}^{\text{F}}]$ All tree transformations computed by TOP-transducers with finite look-ahead 82
$\tau[\text{TOP}^{\text{R}}]$ All tree transformations computed by TOP^{R}-transducers 82
τ[XTT] All tree transformations computed by XTT-transducers 87
τ[q-XTT] All tree transformations computed by quasi-alphabetic tree transducers 124
τ[ln-XTT] All tree transformations computed by linear non-deleting XTT-transducers 88
$\tau[\text{XTT}^{\text{F}}]$ All tree transformations computed by XTT-transducers with finite look-ahead 88
$\tau[\text{XTT}^{\text{R}}]$ All tree transformations computed by XTT-transducers with regular look-ahead 88