Tree Transducer Composition as Program Transformation

Dissertation

zur Erlangung des akademischen Grades
Doktor rerum naturalium (Dr. rer. nat.)

vorgelegt an der

Technischen Universität Dresden
Fakultät Informatik

eingereicht im September 2004 von

Diplom-Informatiker Janis Voigtländer
geb. am 20. November 1977 in Halle/Saale

Betreuer: Dr.-Ing. Armin Kühnemann
Betreuender Hochschullehrer: Prof. Dr.-Ing. habil. Heiko Vogler

Gutachter: Dr.-Ing. Armin Kühnemann, TU Dresden
Prof. Dr. rer. nat. habil. Horst Reichel, TU Dresden
Prof. Dr. phil. nat. habil. Helmut Seidl, TU München

Tag der Verteidigung: 24. Januar 2005

Fassung August 2005

Abstract

Nonstrict, purely functional programming languages offer a high potential for the modularization of software. But beside their advantages with respect to reliability and reusability, modularly specified programs often have the disadvantage of low execution efficiency, caused in particular by the creation and consumption of structured intermediate results. One possible approach to cure this conflict is the automatic, semantics-preserving optimization of programs, for which purely functional languages are again particularly suited due to their mathematical foundation.

This dissertation studies a specific transformation for the elimination of intermediate results (for so called *deforestation*) regarding its impact on the program efficiency under nonstrict evaluation. The formal framework is provided by concepts from the theory of *tree transducers*. One special feature of the transformation under consideration is the successful handling of *accumulating parameters*, which find frequent use in functional programs. The core of the thesis is the derivation of effectively decidable, syntactic conditions on the original program under which the transformed program is to be preferred over it with respect to efficiency.

Acknowledgments

The foundation for this thesis was laid during my Masters studies when Heiko Vogler and Armin Kühnemann introduced me to tree transducers and program transformation. Afterwards, they employed me on a project position that provided ideal conditions for conducting the research reported here. It was funded by the "Deutsche Forschungsgemeinschaft" under grants KU 1290/2-1, 2-3, and 2-4. I also received generous financial support from Heiko's chair, the "Gesellschaft von Freunden und Förderern der TU Dresden", and the "Researcher Travel Grant Program by European Commission DG Information Society" for presenting my results at various conferences. Helmut Seidl invited me to give a colloquium talk at TU München and later accepted to serve as external referee for this thesis. I am likewise obliged to Horst Reichel for accepting to serve as referee in Dresden.

Heiko and Armin influenced my scientific development in many ways over the last years through encouragement, guidance, and insistence on both mathematical rigor and clear presentation. As project leader and PhD supervisor, Armin granted me considerable freedom in choosing research directions to pursue but always remained available for giving advice. I also learned from collaboration with my coauthors Armin, Jürgen Giesl, and Patricia Johann. Special thanks are due to Armin and Patricia for traveling to present our respective joint papers at conferences when I would not leave my family in the making. Discussions with Claus Jürgensen and Sebastian Maneth increased my understanding of related work. Andreas Maletti and Stefan Reuther implemented my composition construction and a preliminary version of the formal efficiency analysis, thus providing useful tools for working with examples.

В заключение бих искал да благодаря на съпругата си, Марина, за силната ѝ вяра в мен и на сина си, Кирил, защото е истинско чудо. Обичам ви.

Contents

1 Introduction **1**
- 1.1 Functional languages . . . 1
- 1.2 Deforestation and accumulating parameters . . . 3
- 1.3 Tree transducers and their composition . . . 4
- 1.4 Formal efficiency analysis . . . 4
- 1.5 The contents of this thesis . . . 7

2 Preliminaries **9**
- 2.1 Trees, substitutions, and rewrite systems . . . 10
- 2.2 Macro tree transducers . . . 12
- 2.3 About proofs in this thesis . . . 15

3 Tree transducer composition **19**
- 3.1 The basic ideas . . . 19
- 3.2 The formal construction . . . 21
- 3.3 Application to the running example . . . 24

4 Formal efficiency analysis **29**
- 4.1 Call-by-name reduction . . . 30
- 4.2 Outline of the analysis . . . 33
- 4.3 Annotating programs to reflect computational costs . . . 34
- 4.4 Pushing annotation through composition . . . 39
- 4.5 Combining annotations, and a first efficiency result . . . 45
- 4.6 Unifying the treatment of ∘- and •-symbols . . . 48
- 4.7 Moving •-symbols to cancel out ∘-symbols . . . 53
- 4.8 Constantly bounding the number of ∘-symbols . . . 58
- 4.9 Assembling the call-by-name analysis . . . 60
- 4.10 From call-by-name to call-by-need . . . 64

5 Efficiency of the analysis **69**

6 Aspects of the transformation in practice **77**
- 6.1 Infinite and partial trees . . . 77
- 6.2 Partial demand . . . 84
- 6.3 Tail calls . . . 88

7 Other techniques and avenues for future research **91**
7.1 Tree transducer composition 91
7.1.1 Composition via attributed tree transducers 92
7.1.2 Composition for functions of linear size increase 93
7.1.3 Composition in a categorical framework 95
7.2 Classical deforestation 98
7.2.1 Efficiency comparison based on examples 100
7.2.2 An annotation approach to formal comparison 101
7.3 Lazy composition 103
7.4 Free theorems-based program transformations 108
7.5 Formal efficiency analysis 113
7.5.1 Efficiency analysis for functional programs in general 113
7.5.2 Efficiency analysis for tree transducer composition 114
7.6 Deaccumulation — Improving provability 115

8 Conclusion **121**

Bibliography **123**

Proof appendix **133**

Index **167**

Chapter 1

Introduction

This thesis is about program transformation for lazy functional languages. In the introduction we consider the functional paradigm, recall its advantages with respect to modularity of programs but also the associated costs, and point out automatic transformation of programs as a possible solution to this antagonism. Motivated by failures of classical such transformation techniques for programs using a particular style that is used quite frequently in functional languages, we consider *tree transducer composition*. The thesis centers around establishing efficiency guarantees for the latter technique. The approach used in this formal analysis is briefly discussed in this introduction, which ends with an overview of the remaining chapters and the contributions therein.

1.1 Functional languages

Functional programming is based on the mathematical concept of functions, transferring values to values and doing nothing else. This is different from the concept of functions or procedures in imperative languages, where they may additionally alter some global state. In fact, the essence of (pure) functional languages is usually described as "no side-effects", implying that every expression in such a language has a value that is independent of when it is evaluated. This relieves the programmer from being explicit about the control flow by specifying sequences of commands. Instead, a program is a collection of equations denoting the return values of functions for given arguments by expressions (potentially containing variables for abstracted parts of the input). This declarative style of concentrating on what to compute rather than detailing the order of actions in the computation leads to more productive programming by eliminating common sources of error and by increasing both the informal and the formal understanding of the meaning of programs. In particular, the fact that two expressions with the same value can be replaced for each other in any program context without changing the overall semantics leads to the powerful principle of equational reasoning (cf. e.g. [BW88]). The concise semantic setting also encourages the use of concepts like strong typing, polymorphism, and higher-order features, which are deeply rooted in functional programming even though in

principle they are not particular to this paradigm alone. All these aspects enable programmers to work on an appropriate level of abstraction, typically leading to succinct and highly modular programs.

It has been argued [Hug89] that the high potential for modularization (rather than a lack of side-effects as such) is the key advantage of functional languages over their competitors as the former better meet the software engineering concerns for reliability, maintainability, and reusability. But it is equally well known that this advantage also exacts a price, namely in the efficiency of programs. Solving an overall problem by composing functions that solve subproblems means that intermediate results have to be created at interfacing positions, and the (good) tendency in functional languages to structure a program into many small functions that are then combined by composition means an inflation of such intermediate results, which is particularly costly if these are large data structures rather than primitive values. Fortunately, functional languages also offer a cure for this tension between modularity and efficiency. Due to their concise semantic base, functional programs are very amenable to automatic transformation techniques, probably more so than programs in any other paradigm. In particular, semantics-preservation can also be established for relatively high-level program transformations that effect an actual algorithmic change in contrast to the transformations on the instruction level typically found in compilers for imperative languages. Hence, one can aim at transforming a modular but inefficient program into a monolithic, efficient program. The latter may lack the benefits attending modularity, but as long as the transformation is performed automatically and semantic equivalence of the transformed program to the original one is guaranteed, the programmer need not be concerned. This thesis studies one such transformation, in particular its impact on efficiency. Before going into more detail, we briefly situate our work with respect to the different flavors of functional languages in existence.

The first important distinction to be made about functional languages relates to whether every argument of a function needs to be defined in order for the function result to be defined. If this is the case, the language is called *strict*, otherwise *nonstrict*. Strict languages typically have more efficient implementations because they can use more efficient evaluation strategies to turn declarative function specifications into executable programs. Nonstrict languages, on the other hand, have richer equational theories because programs terminate more often. Moreover, nonstrictness enables to decompose problems in unique ways that are not available in a strict language. Indeed, [Hug89] arrives at the conclusion that nonstrictness (in the form of *lazy evaluation*) is perhaps the most powerful tool for structuring functional programs, adding considerably to their expressiveness. Our transformation is applicable in both strict and nonstrict languages, but for the analysis of efficiency we will consider evaluation strategies for nonstrict languages only. Another important distinction is that between *pure* and *impure* languages. The mathematical concept of functions as base for a programming language has its weaknesses when one needs to describe computational aspects such as generation of unique labels or interaction with the user. In strict languages this is typically resolved by allowing

side-effects to occur, which makes them impure, thus retracting some of the benefits of the functional paradigm. In nonstrict languages the favored solution is to use *monads* [Wad92] to integrate the missing features without sacrificing purity. The programs handled by our transformation do not use any "nonfunctional" features, so even in an impure language they fit into the pure core.

1.2 Deforestation and accumulating parameters

Under the name *deforestation*, automatic elimination of structured intermediate results (i.e. trees) was pioneered in [Wad90] through an algorithmic instance of the *unfold/fold-technique* [BD77] that is suitable for integration into optimizing compilers for functional languages. This unfold/fold-based approach has seen many extensions—e.g. in [HJ92], [MW93], and [Chi94]—, which we will collectively refer to as *classical deforestation* in the following. A fundamental weakness of classical deforestation is its inability to eliminate intermediate results that are built inside so called *accumulating parameters*. This problem is also shared by *shortcut deforestation* techniques [GLP93, Gil96, Chi99, Joh02, Sve02], based on denoting producers and consumers of intermediate results in terms of certain higher-order, polymorphic combinators, compositions of which can be transformed by rules that are justified using *free theorems* [Wad89b].

An accumulating parameter is one that may grow in recursive function calls. Consider, for example, the following definition of a function g with two parameters, where a case distinction is performed on the first argument and equations are written as directed rules:

$$\begin{array}{lll} g\ (\delta\ v_1\ v_2)\ z_1 & \rightarrow & \delta\ (g\ v_1\ (g\ v_2\ z_1)) \\ g\ (\alpha\ v_1)\ z_1 & \rightarrow & \alpha\ (g\ v_1\ z_1) \\ g\ (\beta\ v_1)\ z_1 & \rightarrow & \beta\ (g\ v_1\ z_1) \\ g\ \epsilon\ z_1 & \rightarrow & \epsilon\ z_1 \end{array}$$

The function is defined by recursion, which is one of the most important concepts in functional programming and means that on rule right-hand sides the function being defined can itself be called. Here every such call performs a strict descent in the first parameter, whereas the second parameter is accumulating (cf. the first rule). The above rules together with initial expression $g\ v_1\ \gamma$ constitute the generally favored solution for computing (as a monadic tree, i.e., essentially as a list) the prefix traversal of a tree built from binary symbol δ, unary symbols α and β, and nullary symbol ϵ. In fact, accumulating style is a quite common programming idiom in functional languages (cf. e.g. Chapter 6 of [FFFK01]). As a consequence, the problem of eliminating intermediate results produced inside accumulating parameters has received much attention [Küh98, Küh99, CDPR99, Voi01, KGF02a, KGF02b, Voi02c, Nis02, VK04a, Voi04b, Nis04]. All solutions proposed so far derive from restricting the class of input programs to certain extended schemes of primitive recursion, which can be formalized as (deterministic) *macro tree transducers* [Eng80].

1.3 Tree transducers and their composition

A macro tree transducer (for short *mtt*) translates trees over a ranked alphabet of input symbols into trees over a ranked alphabet of output symbols. For this translation process an mtt uses a set of functions, thus capturing the principle of mutual recursion. Every function is defined by pattern-matching on the root symbol of its first argument t. The right-hand side of every rule may contain the other function arguments, output symbols, and recursive function calls—potentially to other functions of the mtt—, the first arguments of which must be variables that refer to subtrees of t. The above program for prefix traversal is a simple example of an mtt. Since many typical functions on algebraic data types are defined by structural descent on a distinguished argument, mtts represent a large class of functional programs using accumulating parameters, which are called *context parameters* in this scope. For example, manipulation of abstract syntax trees in compilers often follows the recursion scheme of mtts [Vog91, FV98], and the "tree transformation core" of XML processing languages can be compiled into compositions of mtts [EM03a, MBPS04].

While traditionally the study of mtts (and other tree transducer models) has mainly been motivated by automata theoretic concerns, their interpretation as special functional programs has also led to practical applications. Indeed, by transforming functional programs corresponding to restricted mtts into *attribute grammars* [Knu68] or *attributed tree transducers* [Fül81], performing a composition construction on that level, and transforming the result back into a functional program, [Küh97, Küh98] and [CDPR98, CDPR99] were the first to achieve elimination of intermediate results produced inside accumulating parameters. The transformation technique consisted of a combination of constructions from [Fül81, Fra82, CF82, Gan83, GG84, Gie88] and relied on certain syntactic restrictions imposed on the original mtts, viz. *weakly single-useness* (as introduced in [Küh98]) and *context-linearity* (meaning that context parameters are never copied). In [Voi01, VK04a] a direct construction without detour to the "attribute world" was presented that is applicable to a pair of less restricted mtts. In particular, it successfully performs a composition also if the producer of the intermediate result is an arbitrary mtt, while the consumer is a *top-down tree transducer* (for short *tdtt*; [Rou70, Tha70, Eng75]), i.e. an mtt without context parameters. Thus, it captures a result from [EV85] that was used in [Küh99] as another application of tree transducer theory to accumulating parameter deforestation. In fact, except for [Nis04] (allowing a restricted use of stacks in input programs), none of the techniques proposed to date for eliminating intermediate results produced inside accumulating parameters handles a richer class of programs than the construction from [VK04a]. The latter's direct nature made it suitable for integration into an optimizing compiler [Reu03].

1.4 Formal efficiency analysis

The most important property to be established for a transformation that is performed by a compiler is of course the preservation of program semantics. But if a

transformation's purpose is to improve the efficiency of programs in some sense, this more intensional aspect needs to be considered as well. While elimination of intermediate results can reduce the amount of memory allocations and deallocations, this alone is not sufficient to be sure that the transformed program is faster—or at least not slower—than the original one. In fact, none of the deforestation techniques mentioned above provides such a guarantee when applied to arbitrary input programs. The standard result about classical deforestation is that it does not deteriorate the efficiency of linear input programs, where efficiency is measured by counting *call-by-need* [Wad71] reduction steps. Examples show that for tree transducer composition linearity of input programs is neither a sufficient nor a necessary condition for efficiency nondeterioration with respect to this measure. For the special case that one of the mtts to be composed is a tdtt sufficient conditions were developed in [Küh99] and [Höf99] using rather ad-hoc reasoning. In [Voi02b] we have studied this case using a more systematic approach, which in this thesis is extended to the full construction from [VK04a], thus providing the first formal efficiency guarantees for general tree transducer composition.[1] The criteria obtained improve on previous results and cover many typical examples.

Reasoning about the efficiency of functional programs is challenging because methods like equational reasoning, developed to study extensional aspects—i.e., what a program computes—, are not readily applicable to it. One reason is that time and space consumption can be considered as kind of side-effects because they fall out of the pure mathematical concept of a function as mapping input to output and doing nothing else. A further complication in nonstrict languages is that the number of steps performed using *call-by-name* or call-by-need reduction (the standard evaluation strategies in such languages) is not a compositional criterion, i.e., the total cost of a composite expression cannot be understood solely from the cost properties of its subexpressions in isolation. This is in contrast to the extensional semantics of functional languages, which is usually praised for its compositionality. As a consequence, the techniques used, for example, to establish the semantic correctness of the construction from [VK04a] in [Voi01, VK04b] are of no immediate use when it comes to studying this construction's impact on the efficiency of programs. Our approach, therefore, is based on externalizing the intensional property "computation time" by making it part of the observable program output, thus making it amenable to methods developed for reasoning about extensional properties. It can be summarized as follows:

– First, we enrich the original and the composed program, respectively, by annotating their rules with special symbols and adding new rules in such a way that from the output trees computed for a particular input tree one can read off the numbers of call-by-name reduction steps performed by the programs under consideration for that input.

[1] While none of the other techniques for deforesting inside accumulating parameters proposed so far [KGF02a, KGF02b, Voi02c, Nis02, Voi04b, Nis04] comes with any efficiency guarantees whatsoever, we will note in Subsection 7.1.1 that our results apply also to the indirect constructions via attributed tree transducers or attribute grammars from [Küh98] and [CDPR99].

– Then, we devise another annotation for the original program such that applying the composition construction to the thus annotated original program yields essentially the annotated version of the composed program counting reduction steps. What was a simple exercise for the special case considered in [Voi02b] requires some imagination for the general case. In fact, we consider the presented annotation (and the associated proof establishing its suitability) a key contribution because on an intuitive level it exposes the essence of the rather complicated construction from [VK04a] in a surprisingly elegant way, while on the technical side it allows to completely remove the details of the composition construction from concern in any further efficiency analysis.

– Left with annotated versions of the original program only, we can combine and manipulate annotations, trading those special symbols that correspond to steps of the composed program against those that correspond to steps of the original program. The aim is to arrive at an annotation for which simple conditions suffice to guarantee that for every input the number of symbols produced of the former kind is outweighed by that of symbols produced of the latter kind. Some of the involved manipulations will be safe—i.e., will not lead to an overestimation of the number of steps performed by the original program or an underestimation of the number of steps performed by the composed program—only under certain conditions that will also be recorded. Proofs will be modularized by "factoring out" what are the essential properties guaranteed by those additional conditions, thus identifying potential levers for further generalizing the analysis.

– Having established sufficient conditions for efficiency nondeterioration under call-by-name reduction, we can transfer the results to the call-by-need setting by giving conditions under which the costs measured under the two reduction strategies do not differ for the original program. In contrast to the situation for classical deforestation, where this is typically achieved by restricting input programs to be linear, the particular recursion scheme of mtts allows for more liberal conditions to be used here.[2]

To be of practical use in a compiler, the conditions that our analysis comes up with as guaranteeing efficiency nondeterioration through the composition construction must be effectively (and preferably efficiently) testable on every input program. We present an efficient decision procedure to this purpose and prove its correctness. Further contributions are listed in the following overview of the thesis.

[2] In [Voi02b], rather than first studying call-by-name efficiency and then transferring the results to the call-by-need setting, the same conditions were used from the outset to study only call-by-need efficiency. For expositional reasons and because it offers additional insights about call-by-name reduction, we have in this thesis chosen to separate the study as described above.

1.5 The contents of this thesis

Beside this introduction the thesis contains seven chapters and a proof appendix.

Chapter 2 defines necessary notions and introduces the basic concepts of mtts, their semantics, and important syntactic restrictions. It also discusses the proof principles to be used throughout and gives some auxiliary lemmata.

Chapter 3 presents tree transducer composition by considering the underlying ideas on an intuitive level and by giving the formal construction. The presentation follows [VK04a], though with different notation.

Chapter 4 develops our efficiency analysis along the lines discussed in the previous section. The results presented in this chapter are essentially those of [Voi02b] and [Voi04a]. Many of the proofs (as well as some from Chapters 2 and 5) are given separately in the appendix.

Chapter 5 develops and proves correctness of an efficient decision procedure for the key problem when applying our formal efficiency analysis to a particular program. A slightly less general variant of this procedure has already been implemented under our supervision (joint with Armin Kühnemann) in [Reu03] to check the conditions developed in [Voi02b] for the special case that one of the mtts to be composed is a tdtt. The correctness proof is novel and presented here for the first time.

Chapter 6 studies some further aspects that are important when moving from the theoretical framework of tree transducers to practical application of our composition construction. In particular, we argue that the transformation is semantics-preserving also in the presence of partial and infinite trees (in contrast to only for completely defined, finite trees as established by the original correctness proof in [Voi01, VK04b]), and discuss how to adjust the efficiency analysis so as to take two phenomena into account that may affect efficiency in a real functional language, namely *partial demand* and *tail calls*. None of these aspects has been treated before.

Chapter 7 puts our composition construction and the formal efficiency analysis in the context of related work and proposes directions for future research. Among others, it offers new insights about the relation of our transformation to other approaches to tree transducer composition and details on why classical and shortcut deforestation fail to effectively eliminate intermediate results inside accumulating parameters where tree transducer composition succeeds. We also consider several other (published) results with which we have contributed to the area of program transformation that could not all be accommodated in full in the frame of this thesis. In particular, we explain in some detail how *tupling* and *circular bindings* can be used to give a different account of the ideas behind tree transducer composition, leading to the alternative *lazy composition* transformation put forward

in [Voi02c, Voi04b]. We also exemplify how a transformation technique developed in [GKV03] that is based on tree transducer theory but goes the opposite way of composition by **introducing** intermediate results can improve the suitability of programs for verification, and briefly discuss our work in the field of program transformations based on free theorems [Voi02a, JV04a].

Chapter 8 concludes with some retrospective thoughts on our overall approach to call-by-need efficiency analysis and a perspective for advancing along the axis of realistic efficiency measures.

Chapter 2

Preliminaries

We denote by $\mathbb{N}$ the set of natural numbers including 0 and by $\mathbb{Z}$ the set of integers. We set $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. For every $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$. The mapping max gives for every finite subset of $\mathbb{N}$ the maximum of that subset's elements; $max(\emptyset) = 0$.

Let S be a set. We denote by S^* the set of finite sequences of elements of S, where ε denotes the empty sequence. The power set of S is denoted by $\mathcal{P}(S)$. If S is finite, then the number of its elements is denoted by $|S|$. The Cartesian product of sets $S_1, \ldots, S_n$ is denoted by $S_1 \times \cdots \times S_n$. The n-fold Cartesian product of one set S is denoted by S^n. Given a binary relation $\Rightarrow \in \mathcal{P}(S^2)$, we write $s \Rightarrow s'$ for $(s, s') \in \Rightarrow$. We denote by $\Rightarrow^n$ (with $n \in \mathbb{N}$) the n-fold composition, by $\Rightarrow^?$ the reflexive, and by $\Rightarrow^*$ the reflexive, transitive closure of $\Rightarrow$, respectively (all in $\mathcal{P}(S^2)$). If for every $s \in S$ there exists at most one $s' \in S$ with $s \Rightarrow s'$, then $\Rightarrow$ is called *deterministic*. If for every $s, s_1, s_2 \in S$ with $s \Rightarrow^* s_1$ and $s \Rightarrow^* s_2$ there exists $s' \in S$ with $s_1 \Rightarrow^* s'$ and $s_2 \Rightarrow^* s'$, then $\Rightarrow$ is called *confluent*. Determinism implies confluence. If there is no infinite chain $s_1 \Rightarrow s_2 \Rightarrow s_3 \Rightarrow \cdots$, then $\Rightarrow$ is called *terminating*. If for a given $s \in S$ there is no s' with $s \Rightarrow s'$, then s is called *irreducible* with respect to $\Rightarrow$. If $s \Rightarrow^* s'$ and s' is irreducible w.r.t. $\Rightarrow$, then s' is called a *normal form* of s w.r.t. $\Rightarrow$. If $\Rightarrow$ is confluent and terminating, then every $s \in S$ has a unique normal form, denoted by $nf(\Rightarrow, s)$.

We denote by id the identity function on every set. We use two symmetric versions of function composition, namely for every sets S_1, S_2, S_3, and functions $f : S_1 \longrightarrow S_2$ and $g : S_2 \longrightarrow S_3$ we define $f; g = g \cdot f : S_1 \longrightarrow S_3$ such that for every $s \in S_1$: $(g \cdot f)(s) = g(f(s))$. For every function $f : S \longrightarrow S$ and $n \in \mathbb{N}$ the n-fold composition of f is denoted by f^n, where $f^0 = id$. The product of functions $f_1 : S_1 \longrightarrow S_1', \ldots, f_n : S_n \longrightarrow S_n'$ is denoted by $f_1 \times \cdots \times f_n : S_1 \times \cdots \times S_n \longrightarrow S_1' \times \cdots \times S_n'$ and defined by $(f_1 \times \cdots \times f_n)(s_1, \ldots, s_n) = (f_1(s_1), \ldots, f_n(s_n))$. For two functions $f : S \longrightarrow \mathbb{Z}$ and $g : S \longrightarrow \mathbb{Z}$ we write $f \leq g$ iff $f(s) \leq g(s)$ for every $s \in S$, and we write $f - g : S \longrightarrow \mathbb{Z}$ for the function that maps every $s \in S$ to $f(s) - g(s)$.

2.1 Trees, substitutions, and rewrite systems

A *ranked alphabet* is a pair $(\Sigma, \mathit{rank}_\Sigma)$, where Σ is a finite set of symbols and rank_Σ assigns to each of these symbols a natural number, its *rank*. In the following, we drop the rank_Σ-function from the denotation and only mention Σ when referring to a ranked alphabet. For every $p \in \mathbb{N}$ we define $\Sigma^{(p)} = \{\sigma \in \Sigma \mid \mathit{rank}_\Sigma(\sigma) = p\}$. The rank p of a symbol σ is also denoted by writing $\sigma^{(p)}$. A *nullary* symbol is one of rank 0, a *unary* symbol one of rank 1, and a *binary* symbol one of rank 2. For the sake of brevity, quantifications over a symbol in a ranked alphabet implicitly quantify also its rank. For example, we write "for every $\sigma \in \Sigma^{(p)}$" instead of "for every $p \in \mathbb{N}$, $\sigma \in \Sigma^{(p)}$" and "there exists $f \in F^{(r+1)}$" instead of "there exist $r \in \mathbb{N}$ and $f \in F^{(r+1)}$". For a ranked alphabet Σ we denote the set of all its ranks as $\mathit{rank}(\Sigma) = \{p \in \mathbb{N} \mid \exists \sigma \in \Sigma.\ \mathit{rank}_\Sigma(\sigma) = p\}$.

We use several sets of lowercase variables. We denote by U the (infinite) set $\{u_1, u_2, u_3, \ldots\}$ and for every $p \in \mathbb{N}$ by U_p the finite set $\{u_1, \ldots, u_p\} \subseteq U$; analogously for V, Y, Z, and $Y' = \{y'_1, y'_2, y'_3, \ldots\}$.

For a ranked alphabet Σ and a set X of variables disjoint from Σ we define the set $T_\Sigma(X)$ of *trees over* Σ *indexed by* X as the smallest set $T \subseteq (\Sigma \cup X \cup \{(,)\})^*$ such that (i) $X \subseteq T$ and (ii) for every $\sigma \in \Sigma^{(p)}$ and $t_1, \ldots, t_p \in T$: $(\sigma\ t_1 \cdots t_p) \in T$. If readability allows, outer brackets of trees are omitted. Note that every symbol $\sigma \in \Sigma^{(p)}$ can also be considered as a function of type $T_\Sigma(X)^p \longrightarrow T_\Sigma(X)$, mapping $(t_1, \ldots, t_p)$ to $\sigma\ t_1 \cdots t_p$. We denote the set $T_\Sigma(\emptyset)$ by T_Σ.

We need the *set of paths in a tree*, the *label at a path in a tree*, the *subtree at a path in a tree*, the *size of a tree*, the *height of a tree*, and the *number of occurrences of some symbol* $s \in \Sigma \cup X$ *in a tree*, given by the functions *paths*, *lab*, *sub*, *size*, *height*, and $|\,.\,|_s$, which are defined, with $x \in X$, $\sigma \in \Sigma^{(p)}$, $i \in \mathbb{N}_+$, $\pi \in (\mathbb{N}_+)^*$, and $t, t_1, \ldots, t_p \in T_\Sigma(X)$, as follows:

$$
\begin{aligned}
&\mathit{paths} : T_\Sigma(X) \longrightarrow \mathcal{P}((\mathbb{N}_+)^*)\\
&\mathit{paths}(x) &&= \{\varepsilon\}\\
&\mathit{paths}(\sigma\ t_1 \cdots t_p) &&= \{\varepsilon\} \cup \{i\pi \mid i \in [p], \pi \in \mathit{paths}(t_i)\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{lab} : \{(t, \pi) \mid t \in T_\Sigma(X), \pi \in \mathit{paths}(t)\} \longrightarrow \Sigma \cup X\\
&\mathit{lab}(x, \varepsilon) &&= x\\
&\mathit{lab}(\sigma\ t_1 \cdots t_p, \varepsilon) &&= \sigma\\
&\mathit{lab}(\sigma\ t_1 \cdots t_p, i\pi) &&= \mathit{lab}(t_i, \pi)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{sub} : \{(t, \pi) \mid t \in T_\Sigma(X), \pi \in \mathit{paths}(t)\} \longrightarrow T_\Sigma(X)\\
&\mathit{sub}(t, \varepsilon) &&= t\\
&\mathit{sub}(\sigma\ t_1 \cdots t_p, i\pi) &&= \mathit{sub}(t_i, \pi)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{size} : T_\Sigma(X) \longrightarrow \mathbb{N}_+\\
&\mathit{size}(x) &&= 1\\
&\mathit{size}(\sigma\ t_1 \cdots t_p) &&= 1 + \sum_{i \in [p]} \mathit{size}(t_i)
\end{aligned}
$$

$$
\begin{array}{l}
\mathit{height} : T_\Sigma(X) \longrightarrow \mathbb{N}_+ \\
\mathit{height}(x) \quad\quad\quad = 1 \\
\mathit{height}(\sigma\ t_1 \cdots t_p) = 1 + \mathit{max}(\{\mathit{height}(t_i) \mid i \in [p]\})
\end{array}
$$

$$
\begin{array}{l}
|\,\cdot\,|_s : T_\Sigma(X) \longrightarrow \mathbb{N} \\
|x|_s \quad\quad\quad = 0 \quad\quad \text{if } x \neq s, \qquad |s|_s \quad\quad\quad = 1 \quad\quad \text{if } s \in X \\
|\sigma\ t_1 \cdots t_p|_s = \sum\limits_{i \in [p]} |t_i|_s \quad \text{if } \sigma \neq s, \qquad |s\ t_1 \cdots t_p|_s = 1 + \sum\limits_{i \in [p]} |t_i|_s \quad \text{if } s \in \Sigma^{(p)}.
\end{array}
$$

The label $\mathit{lab}(t, \varepsilon)$ is called the *root symbol* of the tree t.

We use two kinds of substitution over trees, written postfix and left-associative. For some $n \in \mathbb{N}$, a ranked alphabet Σ', and a set X' of variables such that $(\Sigma \cup \Sigma') \cap (X \cup X') = \emptyset$, the first-order substitution $\cdot[x_1, \ldots, x_n \leftarrow t'_1, \ldots, t'_n]$ for pairwise different variables $x_1, \ldots, x_n \in X$ and trees $t'_1, \ldots, t'_n \in T_{\Sigma'}(X')$, and the second-order substitution $\cdot[\sigma_1, \ldots, \sigma_n \twoheadleftarrow f_1, \ldots, f_n]$ for pairwise different symbols $\sigma_1 \in \Sigma^{(p_1)}, \ldots, \sigma_n \in \Sigma^{(p_n)}$ and functions $f_1 : T_{\Sigma\cup\Sigma'}(X)^{p_1} \longrightarrow T_{\Sigma\cup\Sigma'}(X), \ldots, f_n : T_{\Sigma\cup\Sigma'}(X)^{p_n} \longrightarrow T_{\Sigma\cup\Sigma'}(X)$, are defined, with $x \in X$, $\sigma \in \Sigma^{(p)}$, and $t_1, \ldots, t_{p/p_i} \in T_\Sigma(X)$, as follows:

$$
\begin{array}{rll}
\cdot[x_1, \ldots, x_n \leftarrow t'_1, \ldots, t'_n] : T_\Sigma(X) \longrightarrow T_{\Sigma\cup\Sigma'}((X \setminus \{x_1, \ldots, x_n\}) \cup X') & & \\
x[x_1, \ldots, x_n \leftarrow t'_1, \ldots, t'_n] = x & & \text{if } x \notin \{x_1, \ldots, x_n\} \\
x_i[x_1, \ldots, x_n \leftarrow t'_1, \ldots, t'_n] = t'_i & & \text{for } i \in [n] \\
(\sigma\ t_1 \cdots t_p)[x_1, \ldots, x_n \leftarrow t'_1, \ldots, t'_n] = \sigma\ & t_1[x_1, \ldots, x_n \leftarrow t'_1, \ldots, t'_n] & \\
& \cdots & \\
& t_p[x_1, \ldots, x_n \leftarrow t'_1, \ldots, t'_n] &
\end{array}
$$

$$
\begin{array}{rll}
\cdot[\sigma_1, \ldots, \sigma_n \twoheadleftarrow f_1, \ldots, f_n] : T_\Sigma(X) \longrightarrow T_{\Sigma\cup\Sigma'}(X) & & \\
x[\sigma_1, \ldots, \sigma_n \twoheadleftarrow f_1, \ldots, f_n] = x & & \\
(\sigma\ t_1 \cdots t_p)[\sigma_1, \ldots, \sigma_n \twoheadleftarrow f_1, \ldots, f_n] = \sigma\ & t_1[\sigma_1, \ldots, \sigma_n \twoheadleftarrow f_1, \ldots, f_n] & \\
& \cdots & \\
& t_p[\sigma_1, \ldots, \sigma_n \twoheadleftarrow f_1, \ldots, f_n] & \\
& & \text{if } \sigma \notin \{\sigma_1, \ldots, \sigma_n\} \\
(\sigma_i\ t_1 \cdots t_{p_i})[\sigma_1, \ldots, \sigma_n \twoheadleftarrow f_1, \ldots, f_n] = f_i(& t_1[\sigma_1, \ldots, \sigma_n \twoheadleftarrow f_1, \ldots, f_n], & \\
& \cdots, & \\
& t_{p_i}[\sigma_1, \ldots, \sigma_n \twoheadleftarrow f_1, \ldots, f_n]) & \text{for } i \in [n].
\end{array}
$$

We also use alternative notations similar to set comprehensions, e.g. $\cdot[x_i \leftarrow t'_i \mid i \in [n]]$, and appropriate multi-line notations for long substitutions. Substitution binds stronger than tree forming, i.e., "$\sigma\ t[\cdots]$" for a unary symbol σ means "$\sigma\ (t[\cdots])$" rather than "$(\sigma\ t)[\cdots]$".

A *rewrite rule* over Σ and X is a rule of the form $\mathit{lhs} \to \mathit{rhs}$ with $\mathit{lhs}, \mathit{rhs} \in T_\Sigma(X)$ such that the left-hand side lhs does not contain two occurrences of the same variable and every variable occurring in the right-hand side rhs is also contained in lhs. A set R of rewrite rules over Σ and X is called a *rewrite system* (over Σ

and X)[1] [DJ90, BN98]. For every $\Sigma' \supseteq \Sigma$ it induces a binary *reduction relation* $\Rightarrow_R \subseteq T_{\Sigma'} \times T_{\Sigma'}$ such that $t \Rightarrow_R t'$ iff R contains a rule $lhs \to rhs$, there is a tree $c \in T_{\Sigma'}(\{x\})$ (with $x \notin \Sigma' \cup X$) that contains x exactly once, and there exist $n \in \mathbb{N}, t_1, \ldots, t_n \in T_{\Sigma'}$ and pairwise different variables $x_1, \ldots, x_n \in X$ such that $t = c[x \leftarrow lhs[x_1, \ldots, x_n \leftarrow t_1, \ldots, t_n]]$ and $t' = c[x \leftarrow rhs[x_1, \ldots, x_n \leftarrow t_1, \ldots, t_n]]$.

2.2 Macro tree transducers

In this section we introduce macro tree transducers, their semantics, and syntactic restrictions. We also present the running example to be used throughout the thesis and recall a well-known lemma from the literature.

Definition 2.1 (macro tree transducer, $RHS(\ldots)$)
A *macro tree transducer* (for short *mtt*) M is a tuple $(F, \Sigma, \Delta, e, R)$ consisting of:

- a ranked alphabet F of *states*, where $F^{(0)} = \emptyset$ and $F \neq \emptyset$,
- a ranked alphabet Σ of *input symbols*, where $F \cap \Sigma = \emptyset$,
- a ranked alphabet Δ of *output symbols*, where $F \cap \Delta = \emptyset$,
- an *initial expression* $e \in RHS(F, \Delta, U_1, \emptyset)$, and
- a set R containing for every $f \in F^{(r+1)}$ and $\sigma \in \Sigma^{(p)}$ exactly one rule of the form $f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r \to rhs_{M,f,\sigma}$, where $rhs_{M,f,\sigma} \in RHS(F, \Delta, U_p, Y_r)$.

Here for variable sets X and X' the set $RHS(F, \Delta, X, X')$ is the smallest set $RHS \subseteq T_{F \cup \Delta}(X \cup X')$ satisfying the following conditions:

- $X' \subseteq RHS$,
- for every $\delta \in \Delta^{(q)}$ and $\phi_1, \ldots, \phi_q \in RHS$: $(\delta\ \phi_1 \cdots \phi_q) \in RHS$, and
- for every $f \in F^{(r+1)}$, $x \in X$, and $\phi_1, \ldots, \phi_r \in RHS$: $(f\ x\ \phi_1 \cdots \phi_r) \in RHS$. □

Note that R in the above definition is a rewrite system over $F \cup \Sigma \cup \Delta$ and $U \cup Y$. For given $\sigma \in \Sigma$, every rule in R of the form $f\ (\sigma\ \cdots)\ \cdots \to \cdots$ for some $f \in F$ is called a *σ-rule*. A subtree of the form $(f\ t\ \cdots)$ is referred to as a *call* to f on t. The first argument of a state f is called *recursion argument*; the others, *context parameters*. Correspondingly, variables from U are called *recursion variables* and variables from Y are called *context variables*. However, the actual variable names used in mtt rules are not fixed to come from U_p and Y_r for some $p, r \in \mathbb{N}$; consistent renaming is allowed. For example, we later use recursion variables from V and context variables from Z for the second mtt in the composition construction (with states in G, input symbols in Δ, and output symbols in Ω), in the way we have already done for the introductory example.

[1] When Σ and X are clear from the context, we only mention R. Note that a rewrite system over Σ and X is also a rewrite system over $\Sigma' \supseteq \Sigma$ and $X' \supseteq X$ if $\Sigma' \cap X' = \emptyset$.

Example 2.2 (two mtts, to be used as running example)
Let $\Sigma_{mon} = \{\alpha^{(1)}, \beta^{(1)}, \epsilon^{(0)}\}$, $\Delta_{tree} = \{\delta^{(2)}, \alpha^{(1)}, \beta^{(1)}, \epsilon^{(0)}\}$, and $\Omega_{mon} = \{\delta^{(1)}, \alpha^{(1)}, \beta^{(1)}, \epsilon^{(1)}, \gamma^{(0)}\}$. We define the mtts $M_{spine} = (\{f_1^{(2)}, f_2^{(3)}\}, \Sigma_{mon}, \Delta_{tree}, e_{M_{spine}}, R_{spine})$ with $e_{M_{spine}} = f_1\ u_1\ \epsilon$ and set of rules

$$
\begin{array}{lll}
R_{spine}: & f_1\ (\alpha\ u_1)\ y_1 \to \alpha\ (f_1\ u_1\ y_1) & f_2\ (\alpha\ u_1)\ y_1\ y_2 \to \alpha\ (f_1\ u_1\ (\delta\ y_1\ y_2)) \\
& f_1\ (\beta\ u_1)\ y_1 \to f_2\ u_1\ y_1\ (\beta\ \epsilon) & f_2\ (\beta\ u_1)\ y_1\ y_2 \to f_2\ u_1\ y_1\ (\beta\ (\beta\ y_2)) \\
& f_1\ \epsilon\ \ \ y_1 \to y_1 & f_2\ \epsilon\ \ \ y_1\ y_2 \to \delta\ y_1\ y_2
\end{array}
$$

and $M_{pfx} = (\{g^{(2)}\}, \Delta_{tree}, \Omega_{mon}, e_{M_{pfx}}, R_{pfx})$, where $e_{M_{pfx}} = g\ v_1\ \gamma$ and R_{pfx} contains the rules for g given in the introduction. □

The semantics of an mtt is a function from trees over the input ranked alphabet to trees over the output ranked alphabet. It is given by substituting the input tree for u_1 (or for v_1 if recursion variables are drawn from V) in the initial expression and then computing the normal form with respect to the reduction relation induced by the mtt's set of rules. This normal form exists and is unique because the rules of any mtt induce a confluent and terminating reduction relation (cf. e.g. [FV98]).

Definition 2.3 (semantics of mtts)
The *tree transduction induced by an mtt* $M = (F, \Sigma, \Delta, e, R)$ is the function $\tau_M : T_\Sigma \longrightarrow T_\Delta$ that assigns to every $t \in T_\Sigma$ the value $nf(\Rightarrow_R, e[u_1 \leftarrow t])$. Any function that is induced thus by an mtt is called a *macro tree transduction*. □

It is easy to see that during the reduction of $e[u_1 \leftarrow t]$ with $\Rightarrow_R$ only well-formed terms as defined in the following can occur.

Definition 2.4 (well-formed terms)
Let $M = (F, \Sigma, \Delta, e, R)$ be an mtt. The set WFT_M of *well-formed terms for* M is the smallest set $WFT \subseteq T_{F \cup \Delta \cup \Sigma}$ satisfying the following conditions:

- for every $f \in F^{(r+1)}$, $t \in T_\Sigma$, and $\phi_1, \ldots, \phi_r \in WFT$: $(f\ t\ \phi_1 \cdots \phi_r) \in WFT$, and
- for every $\delta \in \Delta^{(q)}$ and $\phi_1, \ldots, \phi_q \in WFT$: $(\delta\ \phi_1 \cdots \phi_q) \in WFT$. □

The tree transduction induced by the mtt M_{spine} from Example 2.2 maps every element of $T_{\Sigma_{mon}}$ to one of $T_{\Delta_{tree}}$ as depicted in Figure 2.a. While on first sight that transformation might seem a bit artificial[2], it highlights several capabilities of mtts. Firstly, there is the possibility of spatially separating the output generated for adjacent parts of the input, realized here by directly reproducing encountered α-symbols at the top of the output, whereas β-symbols are accumulated in context parameters. Secondly, there is the "finite control" aspect of using different states to remember the last input symbol read and so recognize the beginning and ending of "β-blocks". Finally, note that an mtt can reverse an order found in the input, manifested in our example in that the output produced for the top-most β-block appears lowest in the final output, and vice versa.

[2]For more practical examples see [VK04a] and also [KV01], where it was demonstrated how typical functions on polymorphic data types and some higher-order functions like *map* can be

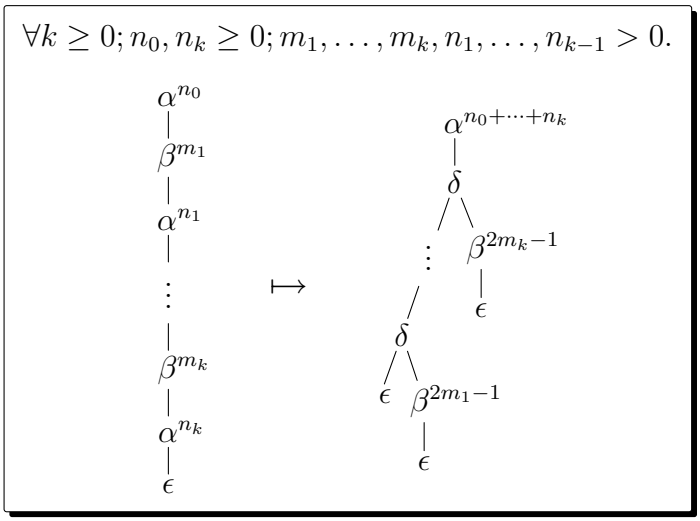

Figure 2.a: $\tau_{M_{spine}}$

Moreover, the mtts M_{spine} and M_{pfx} serve as a good running example because they seem to constitute about the simplest reasonable composition pair experiencing all the phenomena we later want to illustrate when developing our efficiency analysis. Compared to a more straightforward-looking example, it also has the advantage of requiring a quite disciplined, general approach rather than tempting us to found the analysis on too specific assumptions about the program to be transformed. In particular, M_{spine} is complex enough to defeat simplistic analysis attempts based solely on the size of the intermediate result compared to that of the original input, given that every rational number between (including) 1 and 2 occurs as linear size increase realized by $\tau_{M_{spine}}$ for infinitely many different input trees.

We introduce some syntactic restrictions of mtts that are used throughout.

Definition 2.5 (syntactic restrictions of mtts)
An mtt $M = (F, \Sigma, \Delta, e, R)$ is:

- a *top-down tree transducer* (for short *tdtt*) if $F = F^{(1)}$
- a *homomorphism tree transducer* (for short *htt*) if it is a tdtt and $|F| = 1$
- *basic* [Vog87] if e and the right-hand sides of the rules in R do not contain any nested calls, i.e. subtrees of the form $(f\ \cdots(f'\ \cdots)\cdots)$ with $f, f' \in F$
- *weakly single-use* if for every $\sigma \in \Sigma^{(p)}$, $i \in [p]$, and $f \in F \setminus F^{(1)}$ at most one call to f on u_i occurs in all right-hand sides of σ-rules in R[3]

viewed as mtts.

[3]In [Küh98] and [VK04a] this condition was required for all states, not just for nonunary ones, and an analogous condition was imposed also on the initial expression (the "right-hand side of the rule for the initial synthesized function at the root symbol" in the terminology of [Küh98]). It was, however, observed in [VK04a] that "weakly single-use" can be generalized as done here, without requiring any change to the presented construction or to its correctness proof.

– *recursion-linear* if for every $f \in F$, $\sigma \in \Sigma^{(p)}$, and $i \in [p]$: $|rhs_{M,f,\sigma}|_{u_i} \leq 1$

– *context-linear*[4] if for every $f \in F^{(r+1)}$, $\sigma \in \Sigma$, and $k \in [r]$: $|rhs_{M,f,\sigma}|_{y_k} \leq 1$

– *linear* if it is recursion- and context-linear

– *atmost* if it is recursion-linear and it is context-linear or basic

– *recursion-nondeleting* if for every $f \in F$, $\sigma \in \Sigma^{(p)}$, and $i \in [p]$: $|rhs_{M,f,\sigma}|_{u_i} \geq 1$

– *context-nondeleting* if for every $f \in F^{(r+1)}$, $\sigma \in \Sigma$, and $k \in [r]$: $|rhs_{M,f,\sigma}|_{y_k} \geq 1$

– *nondeleting* if it is recursion- and context-nondeleting

– *atleast* if it is recursion-nondeleting and it is context-nondeleting or basic

– *producing* if for every $f \in F$ and $\sigma \in \Sigma$ it exists $\delta \in \Delta$ with $|rhs_{M,f,\sigma}|_\delta \geq 1$. □

Note that every tdtt is basic, weakly single-use, and context-linear and -nondeleting. Further note that atmostness generalizes linearity, while atleastness generalizes nondeletingness. For tdtts the restrictions recursion-linear, linear, and atmost coincide, as do recursion-nondeleting, nondeleting, and atleast.

Example 2.6 (syntactic restrictions checked for the mtts from Ex. 2.2)
The mtt M_{spine} is basic, recursion- and context-linear, and recursion- and context-nondeleting. It is thus also linear, atmost, nondeleting, and atleast, but it is not a tdtt, not an htt, not weakly single-use (due to two calls each to f_1 and f_2 on u_1 in α- and β-rules, respectively), and not producing (due to the rule for f_1 at ϵ). The mtt M_{pfx} fulfills all the properties from Definition 2.5, except for being a tdtt, an htt, or basic. □

The following lemma is well known. It is a consequence of Theorem 3.24 in [EV85].

Lemma 2.7 (exponential height-height bound for mtts)
Let $M = (F, \Sigma, \Delta, e, R)$ be an mtt. There is a constant $c \in \mathbb{N}$ such that for every $t \in T_\Sigma$: $height(\tau_M(t)) \leq c^{height(t)}$. □

2.3 About proofs in this thesis

The most important proof principle to be used in this thesis is *induction*. The well-known principle of induction over natural numbers—namely that a statement is true for every natural number if its validity for $n \in \mathbb{N}$ can be proved from its validity for every $m \in \mathbb{N}$ with $m < n$—can be transferred to statements about trees over ranked alphabets. We will use the following special variant.

[4] This restriction was called *non-copying* in [VK04a].

Proof principle (*simultaneous induction*; cf. e.g. [EV85, KV94, FV98])
Let Σ be a ranked alphabet. For given $t \in T_\Sigma$, a subtree $sub(t, \pi)$ with $\pi \in paths(t)$ is called *proper* if $\pi \neq \varepsilon$ and *direct* if $\pi \in \mathbb{N}_+$. Let (a) and (b) be statements, called *induction hypotheses*, where (a) has a free variable t of type T_Σ and (b) has free variables p of type $\mathbb{N}$ and $t_1, \ldots, t_p$ of type T_Σ. If

(a) $\Leftarrow$ (b) : for every $t \in T_\Sigma$ such that (a) holds for every proper subtree of t and (b) holds for every $p \in \mathbb{N}$ and direct subtrees $t_1, \ldots, t_p$ of t, we can prove that (a) holds, and

(b) $\Leftarrow$ (a) : for every $p \in \mathbb{N}$ and $t_1, \ldots, t_p \in T_\Sigma$ such that (a) holds for each of the $t_1, \ldots, t_p$, we can prove that (b) holds,

then (a) holds for every $t \in T_\Sigma$ and (b) holds for every $p \in \mathbb{N}$ and $t_1, \ldots, t_p \in T_\Sigma$. □

Note the somewhat unusual reference to (a) from within the *induction step* **(a) $\Leftarrow$ (b)**, which gives more—and sometimes needed—freedom in proofs. Since it refers only to proper subtrees of the one under consideration, the soundness of the proof principle can still be argued along similar lines as in Section 2.5 of [EV85]. Further note that ordinary *structural induction* over trees is a special case of our form of simultaneous induction, with (b) being trivially true. Simultaneous and structural induction will also be used for trees with a more refined structure, in particular for elements of $RHS(\cdots)$-sets (cf. Definition 2.1) and elements of $WFT_{M_1;M_2}$ (cf. Definition 4.2). Where other (than simultaneous or structural) induction methods are used in proofs, we will explicitly mention what the induction is over. In function definitions, however, we will sometimes use induction implicitly, as we have already done for *paths* and other functions in Section 2.1. In all these cases, straightforward proofs could be given showing that the functions in question are really well-defined.

Many proofs can be found in the appendix rather than in the main part of the thesis. Where these are not proofs that consist only of plugging definitions and auxiliary lemmata together in a fairly obvious way, we have made some effort to provide intuitive explanations in the main part by pointing to the key observations used inside induction steps. Often the induction hypotheses already reveal all the inventiveness needed for the proof. Indeed, some straightforward inductions are omitted for exactly this reason. Inside the inductions that are given explicitly we have usually chosen such a granularity of proof steps that every single step is relatively basic, so that proof-checking should be a cheap (indeed potentially mechanizable) operation. In particular, generalization of induction hypotheses was used to a large extent to avoid the need for complicated "normal form laws" as necessary, e.g., in the proofs in [Voi01, VK04b]. One recurring operation is manipulation of sum formulae. In such proof steps we use the standard distributivity, associativity, and commutativity properties, as well as common index operations.

To see some of the tactics abstractly discussed above in action, consider the proof of the following lemma in the appendix.

Lemma 2.8 (auxiliary)
Let $M = (F, \Sigma, \Delta, e, R)$ be an mtt. For every $t \in T_\Sigma$, $f \in F^{(r'+1)}$, $r \in \mathbb{N}$, $k \in [r]$, and $\theta_1, \ldots, \theta_{r'} \in T_{F \cup \Sigma \cup \Delta}(Y_r)$:[5]

$$|nf(\Rightarrow_R, f\ t\ \theta_1 \cdots \theta_{r'})|_{y_k} = \sum_{k' \in [r']} |nf(\Rightarrow_R, f\ t\ y_1 \cdots y_{r'})|_{y_{k'}} * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}. \quad \square$$

The following lemma will also be used extensively throughout the thesis.

Lemma 2.9 (properties of context-linear and -nondeleting mtts, resp.)
Let $M = (F, \Sigma, \Delta, e, R)$ be an mtt. For every $t \in T_\Sigma$, $f \in F^{(r+1)}$, and $k \in [r]$:

1. if M is context-linear, then $|nf(\Rightarrow_R, f\ t\ y_1 \cdots y_r)|_{y_k} \leq 1$
2. if M is context-nondeleting, then $|nf(\Rightarrow_R, f\ t\ y_1 \cdots y_r)|_{y_k} \geq 1$. $\square$

The proof (by simultaneous induction over the structure of elements of T_Σ, using a nested induction over the structure of right-hand sides of rules of M, as well as Lemma 2.8) can be found in the appendix. Actually, we only need to explicitly prove point 1 of the lemma; the proof of point 2 is completely dual in the sense in which the definition of "context-nondeleting" is dual to that of "context-linear" by requiring $|rhs_{M,f,\sigma}|_{y_k} \geq 1$ rather than $|rhs_{M,f,\sigma}|_{y_k} \leq 1$ for every $f \in F^{(r+1)}$, $\sigma \in \Sigma$, and $k \in [r]$.

[5] To minimize the need for brackets when writing sum formulae, we will use the conventions that $*$ binds stronger than a $\sum$ to the left of it, while $\sum$ binds stronger than $+$ or binary $-$, so that, e.g., an expression of the form $\sum a{*}b{+}c{*}\sum d{*}e{-}\sum -f$ is parsed as $\big(\sum(a{*}b)\big){+}c{*}\big(\sum(d{*}e)\big){-}\big(\sum(-f)\big)$.

Chapter 3

Tree transducer composition

In [Voi01, VK04a] a direct composition construction is presented that given two mtts M_1 and M_2, where the output ranked alphabet of M_1 is the input ranked alphabet of M_2 and where one of M_1 and M_2 is a tdtt or M_1 is context-linear and M_2 is weakly single-use, produces an mtt $\overline{M_1;M_2}$ such that $\tau_{\overline{M_1;M_2}} = \tau_{M_1} ; \tau_{M_2}$. Before recalling the formal construction (with adapted notations), we briefly consider its ingredients on an intuitive level. For a more detailed step-by-step development of the underlying ideas, consult Section 4 of [VK04a]. The brave-hearted may also want to study the formal correctness proof supplied in [VK04b].

3.1 The basic ideas

The key to avoiding the production and consumption of an intermediate result is that whenever a state g of M_2 would recurse on the output generated by a state f of M_1, the final output is instead computed more directly through a call to a new state $\overline{fg}$[1] of $\overline{M_1;M_2}$ on f's recursion argument. This means that every nesting of the form

$$g\ (f\ t\ \phi_1 \cdots \phi_r)\ \eta_1 \cdots \eta_s \tag{1}$$

is replaced by an appropriate call to $\overline{fg}$ on t. Given that the (intermediate) output generated by f is essentially built up by patching together right-hand sides of M_1's rules at the symbols occurring in t, the rules for such a new state $\overline{fg}$ are obtained by "translating" the right-hand sides of rules for f using M_2, starting with g. More precisely, for every potential root symbol σ of t, the right-hand side $rhs_{\overline{M_1;M_2},\overline{fg},\sigma}$ is determined from $g\ rhs_{M_1,f,\sigma}\ z_1 \cdots z_s$—where $z_1 \cdots z_s$ are the context variables of $\overline{fg}$ that correspond to the context parameters of the outer g-call in (1)—by exhaustively rewriting with the rules of M_2. Thus, the symbols in $rhs_{M_1,f,\sigma}$ that would otherwise build part of the intermediate data structure passed at runtime from the inner f-call to the outer g-call in (1) are consumed at transformation time. Since $rhs_{M_1,f,\sigma}$ may contain recursive calls, one may during the rewriting also encounter situations corresponding to a nesting like (1) above (possibly for some $f' \neq f$ and $g' \neq g$).

[1] Here $\overline{fg}$ is used as a compacter notation for the pair (f, g).

To ensure that all parts of $rhs_{M_1,f,\sigma}$—including those in context parameter positions of recursive calls—are effectively reached by the transformation, any call to a new state replacing such a nesting takes as arguments beside the unchanged context parameters of the outer state from M_2 also "translated" versions of the context parameters of the inner state from M_1 with all states of M_2. Under this strategy, the replacement for a call like (1) takes the following form (assuming that M_2 has exactly $\mu \in \mathbb{N}$ states $g_1, \ldots, g_\mu$):

$$\overline{fg}\ t\ (g_1\ \phi_1\ \underline{\boxed{\cdots}})\ \cdots\ (g_\mu\ \phi_1\ \underline{\boxed{\cdots}})\ \cdots\ (g_1\ \phi_r\ \underline{\boxed{\cdots}})\ \cdots\ (g_\mu\ \phi_r\ \underline{\boxed{\cdots}})\ \eta_1 \cdots \eta_s \quad (2)$$

Since $\overline{fg}$ is thus provided with precomputed translations of f's context parameters, any call to some state of M_2 on some context variable of f surfacing during the processing of $g\ rhs_{M_1,f,\sigma}\ z_1 \cdots z_s$ as above can be resolved by simply selecting the corresponding parameter of $\overline{fg}$, modeled by a rewrite system *Pre* in the formal construction.

Of course, the crux in this strategy of sending the states of M_2 into the context parameters of calls to M_1's states is to determine what should go into the boxes in the replacement (2) for (1) above. Here the preconditions on M_1 and M_2 enter the picture. If one of the two mtts is actually a tdtt, then no information is missing because either $r = 0$ (if M_1 is a tdtt) or none of the $g_1, \ldots, g_\mu$ takes any context parameters (if M_2 is a tdtt). Otherwise, context-linearity of M_1 and weakly single-useness of M_2 ensure that the context parameter values with which the states $g_1, \ldots, g_\mu$ may reach the context parameters of the inner f-call during reduction of (1) are unambiguous. This is so because a context-linear M_1 will not copy any of the context parameters during the computation of $f\ t\ \phi_1 \cdots \phi_r$ and because for a weakly single-use M_2 it is guaranteed that no position in the resulting intermediate tree can be reached by some state with two different sequences of context parameters during the computation started with g and $\eta_1, \ldots, \eta_s$. Moreover, these unambiguous values providing the information missing in (2) can actually be computed without leaving the recursion scheme of mtts. To this aim, a new state $\overline{k_f l_{g'}}$ [2] is introduced for every $k \in [r]$, $g' \in \{g_1, \ldots, g_\mu\}$, and context parameter position l of g'. The intended semantics of those auxiliary states is illustrated in Figure 3.a. Their rules are constructed following an intricate scheme of "walking upwards" from occurrences of context variables in M_1's right-hand sides, briefly considered next.

The basic idea is that the lth context parameter of some (nonunary) state g on reaching the jth subtree of some δ-symbol in an intermediate tree is essentially determined by the lth context parameter of the unique (if existing) call to g on v_j in the δ-rules of the weakly single-use M_2, assuming knowledge of the context parameters, on reaching the position of the δ-symbol under consideration, of the state in whose rule at δ the call to g on v_j occurs. Since instead of a concrete intermediate tree we only have access to the right-hand sides of M_1's rules from which it is going to be patched together, and since these right-hand sides may contain recursive calls, we may also need to determine the lth context parameter of some g on reaching a

[2] Here $\overline{k_f l_{g'}}$ is used as a compacter notation for the pair of pairs $((k, f), (l, g'))$.

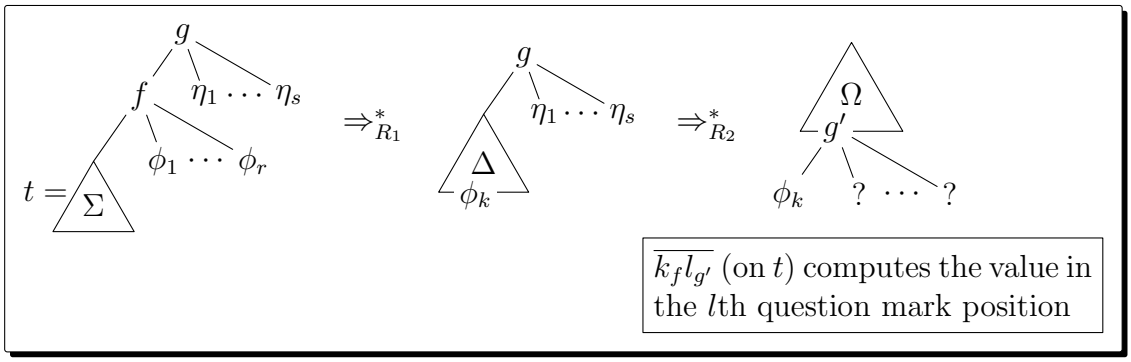

Figure 3.a: Role of the auxiliary states

context parameter of some call to $f \in F$ on a recursion variable. Unsurprisingly, this will be resolved via a (recursive) call to the appropriate auxiliary state, somewhat analogously to the situation during the construction of $rhs_{\overline{M_1;M_2},\overline{fg},\sigma}$ by translation of $rhs_{M_1,f,\sigma}$ with g, where a nesting like (1) for some f' and g' gives rise to a recursive call to $\overline{f'g'}$. The precise description of the "walking upwards" can be found in the formal construction in the next section in the definition of the *par*-functions, using $nest_f$-functions to arrange a repeated nesting of $g_1, \ldots, g_\mu$-translations of f's context parameters interspersed with appropriate calls to the auxiliary states.

Another use of $nest_f$ is in the ultimate completion of the replacement (2) for (1), specified through a rewrite system *Pair*. The reasons why a nesting of translations is in general necessary but can be kept of finite depth are discussed at length in [VK04a]. There it is also explained which context parameters the auxiliary states require and why some of them can be filled with a dummy symbol *nil* in *Pair*. Beside Section 4 of [VK04a] also Section 5.3 of that paper is of interest here because it demonstrates that a seemingly simpler approach constructing auxiliary states with reduced ranks does not work out.

3.2 The formal construction

The following construction differs from Construction 5.1 in [VK04a] only in that the required dummy symbol is explicitly added to the output ranked alphabet rather than randomly picking an existing nullary symbol.

Construction 3.1 (direct composition of restricted mtts)
Let $M_1 = (F, \Sigma, \Delta, e_{M_1}, R_1)$ and $M_2 = (G, \Delta, \Omega, e_{M_2}, R_2)$ be mtts such that one of the two is a tdtt or M_1 is context-linear and M_2 is weakly single-use. Without loss of generality, assume that F and G are disjoint and that M_1 uses recursion variables from U and context variables from Y, whereas M_2 uses V and Z, respectively. Let $\mu = |G|$ and fix some ordering of the states in G such that $G = \{g_1, \ldots, g_\mu\}$. For every $n \in [\mu]$ let $s_n \in \mathbb{N}$ be such that $g_n \in G^{(s_n+1)}$. Additionally, set $r_{max} =$

$max(rank(F)) - 1$, $Z_G = \{z_{g_1,1}, \ldots, z_{g_\mu,s_\mu}\}$ [3], and let $nil \notin \Omega$ be some arbitrary symbol. The mtt $\overline{M_1;M_2} = (H, \Sigma, \Omega \cup \{nil^{(0)}\}, e_{\overline{M_1;M_2}}, \overline{R_1;R_2})$ is obtained as follows:

$$\begin{aligned} H = \quad & \{\overline{fg}^{(r*\mu+s+1)} \mid f \in F^{(r+1)}, g \in G^{(s+1)}\} \\ \cup\ & \{\overline{k_f l_g}^{(r*\mu+|Z_G|+1)} \mid f \in F^{(r+1)}, g \in G^{(s+1)}, k \in [r], l \in [s]\}\,, \end{aligned}$$

$e_{\overline{M_1;M_2}} = nf(\Rightarrow_{R_2 \cup Pair}, e_{M_2}[v_1 \leftarrow e_{M_1}])$, and

$\overline{R_1;R_2}$ contains

- for every $f \in F^{(r+1)}$, $g \in G^{(s+1)}$, and $\sigma \in \Sigma^{(p)}$ the rule

  $$\overline{fg}\ (\sigma\ u_1 \cdots u_p)\ y_{1,g_1} \cdots y_{r,g_\mu}\ z_1 \cdots z_s \to nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, g\ rhs_{M_1,f,\sigma}\ z_1 \cdots z_s)$$ [4]

  and
- for every $f \in F^{(r+1)}$, $g \in G^{(s+1)}$, $k \in [r]$, $l \in [s]$, and $\sigma \in \Sigma^{(p)}$ the rule

  $$\overline{k_f l_g}\ (\sigma\ u_1 \cdots u_p)\ y_{1,g_1} \cdots y_{r,g_\mu}\ z_{g_1,1} \cdots z_{g_\mu,s_\mu} \to nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, \varrho)\,,$$

  where $\varrho = nil$ if $rhs_{M_1,f,\sigma}$ does not contain y_k;
  otherwise, $\varrho = par_{M_2, rhs_{M_1,f,\sigma}}(\pi, g, l)$, where $\pi \in paths(rhs_{M_1,f,\sigma})$ is the path of the unique occurrence of y_k in $rhs_{M_1,f,\sigma}$. [5]

The rewrite systems Pre (over $G \cup Y_{r_{max}} \cup \{y^{(0)}_{k,g} \mid k \in [r_{max}], g \in G\}$ and Z) and $Pair$ (over $F \cup G \cup H \cup \{nil\}$ and $\{u\} \cup Y' \cup Z$) are defined as follows:

Pre : $g\ y_k\ z_1 \cdots z_s$	$\to y_{k,g}$	$\forall g \in G^{(s+1)}, k \in [r_{max}]$
$Pair$: $g\ (f\ u\ y'_1 \cdots y'_r)\ z_1 \cdots z_s \to$		$\forall g \in G^{(s+1)}, f \in F^{(r+1)}$
$\overline{fg}\ u\ nest_f(1, g_1, \emptyset)[z_{g,l} \leftarrow z_l \mid l \in [s],$		
$z_{g',l} \leftarrow nil \mid g' \in G^{(s'+1)} \setminus \{g\}, l \in [s']]$		
$\cdots$		
$nest_f(r, g_\mu, \emptyset)[z_{g,l} \leftarrow z_l \mid l \in [s],$		
$z_{g',l} \leftarrow nil \mid g' \in G^{(s'+1)} \setminus \{g\}, l \in [s']]$		
$z_1 \cdots z_s$		

For every $f \in F^{(r+1)}$ the function

$$nest_f : [r] \times G \times \mathcal{P}([r] \times G) \longrightarrow T_{G \cup H \cup \{nil\}}(\{u\} \cup Y'_r \cup Z_G)$$

[3] The enumeration $z_{g_1,1}, \ldots, z_{g_\mu,s_\mu}$ resolves to $z_{g_1,1}, \ldots, z_{g_1,s_1}, \ldots, z_{g_\mu,1}, \ldots, z_{g_\mu,s_\mu}$, and correspondingly for other enumerations doubly indexed from $(g_1, 1)$ to (g_μ, s_μ) below.

[4] The enumeration $y_{1,g_1} \cdots y_{r,g_\mu}$ resolves to $y_{1,g_1} \cdots y_{1,g_\mu} \cdots y_{r,g_1} \cdots y_{r,g_\mu}$, and correspondingly for other enumerations doubly indexed from $(1, g_1)$ to (r, g_μ), or from $(1, g_1)$ to (r', g_μ), below.

[5] Note that if the ranges of the above quantifications of f, g, k, and l are nonempty, then neither M_1 nor M_2 can be a tdtt and hence M_1 is context-linear by the construction's preconditions.

is defined by setting for every $k \in [r]$, $g' \in G^{(s'+1)}$, and $\mathcal{C} \subseteq [r] \times G$:

$$
\begin{aligned}
&nest_f(k, g', \mathcal{C}) \\
&= \begin{cases}
g'\ y'_k\ (\overline{k_f 1_{g'}}\ u\ nest_f(1, g_1, \mathcal{C} \cup \{(k,g')\})\ \cdots\ nest_f(r, g_\mu, \mathcal{C} \cup \{(k,g')\}) \\
\qquad\qquad z_{g_1,1} \cdots z_{g_\mu,s_\mu}) \\
\qquad \cdots \\
\qquad (\overline{k_f s'_{g'}}\ u\ nest_f(1, g_1, \mathcal{C} \cup \{(k,g')\})\ \cdots\ nest_f(r, g_\mu, \mathcal{C} \cup \{(k,g')\}) \\
\qquad\qquad z_{g_1,1} \cdots z_{g_\mu,s_\mu}) & \text{if } (k,g') \notin \mathcal{C} \\
nil & \text{otherwise.}
\end{cases}
\end{aligned}
$$

The combined reduction relations $\Rightarrow_{R_2 \cup Pair}$ and $\Rightarrow_{R_2 \cup Pre \cup Pair}$ are terminating. Since there are no *critical pairs* [DJ90, BN98] in $R_2 \cup Pre \cup Pair$, they are also confluent. Hence, the above use of normal forms with respect to them is justified.

We additionally define for every $p \in \mathbb{N}$, $r \in \mathbb{N}_+$ with $(r+1) \in rank(F)$, and $\phi \in RHS(F, \Delta, U_p, Y_r)$ the function

$$
\begin{aligned}
par_{M_2,\phi} : \{(\pi, g, l) \mid \pi \in paths(\phi), lab(\phi, \pi) \notin U_p, g \in G^{(s+1)}, l \in [s]\} \\
\longrightarrow T_{F \cup G \cup H \cup \Delta \cup \Omega \cup \{nil\}}(U_p \cup Y_r \cup Z_G)
\end{aligned}
$$

by induction over the prefix-order of paths in ϕ as follows. For every $g \in G^{(s+1)}$ and $l \in [s]$:

- $par_{M_2,\phi}(\varepsilon, g, l) = z_{g,l}$
- For every $j \in \mathbb{N}_+$ and $\pi j \in paths(\phi)$ with $lab(\phi, \pi j) \notin U_p$ we perform a case distinction on $lab(\phi, \pi)$, which must be in $\Delta \cup F$ because by $\pi j \in paths(\phi)$ it cannot be a "leaf" of ϕ.

  $\underline{lab(\phi, \pi) = \delta}$ for some $\delta \in \Delta^{(q)}$, where $j \in [q]$:
  If any of the δ-rules of M_2 contains a call to g on v_j as follows:

  $$g'\ (\delta\ v_1 \cdots v_q)\ z_1 \cdots z_{s'} \to \cdots (g\ v_j\ \psi_1 \cdots \psi_s) \cdots,$$

  where $g' \in G^{(s'+1)}$ and $\psi_1, \ldots, \psi_s \in RHS(G, \Omega, V_q, Z_{s'})$, then[6]

  $$
  \begin{aligned}
  par_{M_2,\phi}(\pi j, g, l) = \psi_l[v_1, \ldots, v_q \leftarrow sub(\phi, \pi 1), \ldots, sub(\phi, \pi q), \\
  z_1, \ldots, z_{s'} \leftarrow par_{M_2,\phi}(\pi, g', 1), \ldots, par_{M_2,\phi}(\pi, g', s')].
  \end{aligned}
  $$

  If no such call exists in the δ-rules of M_2, then $par_{M_2,\phi}(\pi j, g, l) = nil$.

[6]Note that if the ranges of the above quantifications of r, g, and l are nonempty, then neither M_1 nor M_2 can be a tdtt and hence M_2 is weakly single-use by the construction's preconditions. Moreover, $g \in G^{(s+1)}$ and $l \in [s]$ imply that g is a nonunary state, i.e., the weakly single-use condition applies to g-calls, so that g' and the $\psi_1, \ldots, \psi_s$ are uniquely determined.

$\underline{lab(\phi,\pi) = f}$ for some $f \in F^{(r'+1)}$, where $j-1 \in [r']$:

$$
\begin{aligned}
&par_{M_2,\phi}(\pi j, g, l) = \\
&\qquad \overline{((j-1)_f l_g}\ u\ nest_f(1, g_1, \{(j-1, g)\}) \\
&\qquad\qquad \cdots \\
&\qquad\qquad nest_f(r', g_\mu, \{(j-1, g)\}) \\
&\qquad\qquad z_{g_1,1} \cdots z_{g_\mu,s_\mu}) \\
&\qquad [u \leftarrow lab(\phi, \pi 1), \\
&\qquad\ y'_1, \ldots, y'_{r'} \leftarrow sub(\phi, \pi 2), \ldots, sub(\phi, \pi(r'+1)), \\
&\qquad\ z_{g_1,1}, \ldots, z_{g_\mu,s_\mu} \leftarrow par_{M_2,\phi}(\pi, g_1, 1), \ldots, par_{M_2,\phi}(\pi, g_\mu, s_\mu)]. \quad \square
\end{aligned}
$$

The following theorem was proved in [Voi01]; an even more detailed account can be found in [VK04b].

Theorem 3.2 (correctness of Construction 3.1; Theorem 5.2 of [VK04a]) $\tau_{M_1} ; \tau_{M_2} = \tau_{\overline{M_1;M_2}}$ $\square$

To provide some additional intuition about the construction, we observe the exact roles that the states of the resulting mtt play by describing the output they compute in terms of computations using the combined reduction relation $\Rightarrow_{R_1 \cup R_2}$ induced by the original mtts[7]. The observation follows by combining statements **I**a and **I**b of Lemma A.15, Definition A.6(1), and Theorem A.4 in [VK04b].

Observation 3.3 ("extensional semantics" for the states of $\overline{M_1;M_2}$) For every $f \in F^{(r+1)}$, $g \in G^{(s+1)}$, and $t \in T_\Sigma$:

$$
\begin{aligned}
&nf(\Rightarrow_{\overline{R_1;R_2}}, \overline{f g}\ t\ y_{1,g_1} \cdots y_{r,g_\mu}\ z_1 \cdots z_s) \\
&= nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_1 \cup R_2}, g\ (f\ t\ y_1 \cdots y_r)\ z_1 \cdots z_s)).
\end{aligned}
$$

Moreover, for every $k \in [r]$ and $g' \in G^{(s'+1)}$, if $nf(\Rightarrow_{R_1 \cup R_2}, g\ (f\ t\ y_1 \cdots y_r)\ z_1 \cdots z_s)$ contains a subtree of the form $(g'\ y_k\ \varrho_1 \cdots \varrho_{s'})$, then

$$
\begin{aligned}
&nf(\Rightarrow_{\overline{R_1;R_2}}, \overline{k_f l_{g'}}\ t\ y_{1,g_1} \cdots y_{r,g_\mu}\ z_{g_1,1} \cdots z_{g_\mu,s_\mu}) \\
&= nf(\Rightarrow_{Pre}, \varrho_l)[z_1, \ldots, z_s \leftarrow z_{g,1}, \ldots, z_{g,s}]
\end{aligned}
$$

for every $l \in [s']$. $\square$

3.3 Application to the running example

For the sake of illustration, we apply Construction 3.1 to the mtts M_{spine} and M_{pfx} serving as running example. We only show the construction of two rules for the composed mtt, chosen such that every constituent of the construction comes to bear; the other rules and the initial expression are obtained similarly. First, we give one rule each from *Pre* and from *Pair*. Since $r_{max} = 2$, the former contains the following rule:

[7] Since F and G are assumed to be disjoint, $\Rightarrow_{R_1 \cup R_2}$ is confluent. It is also terminating.

$$g\ y_1\ z_1 \to y_{1,g}$$

To obtain the *Pair*-rule for g and f_2, we do some calculation using the $nest_{f_2}$-function:

$$\begin{aligned}
g\ (f_2\ u\ y_1'\ y_2')\ z_1 &\to \overline{f_2 g}\ u\ nest_{f_2}(1,g,\emptyset)[z_{g,1} \leftarrow z_1]\ nest_{f_2}(2,g,\emptyset)[z_{g,1} \leftarrow z_1]\ z_1 \\
&= \overline{f_2 g}\ u\ (g\ y_1'\ (\overline{1_{f_2} 1_g}\ u\ nest_{f_2}(1,g,\{(1,g)\}) \\
&\qquad\qquad nest_{f_2}(2,g,\{(1,g)\})\ z_{g,1}))[z_{g,1} \leftarrow z_1] \\
&\qquad (g\ y_2'\ (\overline{2_{f_2} 1_g}\ u\ nest_{f_2}(1,g,\{(2,g)\}) \\
&\qquad\qquad nest_{f_2}(2,g,\{(2,g)\})\ z_{g,1}))[z_{g,1} \leftarrow z_1]\ z_1 \\
&= \overline{f_2 g}\ u\ (g\ y_1'\ (\overline{1_{f_2} 1_g}\ u\ nil\ (g\ y_2'\ (\overline{2_{f_2} 1_g}\ u\ nest_{f_2}(1,g,\{(1,g),(2,g)\}) \\
&\qquad\qquad\qquad nest_{f_2}(2,g,\{(1,g),(2,g)\}) \\
&\qquad\qquad\qquad z_{g,1}))\ z_{g,1}))[z_{g,1} \leftarrow z_1] \\
&\qquad (g\ y_2'\ (\overline{2_{f_2} 1_g}\ u\ (g\ y_1'\ (\overline{1_{f_2} 1_g}\ u\ nest_{f_2}(1,g,\{(2,g),(1,g)\}) \\
&\qquad\qquad\qquad nest_{f_2}(2,g,\{(2,g),(1,g)\}) \\
&\qquad\qquad\qquad z_{g,1}))\ nil\ z_{g,1}))[z_{g,1} \leftarrow z_1]\ z_1 \\
&= \overline{f_2 g}\ u\ (g\ y_1'\ (\overline{1_{f_2} 1_g}\ u\ nil\ (g\ y_2'\ (\overline{2_{f_2} 1_g}\ u\ nil\ nil\ z_1))\ z_1)) \\
&\qquad (g\ y_2'\ (\overline{2_{f_2} 1_g}\ u\ (g\ y_1'\ (\overline{1_{f_2} 1_g}\ u\ nil\ nil\ z_1))\ nil\ z_1))\ z_1
\end{aligned}$$

Using the two rules above, as well as two rules from R_{pfx}, we construct the rule for $\overline{f_1 g}$ at β as follows:

$$\begin{aligned}
\overline{f_1 g}\ (\beta\ u_1)\ y_{1,g}\ z_1 &\to nf(\Rightarrow_{R_{pfx} \cup Pre \cup Pair}, g\ rhs_{M_{spine},f_1,\beta}\ z_1) \\
&= nf(\Rightarrow_{R_{pfx} \cup Pre \cup Pair}, g\ (f_2\ u_1\ y_1\ (\beta\ \epsilon))\ z_1) \\
&= nf(\Rightarrow_{R_{pfx} \cup Pre \cup Pair}, \overline{f_2 g}\ u_1\ (g\ y_1 \cdots) \\
&\qquad\qquad (g\ (\beta\ \epsilon)\ (\overline{2_{f_2} 1_g}\ u_1\ (g\ y_1 \cdots)\ nil\ z_1))\ z_1) \\
&= \overline{f_2 g}\ u_1\ y_{1,g}\ (\beta\ (\epsilon\ (\overline{2_{f_2} 1_g}\ u_1\ y_{1,g}\ nil\ z_1)))\ z_1
\end{aligned}$$

Taking into account that $rhs_{M_{spine},f_1,\alpha} = \alpha\ (f_1\ u_1\ y_1)$ contains y_1 at path 12, we construct the rule for $\overline{1_{f_1} 1_g}$ at α as follows, using the definitions of $par_{M_{pfx},\alpha\ (f_1\ u_1\ y_1)}$ and $nest_{f_1}$, along with the fact that the rule $g\ (\alpha\ v_1)\ z_1 \to \alpha\ (g\ v_1\ z_1)$ is contained in R_{pfx}:

$$\begin{aligned}
\overline{1_{f_1} 1_g}\ (\alpha\ u_1)\ y_{1,g}\ z_{g,1} &\to nf(\Rightarrow_{R_{pfx} \cup Pre \cup Pair}, par_{M_{pfx},\alpha\ (f_1\ u_1\ y_1)}(12,g,1)) \\
&= nf(\Rightarrow_{R_{pfx} \cup Pre \cup Pair}, \\
&\qquad (\overline{1_{f_1} 1_g}\ u\ nest_{f_1}(1,g,\{(1,g)\})\ z_{g,1}) \\
&\qquad\quad [u, y_1', z_{g,1} \leftarrow u_1, y_1, par_{M_{pfx},\alpha\ (f_1\ u_1\ y_1)}(1,g,1)]) \\
&= nf(\Rightarrow_{R_{pfx} \cup Pre \cup Pair}, \overline{1_{f_1} 1_g}\ u_1\ nil\ par_{M_{pfx},\alpha\ (f_1\ u_1\ y_1)}(1,g,1)) \\
&= nf(\Rightarrow_{R_{pfx} \cup Pre \cup Pair}, \\
&\qquad \overline{1_{f_1} 1_g}\ u_1\ nil\ z_1[v_1 \leftarrow f_1\ u_1\ y_1, \\
&\qquad\qquad z_1 \leftarrow par_{M_{pfx},\alpha\ (f_1\ u_1\ y_1)}(\varepsilon,g,1)]) \\
&= \overline{1_{f_1} 1_g}\ u_1\ nil\ z_{g,1}
\end{aligned}$$

The complete outcome is as follows.

Example 3.4 (composition result for the mtts from Example 2.2)
Composition of the mtts M_{spine} and M_{pfx} yields the mtt $\overline{M_{spine};M_{pfx}} = (\{\overline{f_1 g}^{(3)}, \overline{f_2 g}^{(4)}, \overline{1_{f_1} 1_g}^{(3)}, \overline{1_{f_2} 1_g}^{(4)}, \overline{2_{f_2} 1_g}^{(4)}\}, \Sigma_{mon}, \Omega_{mon} \cup \{nil\}, e_{\overline{M_{spine};M_{pfx}}}, \overline{R_{spine};R_{pfx}})$ with rules

$$
\begin{array}{lllll}
\overline{R_{spine};R_{pfx}}: & \overline{f_1 g} & (\alpha\ u_1)\ y_{1,g}\ z_1 & \to \alpha\ (\overline{f_1 g}\ u_1\ y_{1,g}\ z_1) \\
& \overline{f_1 g} & (\beta\ u_1)\ y_{1,g}\ z_1 & \to \overline{f_2 g}\ u_1\ y_{1,g}\ (\beta\ (\epsilon\ (\overline{2_{f_2} 1_g}\ u_1\ y_{1,g}\ nil\ z_1)))\ z_1 \\
& \overline{f_1 g} & \epsilon\ \ y_{1,g}\ z_1 & \to y_{1,g} \\
& \overline{f_2 g} & (\alpha\ u_1)\ y_{1,g}\ y_{2,g}\ z_1 & \to \alpha\ (\overline{f_1 g}\ u_1\ (\delta\ y_{1,g})\ z_1) \\
& \overline{f_2 g} & (\beta\ u_1)\ y_{1,g}\ y_{2,g}\ z_1 & \to \overline{f_2 g}\ u_1\ y_{1,g}\ (\beta\ (\beta\ y_{2,g}))\ z_1 \\
& \overline{f_2 g} & \epsilon\ \ y_{1,g}\ y_{2,g}\ z_1 & \to \delta\ y_{1,g} \\
& \overline{1_{f_1} 1_g} & (\alpha\ u_1)\ y_{1,g}\ z_{g,1} & \to \overline{1_{f_1} 1_g}\ u_1\ nil\ z_{g,1} \\
& \overline{1_{f_1} 1_g} & (\beta\ u_1)\ y_{1,g}\ z_{g,1} & \to \overline{1_{f_2} 1_g}\ u_1\ nil \\
& & & \qquad (\beta\ (\epsilon\ (\overline{2_{f_2} 1_g}\ u_1\ nil\ nil\ z_{g,1})))\ z_{g,1} \\
& \overline{1_{f_1} 1_g} & \epsilon\ \ y_{1,g}\ z_{g,1} & \to z_{g,1} \\
& \overline{1_{f_2} 1_g} & (\alpha\ u_1)\ y_{1,g}\ y_{2,g}\ z_{g,1} & \to y_{2,g} \\
& \overline{1_{f_2} 1_g} & (\beta\ u_1)\ y_{1,g}\ y_{2,g}\ z_{g,1} & \to \overline{1_{f_2} 1_g}\ u_1\ nil\ (\beta\ (\beta\ y_{2,g}))\ z_{g,1} \\
& \overline{1_{f_2} 1_g} & \epsilon\ \ y_{1,g}\ y_{2,g}\ z_{g,1} & \to y_{2,g} \\
& \overline{2_{f_2} 1_g} & (\alpha\ u_1)\ y_{1,g}\ y_{2,g}\ z_{g,1} & \to \overline{1_{f_1} 1_g}\ u_1\ nil\ z_{g,1} \\
& \overline{2_{f_2} 1_g} & (\beta\ u_1)\ y_{1,g}\ y_{2,g}\ z_{g,1} & \to \overline{2_{f_2} 1_g}\ u_1\ y_{1,g}\ nil\ z_{g,1} \\
& \overline{2_{f_2} 1_g} & \epsilon\ \ y_{1,g}\ y_{2,g}\ z_{g,1} & \to z_{g,1}
\end{array}
$$

and initial expression $e_{\overline{M_{spine};M_{pfx}}} = \overline{f_1 g}\ u_1\ (\epsilon\ (\overline{1_{f_1} 1_g}\ u_1\ nil\ \gamma))\ \gamma$. $\square$

The mtt obtained above is unnecessarily complicated because—for the sake of generality—the composition construction tends to introduce context parameters that are superfluous in the sense that they will never (for no possible input tree) influence the output generated by a state. Here this is the case for parameter z_1 of $\overline{f_1 g}$, parameters $y_{2,g}$ and z_1 of $\overline{f_2 g}$, parameter $y_{1,g}$ of $\overline{1_{f_1} 1_g}$, parameters $y_{1,g}$ and $z_{g,1}$ of $\overline{1_{f_2} 1_g}$, and parameters $y_{1,g}$ and $y_{2,g}$ of $\overline{2_{f_2} 1_g}$. In a practical implementation such as [Reu03] the mtt resulting from a composition will be post-processed to eliminate such ballast. Such a post-processing, detecting and removing all superfluous context parameters, was developed in Section 4.1 of [Voi01] based on a straightforward fixpoint construction. For $\overline{M_{spine};M_{pfx}}$ this yields an mtt $\overline{M_{spine};M_{pfx}}'$ with states of reduced ranks, initial expression $\overline{f_1 g}\ u_1\ (\epsilon\ (\overline{1_{f_1} 1_g}\ u_1\ \gamma))$, and rules

$$
\begin{array}{lllll}
\overline{R_{spine};R_{pfx}}': & \overline{f_1 g} & (\alpha\ u_1)\ y_{1,g} & \to \alpha\ (\overline{f_1 g}\ u_1\ y_{1,g}) \\
& \overline{f_1 g} & (\beta\ u_1)\ y_{1,g} & \to \overline{f_2 g}\ u_1\ y_{1,g} \\
& \overline{f_1 g} & \epsilon\ \ y_{1,g} & \to y_{1,g} \\
& \overline{f_2 g} & (\alpha\ u_1)\ y_{1,g} & \to \alpha\ (\overline{f_1 g}\ u_1\ (\delta\ y_{1,g})) \\
& \overline{f_2 g} & (\beta\ u_1)\ y_{1,g} & \to \overline{f_2 g}\ u_1\ y_{1,g} \\
& \overline{f_2 g} & \epsilon\ \ y_{1,g} & \to \delta\ y_{1,g}
\end{array}
$$

$$
\begin{array}{lll}
\overline{1_{f_1}1_g}\ (\alpha\ u_1)\ z_{g,1} & \to & \overline{1_{f_1}1_g}\ u_1\ z_{g,1} \\
\overline{1_{f_1}1_g}\ (\beta\ u_1)\ z_{g,1} & \to & \overline{1_{f_2}1_g}\ u_1\ (\beta\ (\epsilon\ (\overline{2_{f_2}1_g}\ u_1\ z_{g,1}))) \\
\overline{1_{f_1}1_g}\ \epsilon\ z_{g,1} & \to & z_{g,1}
\end{array}
$$

$$
\begin{array}{lll}
\overline{1_{f_2}1_g}\ (\alpha\ u_1)\ y_{2,g} & \to & y_{2,g} \\
\overline{1_{f_2}1_g}\ (\beta\ u_1)\ y_{2,g} & \to & \overline{1_{f_2}1_g}\ u_1\ (\beta\ (\beta\ y_{2,g})) \\
\overline{1_{f_2}1_g}\ \epsilon\ y_{2,g} & \to & y_{2,g}
\end{array}
$$

$$
\begin{array}{lll}
\overline{2_{f_2}1_g}\ (\alpha\ u_1)\ z_{g,1} & \to & \overline{1_{f_1}1_g}\ u_1\ z_{g,1} \\
\overline{2_{f_2}1_g}\ (\beta\ u_1)\ z_{g,1} & \to & \overline{2_{f_2}1_g}\ u_1\ z_{g,1} \\
\overline{2_{f_2}1_g}\ \epsilon\ z_{g,1} & \to & z_{g,1}
\end{array}
$$

Figure 3.b shows a computation of this simplified mtt for a particular input. Since the elimination of superfluous context parameters has no effect with respect to the abstract efficiency measures we are going to use, we need not pay special attention to the post-processing in the efficiency analysis and will instead study the immediate outcome of the composition construction itself.

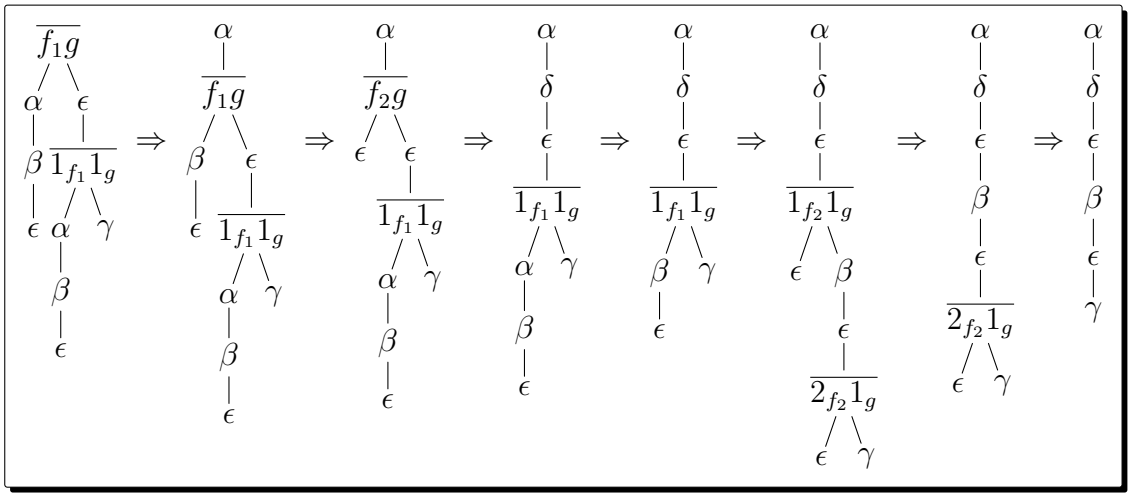

Figure 3.b: Reduction sequence using $\Rightarrow_{\overline{R_{spine};R_{pfx}}'}$

Chapter 4

Formal efficiency analysis

We want to formally relate the efficiency of a composed program obtained by Construction 3.1 to the efficiency of the original program. That is, we want to compare the cost of a computation starting from an instantiation of $e_{\overline{M_1;M_2}}$ and using rules from $\overline{R_1;R_2}$ to the cost of the computation starting from the corresponding instantiation of $e_{M_2}[v_1 \leftarrow e_{M_1}]$ and using rules from $R_1 \cup R_2$.

From the construction it is obvious that intermediate data structures (over the ranked alphabet Δ, which is not present anymore in $\overline{M_1;M_2}$) produced by M_1 in the original program do not appear in the composed program. Thus, no memory cells for this intermediate result have to be allocated in the heap and later be deallocated by the garbage collector. To make sure that this beneficial effect on **space** usage is not in vain, the composed program should not take more **time** to compute its output than the original program. A reasonable notion to consider for the "cost" of a computation is therefore the number of reduction steps required to reach the final output for a given input. While the reduction relation used to provide the semantics of an mtt in Definition 2.3 is in general nondeterministic (though confluent), implementations of functional languages are typically based on deterministic reduction strategies. Since we are interested in applying tree transducer composition to nonstrict languages, we study *call-by-name* (also known as *normal order* or *leftmost-outermost*) reduction. Later, in Section 4.10, we will provide simple conditions on M_1 and M_2 under which the gained insights can also be applied to the setting of *call-by-need* reduction.

Even though the pure number of reduction steps is a somewhat coarse efficiency measure, abstracting from the fact that not every step necessarily takes the same amount of time in a concrete implementation, it is usually sufficient in practice. For example, classical deforestation is successfully applied in a production compiler for the lazy functional language Clean [AGS03], despite the fact that the only known formal efficiency statement regarding this transformation is that it does not increase the number of call-by-need reduction steps for linear programs (the number of call-by-name steps, respectively, for arbitrary programs [San96a]). Other studies of time efficiency of functional programs also use the pure number of reduction steps as the measure of choice (see related work in Section 7.5, and see [LG01] for a notable

exception). In Section 6.3 we will consider efficiency under a refined measure that takes the special role of *tail calls* into account.

4.1 Call-by-name reduction

In this section we define call-by-name reduction relations for the composed and for the original program, respectively. We show that they are adequate restrictions of the corresponding nondeterministic reduction relations and use them to define cost functions as base for our analysis.

For the remainder of this chapter (and for Chapters 5 and 6), let $M_1 = (F, \Sigma, \Delta, e_{M_1}, R_1)$ and $M_2 = (G, \Delta, \Omega, e_{M_2}, R_2)$ be two fixed mtts to which Construction 3.1 is applicable, yielding the mtt $\overline{M_1;M_2} = (H, \Sigma, \Omega \cup \{nil\}, e_{\overline{M_1;M_2}}, \overline{R_1;R_2})$. We use the notions and variable naming conventions from Construction 3.1. Special unary symbols $\diamond$, $\bullet$, $\circ$, and $\star$, assumed not to occur in $\Sigma \cup \Delta \cup \Omega \cup F \cup G$, will be used in annotated versions of the mtts under consideration.

In Definition 2.4 we have described the set of terms that can occur for a given mtt during a computation starting from an instantiation of its initial expression. The deterministic reduction relation for the composed mtt $\overline{M_1;M_2}$ needs to be defined only for such well-formed terms.

Definition 4.1 (call-by-name restriction of $\Rightarrow_{\overline{R_1;R_2}}$)
The *call-by-name reduction relation induced by* $\overline{R_1;R_2}$, denoted by $\overset{cbn}{\Rightarrow}_{\overline{R_1;R_2}}$, is the smallest relation $\Rightarrow \subseteq WFT^2_{\overline{M_1;M_2}}$ satisfying the following conditions:

– for every $h \in H^{(n+1)}$, $\sigma \in \Sigma^{(p)}$, $t_1, \ldots, t_p \in T_\Sigma$, and $\varrho_1, \ldots, \varrho_n \in WFT_{\overline{M_1;M_2}}$:

$$h\ (\sigma\ t_1 \cdots t_p)\ \varrho_1 \cdots \varrho_n \Rightarrow rhs_{\overline{M_1;M_2},h,\sigma}[u_1, \ldots, u_p \leftarrow t_1, \ldots, t_p,\ x_1, \ldots, x_n \leftarrow \varrho_1, \ldots, \varrho_n],$$

where $x_1, \ldots, x_n$ are the context variables (in correct order) from the left-hand side of the rule for h at σ in $\overline{R_1;R_2}$, and

– for every $\omega \in \Omega^{(n)}$, $b \in [n]$, $\varrho_1, \ldots, \varrho_{b-1} \in T_{\Omega \cup \{nil\}}$, $\varrho_b \Rightarrow \varrho'_b$, and $\varrho_{b+1}, \ldots, \varrho_n \in WFT_{\overline{M_1;M_2}}$:

$$\omega\ \varrho_1 \cdots \varrho_n \Rightarrow \omega\ \varrho_1 \cdots \varrho_{b-1}\ \varrho'_b\ \varrho_{b+1} \cdots \varrho_n. \quad \square$$

The reduction relation defined is unmistakably call-by-name because no reduction takes place under function calls. That it is in fact deterministic and also sound and complete in the sense of computing the same normal forms as $\Rightarrow_{\overline{R_1;R_2}}$ is established below in Lemma 4.3. An example call-by-name computation for the mtt $\overline{M_{spine};M_{pfx}}$ would look essentially like in Figure 3.b, apart from some additional clutter due to the superfluous context parameters.

To define the deterministic reduction relation for the original program consisting of the mtts M_1 and M_2, it is useful to first describe the kind of terms that can occur during a computation starting from $e_{M_2}[v_1 \leftarrow e_{M_1}[u_1 \leftarrow t]]$ for some $t \in T_\Sigma$.

Definition 4.2 ($WFT_{M_1;M_2}$, call-by-name restriction of $\Rightarrow_{R_1 \cup R_2}$)
The set $WFT_{M_1;M_2}$ of *well-formed terms for the original program* is the smallest set $WFT \subseteq T_{G \cup \Omega \cup F \cup \Delta \cup \Sigma}$ satisfying the following conditions:

- for every $g \in G^{(s+1)}$, $\phi \in WFT_{M_1}$, $\varrho_1, \ldots, \varrho_s \in WFT$: $(g\ \phi\ \varrho_1 \cdots \varrho_s) \in WFT$, and

- for every $\omega \in \Omega^{(n)}$ and $\varrho_1, \ldots, \varrho_n \in WFT$: $(\omega\ \varrho_1 \cdots \varrho_n) \in WFT$.

The *call-by-name reduction relation induced by R_1 and R_2*, denoted by $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$, is the smallest relation $\Rightarrow \subseteq WFT^2_{M_1;M_2}$ satisfying the following conditions:

- for every $g \in G^{(s+1)}$, $f \in F^{(r+1)}$, $\sigma \in \Sigma^{(p)}$, $t_1, \ldots, t_p \in T_\Sigma$, $\phi_1, \ldots, \phi_r \in WFT_{M_1}$, and $\varrho_1, \ldots, \varrho_s \in WFT_{M_1;M_2}$:

$$\begin{aligned} & g\ (f\ (\sigma\ t_1 \cdots t_p)\ \phi_1 \cdots \phi_r)\ \varrho_1 \cdots \varrho_s \\ \Rightarrow\ & g\ rhs_{M_1,f,\sigma}[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \phi_1, \ldots, \phi_r]\ \varrho_1 \cdots \varrho_s\,, \end{aligned}$$

- for every $g \in G^{(s+1)}$, $\delta \in \Delta^{(q)}$, $\phi_1, \ldots, \phi_q \in WFT_{M_1}$, and $\varrho_1, \ldots, \varrho_s \in WFT_{M_1;M_2}$:

$$g\ (\delta\ \phi_1 \cdots \phi_q)\ \varrho_1 \cdots \varrho_s \Rightarrow rhs_{M_2,g,\delta}[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \phi_1, \ldots, \phi_q, \varrho_1, \ldots, \varrho_s]\,,$$

  and

- for every $\omega \in \Omega^{(n)}$, $b \in [n]$, $\varrho_1, \ldots, \varrho_{b-1} \in T_\Omega$, $\varrho_b \Rightarrow \varrho'_b$, and $\varrho_{b+1}, \ldots, \varrho_n \in WFT_{M_1;M_2}$:

$$\omega\ \varrho_1 \cdots \varrho_n \Rightarrow \omega\ \varrho_1 \cdots \varrho_{b-1}\ \varrho'_b\ \varrho_{b+1} \cdots \varrho_n. \quad \square$$

The reduction relation defined is call-by-name because reduction under a function call takes place only when absolutely required to proceed with the outer call. Figure 4.a shows an example for the mtts M_{spine} and M_{pfx} with the same input as in Figure 3.b. The shaded areas correspond to memory cells that have to be allocated for the intermediate data structure. Note that no such need arose during the reduction using the transformed program in Figure 3.b (only cells for the final output needed to be allocated there, which also happens here), which moreover was shorter by one step. The challenge in our efficiency analysis will be to make general statements about how the lengths of such reduction sequences for the original and the composed program compare to each other, without knowing the actual input tree provided.

But first we should convince ourselves that $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ and $\overset{cbn}{\Rightarrow}_{\overline{R_1;R_2}}$ are adequate restrictions of $\Rightarrow_{R_1 \cup R_2}$ and $\Rightarrow_{\overline{R_1;R_2}}$, respectively. Taking into account that for every $t \in T_\Sigma$, $e_{M_2}[v_1 \leftarrow e_{M_1}[u_1 \leftarrow t]] \in WFT_{M_1;M_2}$ and $e_{\overline{M_1;M_2}}[u_1 \leftarrow t] \in WFT_{\overline{M_1;M_2}}$, this is established by the following lemma.

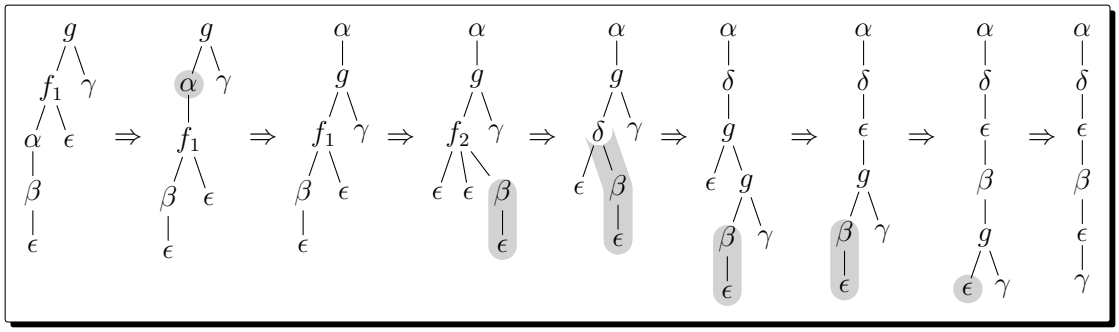

Figure 4.a: Reduction sequence using $\overset{cbn}{\Rightarrow}_{R_{spine} \cup R_{pfx}}$

Lemma 4.3 (adequacy of $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ and $\overset{cbn}{\Rightarrow}_{\overline{R_1;R_2}}$ on well-formed terms)

1. $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ is a deterministic subrelation of $\Rightarrow_{R_1 \cup R_2}$.

2. For every $\varrho \in WFT_{M_1;M_2}$: $nf(\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}, \varrho) = nf(\Rightarrow_{R_1 \cup R_2}, \varrho)$.

3. $\overset{cbn}{\Rightarrow}_{\overline{R_1;R_2}}$ is a deterministic subrelation of $\Rightarrow_{\overline{R_1;R_2}}$.

4. For every $\varrho \in WFT_{\overline{M_1;M_2}}$: $nf(\overset{cbn}{\Rightarrow}_{\overline{R_1;R_2}}, \varrho) = nf(\Rightarrow_{\overline{R_1;R_2}}, \varrho)$.

Proof

1. By straightforward induction over the recursive structure of the definition of $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ one can show that $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2} \subseteq \Rightarrow_{R_1 \cup R_2}$. This inclusion implies that elements of T_Ω are irreducible w.r.t. $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$. The latter implies that for every $\omega\ \varrho_1 \cdots \varrho_n$ there can be at most one $b \in [n]$ such that $\varrho_1, \ldots, \varrho_{b-1} \in T_\Omega$ and $\varrho_b \overset{cbn}{\Rightarrow}_{R_1 \cup R_2} \varrho'_b$, which ensures determinism of $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$.

2. Point 1 implies that $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ is confluent and terminating, thus $nf(\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}, \varrho)$ exists for every $\varrho \in WFT_{M_1;M_2}$. Induction over the structure of elements of $WFT_{M_1;M_2}$ shows that the only such elements which are irreducible w.r.t. $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ are in T_Ω and hence are also irreducible w.r.t. $\Rightarrow_{R_1 \cup R_2}$. The equality of the normal forms follows from this by $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2} \subseteq \Rightarrow_{R_1 \cup R_2}$.

The proof of points 3 and 4 is analogous. □

Since $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ is deterministic, for every $t \in T_\Sigma$ there is exactly one sequence

$$e_{M_2}[v_1 \leftarrow e_{M_1}[u_1 \leftarrow t]] \overset{cbn}{\Rightarrow}_{R_1 \cup R_2} \cdots \overset{cbn}{\Rightarrow}_{R_1 \cup R_2} nf(\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}, e_{M_2}[v_1 \leftarrow e_{M_1}[u_1 \leftarrow t]])\,,$$

and likewise for $\overset{cbn}{\Rightarrow}_{\overline{R_1;R_2}}$ and $e_{\overline{M_1;M_2}}[u_1 \leftarrow t]$. The functions $cbn_{M_1;M_2} : T_\Sigma \longrightarrow \mathbb{N}$ and $cbn_{\overline{M_1;M_2}} : T_\Sigma \longrightarrow \mathbb{N}$ associating to every input tree the number of call-by-name steps required by the original and the composed program, respectively, to compute

the corresponding output tree, are then given as follows:

$$cbn_{M_1;M_2}(t) = c \text{ if } e_{M_2}[v_1 \leftarrow e_{M_1}[u_1 \leftarrow t]] \left(\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}\right)^c nf(\Rightarrow_{R_1 \cup R_2}, e_{M_2}[v_1 \leftarrow e_{M_1}[u_1 \leftarrow t]]) = \tau_{M_2}(\tau_{M_1}(t)) = \tau_{\overline{M_1;M_2}}(t)$$

$$cbn_{\overline{M_1;M_2}}(t) = c' \text{ if } e_{\overline{M_1;M_2}}[u_1 \leftarrow t] \left(\overset{cbn}{\Rightarrow}_{\overline{R_1;R_2}}\right)^{c'} nf(\Rightarrow_{\overline{R_1;R_2}}, e_{\overline{M_1;M_2}}[u_1 \leftarrow t]).$$

We aim to find (decidable) conditions on M_1 and M_2 under which for every $t \in T_\Sigma$ we have $cbn_{\overline{M_1;M_2}}(t) \leq cbn_{M_1;M_2}(t)$. Actually, it would also be satisfactory to find conditions under which there is a some small constant $d \in \mathbb{N}$ such that for every possible input tree the following holds:

$$cbn_{\overline{M_1;M_2}}(t) \leq cbn_{M_1;M_2}(t) + d.$$

This would mean that, whatever the input is, the composed program performs at most d additional reduction steps but avoids the allocation and eventual deallocation of an intermediate data structure of arbitrary size.

4.2 Outline of the analysis

Since required computation time is an intensional property of a program, i.e., it cannot be determined by solely observing which output is generated for which input according to the program's (extensional) semantics, methods developed to reason about extensional properties of programs are not readily applicable to study it. For example, the correctness proof for our composition construction in [VK04b] makes heavy use of reordering of normal form computations, justified by the fact that the reduction relation induced by the rules of any mtt is confluent. But since the efficiency analysis we are aiming at here is concerned with a particular fixed reduction strategy, any offhand use of confluence is denied us now. A seemingly naïve—but often quite effective, see related work in Subsection 7.5.1—strategy in such a situation is to externalize the intensional property, e.g. by transforming or annotating programs, thus making it more accessible.

To concretize how our approach as outlined in the introduction is put into practice, Figure 4.b shows all proof dependencies for the various lemmata proved on the way to the main theorem about call-by-name efficiency. To allow a tracing of the flow of potential "imprecision" along these dependencies, lemmata providing inequational rather than equational statements are marked with $\leq$. In addition, shading is used to indicate a grouping of lemmata and theorems according to their nature or object of study. More specifically, the two lemmata in group (i) show that certain annotations of the original and the composed program, respectively,

can be used to reflect the numbers of performed call-by-name reduction steps in output trees. The members of group (ii) study the composition construction itself and provide a way to remove its working details from our concern for the remainder of the analysis. The lemmata in group (iii) establish how annotations of the original program can be combined and manipulated in a safe manner. In particular, a certain rewrite system $\mathcal{E}_{M_1,M_2}$ will be used arbitrarily often on annotated versions of the first mtt's rule right-hand sides and initial expression. That manipulation process can be stopped when we have reached an annotation which is simple enough to establish an input-independent upper bound for the cost count; corresponding conditions are contained in Lemma 4.39. Throughout the analysis, essential use is made of certain statements about context-linear, context-nondeleting, atmost, and atleast mtts, respectively, and about $\mathcal{E}_{M_1,M_2}$. These statements are established by the lemmata in group (iv). The remaining (unshaded) lemmata are of an auxiliary nature. The figure also lists the three definitions required—in addition to the material presented so far—to grasp the content of Theorem 4.41 right away, in case the reader wants to jump to it immediately. Going the long way through the successive steps leading to Theorem 4.41 is rewarded by Theorem 4.20, encountered halfway, which guarantees call-by-name efficiency nondeterioration by the composition construction for the case that one of the involved mtts is a tdtt. In Section 4.10 we combine Theorems 4.20 and 4.41 with conditions under which there is no difference between the numbers of call-by-name and call-by-need steps required by the original program. This leads to Theorem 4.45, containing conditions for efficiency nondeterioration with respect to call-by-need reduction. Additionally, we will recall a theorem from [Voi02b] establishing call-by-need efficiency nondeterioration for further instances of the special case that one of the mtts to be composed is a tdtt.

4.3 Annotating programs to reflect computational costs

The natural way to keep track of reduction steps is by generating a dedicated "tick"-symbol whenever some rewrite rule is applied, e.g. by replacing every rule

$$f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r \to rhs$$

by a corresponding one with an additional symbol on top of its right-hand side:

$$f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r \to \diamond\ rhs.$$

The dedicated symbol then appears scattered over the output tree generated by computing with such annotated rules. Compared to an attempt of gathering all the information about performed reduction steps in one place, e.g. at the root of the complete output or as an additional function result, this has the advantage that the origin of computational costs is remembered. In particular, discarding a piece of output produced inside an unused parameter position also removes from

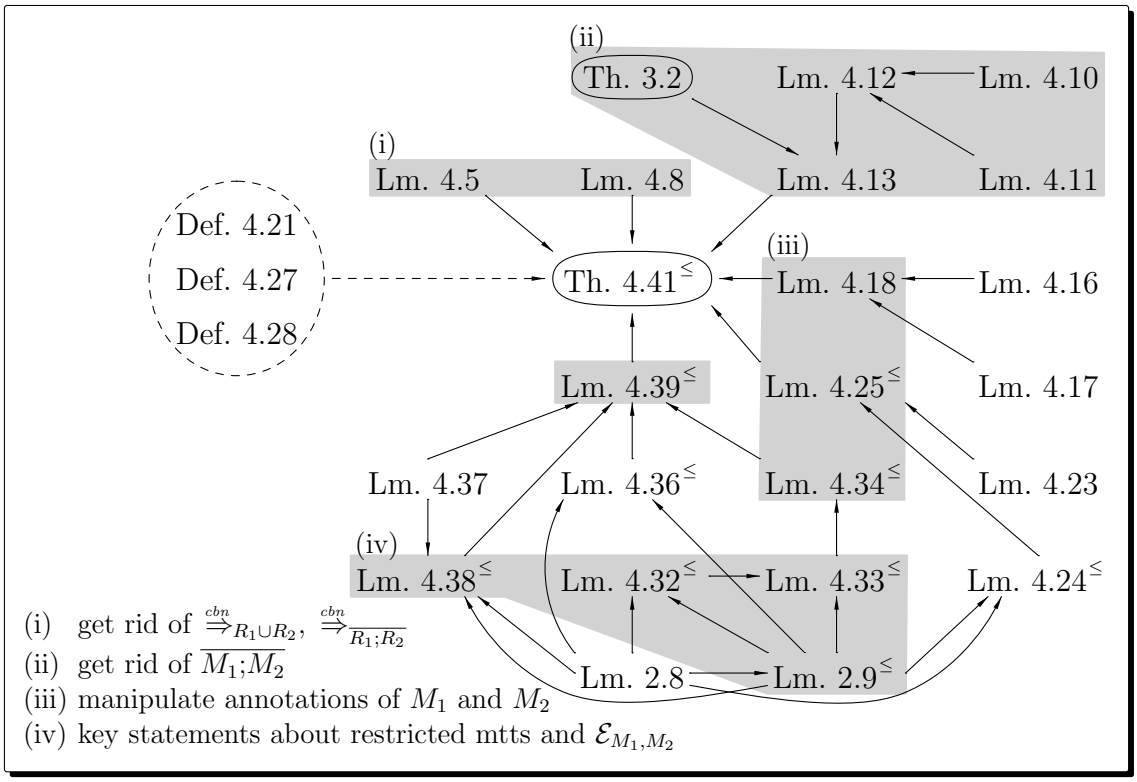

Figure 4.b: Dependencies for Theorem 4.41

consideration any computational costs associated with it. Moreover, the locational information associated with the positions of $\diamond$-symbols is important if the output produced by the annotated rules above is not yet the final output. Recall that in the original program—computing with $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ on an instantiation of $e_{M_2}[v_1 \leftarrow e_{M_1}]$—the rules R_1 are used to produce an intermediate result then consumed by states of M_2. The latter must therefore not only record their own reduction steps properly in the final output, but also propagate information about the computational costs of M_1 for producing the intermediate result. By simply including in their count any $\diamond$-symbols they come across, they can count exactly those reduction steps of M_1 that produce parts of the intermediate result actually demanded during computation of the final output. The following definition gives annotated versions of M_1 and M_2 along these lines. Lemma 4.5 then establishes that the composition of the tree transductions induced by these annotated mtts produces for every input as many $\bullet$-symbols as the original program performs call-by-name steps.

Definition 4.4 ($M_1^{\rightarrow\diamond}$ and $M_2^{\diamond\rightarrow\bullet}$)
The mtts $M_1^{\rightarrow\diamond} = (F, \Sigma, \Delta \cup \{\diamond\}, e_{M_1}, R_1^{\rightarrow\diamond})$ and $M_2^{\diamond\rightarrow\bullet} = (G, \Delta \cup \{\diamond\}, \Omega \cup \{\bullet\}, e_{M_2}, R_2^{\diamond\rightarrow\bullet})$ have rules as follows:

$R_1^{\rightarrow\diamond}$:
$f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r \rightarrow \diamond\ rhs_{M_1,f,\sigma}$ $\qquad \forall f \in F^{(r+1)}, \sigma \in \Sigma^{(p)}$

$R_2^{\diamond\to\bullet}$:		
$g\ (\delta\ v_1 \cdots v_q)$	$z_1 \cdots z_s \to \bullet\ rhs_{M_2,g,\delta}$	$\forall g \in G^{(s+1)}, \delta \in \Delta^{(q)}$
$g\ (\diamond\ v_1)$	$z_1 \cdots z_s \to \bullet\ (g\ v_1\ z_1 \cdots z_s)$	$\forall g \in G^{(s+1)}$

Note that the initial expressions remain unchanged. □

Lemma 4.5 ($M_1^{\to\diamond}$ and $M_2^{\diamond\to\bullet}$ count the call-by-name steps of M_1 and M_2) $cbn_{M_1;M_2} = \tau_{M_1^{\to\diamond}} ; \tau_{M_2^{\diamond\to\bullet}} ; |\,.\,|_\bullet$

Proof

We prove (below) by structural induction on $\varrho \in WFT_{M_1;M_2}$ that for every $\overline{\varrho} \in WFT_{M_1;M_2}$ with $\varrho \overset{cbn}{\Rightarrow}_{R_1 \cup R_2} \overline{\varrho}$ it holds $|nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\bullet}}, \varrho)|_\bullet = 1 + |nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\bullet}}, \overline{\varrho})|_\bullet$. Taking into account that for every $\varrho \left(\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}\right)^0 \varrho'$ it holds $\varrho = \varrho'$ and that for every $c \in \mathbb{N}$ and $\varrho \left(\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}\right)^{c+1} \varrho'$ there is a $\overline{\varrho} \in WFT_{M_1;M_2}$ such that $\varrho \overset{cbn}{\Rightarrow}_{R_1 \cup R_2} \overline{\varrho}$ and $\overline{\varrho} \left(\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}\right)^c \varrho'$, this implies by a straightforward induction on $c \in \mathbb{N}$ that for every $\varrho \left(\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}\right)^c \varrho'$ it holds $|nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\bullet}}, \varrho)|_\bullet = c + |nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\bullet}}, \varrho')|_\bullet$. The lemma follows because for every $t \in T_\Sigma$ we can then from

$$e_{M_2}[v_1 \leftarrow e_{M_1}[u_1 \leftarrow t]] \left(\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}\right)^{cbn_{M_1;M_2}(t)} \tau_{M_2}(\tau_{M_1}(t))$$

conclude that

$$\begin{aligned} &|nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\bullet}}, e_{M_2}[v_1 \leftarrow e_{M_1}[u_1 \leftarrow t]])|_\bullet \\ &= cbn_{M_1;M_2}(t) + |nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\bullet}}, \tau_{M_2}(\tau_{M_1}(t)))|_\bullet \end{aligned}$$

and because $|nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\bullet}}, \tau_{M_2}(\tau_{M_1}(t)))|_\bullet = |\tau_{M_2}(\tau_{M_1}(t))|_\bullet = 0$ (given that $\tau_{M_2}(\tau_{M_1}(t)) \in T_\Omega$), while $nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\bullet}}, e_{M_2}[v_1 \leftarrow e_{M_1}[u_1 \leftarrow t]])$ is equal to $\tau_{M_2^{\diamond\to\bullet}}(\tau_{M_1^{\to\diamond}}(t))$.

$\underline{\varrho = g\ (f\ (\sigma\ t_1 \cdots t_p)\ \phi_1 \cdots \phi_r)\ \varrho_1 \cdots \varrho_s}$ for some $g \in G^{(s+1)}$, $f \in F^{(r+1)}$, $\sigma \in \Sigma^{(p)}$, $t_1, \ldots, t_p \in T_\Sigma$, $\phi_1, \ldots, \phi_r \in WFT_{M_1}$, and $\varrho_1, \ldots, \varrho_s \in WFT_{M_1;M_2}$:
By the precondition $g\ (f\ (\sigma\ t_1 \cdots t_p)\ \phi_1 \cdots \phi_r)\ \varrho_1 \cdots \varrho_s \overset{cbn}{\Rightarrow}_{R_1 \cup R_2} \overline{\varrho}$ and the definition of $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ it holds $\overline{\varrho} = g\ rhs_{M_1,f,\sigma}[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \phi_1, \ldots, \phi_r]\ \varrho_1 \cdots \varrho_s$. We calculate:

$$\begin{aligned} &|nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\bullet}}, g\ (f\ (\sigma\ t_1 \cdots t_p)\ \phi_1 \cdots \phi_r)\ \varrho_1 \cdots \varrho_s)|_\bullet \\ &= \quad \text{(by } \Rightarrow_{R_1^{\to\diamond}}\text{, using that } rhs_{M_1^{\to\diamond},f,\sigma} = \diamond\ rhs_{M_1,f,\sigma} \text{ by Definition 4.4,} \\ &\qquad \text{and by substitution)} \\ &|nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\bullet}}, g\ (\diamond\ rhs_{M_1,f,\sigma}[u_i \leftarrow t_i \mid i \in [p], \\ &\qquad\qquad y_k \leftarrow \phi_k \mid k \in [r]])\ \varrho_1 \cdots \varrho_s)|_\bullet \\ &= \quad \text{(by } \Rightarrow_{R_2^{\diamond\to\bullet}}\text{, using that } rhs_{M_2^{\diamond\to\bullet},g,\diamond} = \bullet\ (g\ v_1\ z_1 \cdots z_s) \text{ by Definition 4.4,} \\ &\qquad \text{and by normal form and definition of } |\,.\,|_\bullet) \\ &1 + |nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\bullet}}, g\ rhs_{M_1,f,\sigma}[u_i \leftarrow t_i \mid i \in [p], \\ &\qquad\qquad y_k \leftarrow \phi_k \mid k \in [r]]\ \varrho_1 \cdots \varrho_s)|_\bullet \end{aligned}$$

$\underline{\varrho = g\ (\delta\ \phi_1 \cdots \phi_q)\ \varrho_1 \cdots \varrho_s}$ for some $g \in G^{(s+1)}$, $\delta \in \Delta^{(q)}$, $\phi_1, \ldots, \phi_q \in WFT_{M_1}$, and $\varrho_1, \ldots, \varrho_s \in WFT_{M_1;M_2}$:
By the precondition $g\ (\delta\ \phi_1 \cdots \phi_q)\ \varrho_1 \cdots \varrho_s \overset{cbn}{\Rightarrow}_{R_1 \cup R_2} \overline{\varrho}$ and the definition of $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ it holds $\overline{\varrho} = rhs_{M_2,g,\delta}[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \phi_1, \ldots, \phi_q, \varrho_1, \ldots, \varrho_s]$. We calculate:

$$
\begin{aligned}
&|nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, g\ (\delta\ \phi_1 \cdots \phi_q)\ \varrho_1 \cdots \varrho_s)|_\bullet \\
&\quad = \quad \text{(by } \Rightarrow_{R_2^{\diamond\rightarrow\bullet}} \text{, using that } rhs_{M_2^{\diamond\rightarrow\bullet},g,\delta} = \bullet\ rhs_{M_2,g,\delta} \text{ by Definition 4.4)} \\
&|nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, (\bullet\ rhs_{M_2,g,\delta})[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \phi_1, \ldots, \phi_q, \varrho_1, \ldots, \varrho_s])|_\bullet \\
&\quad = \quad \text{(by substitution, normal form, and definition of } |\,.\,|_\bullet) \\
&1 + |nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, rhs_{M_2,g,\delta}[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \phi_1, \ldots, \phi_q, \varrho_1, \ldots, \varrho_s])|_\bullet
\end{aligned}
$$

$\underline{\varrho = \omega\ \varrho_1 \cdots \varrho_n}$ for some $\omega \in \Omega^{(n)}$ and $\varrho_1, \ldots, \varrho_n \in WFT_{M_1;M_2}$:
By the precondition $\omega\ \varrho_1 \cdots \varrho_n \overset{cbn}{\Rightarrow}_{R_1 \cup R_2} \overline{\varrho}$ and the definition of $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ there exist $b \in [n]$ and $\overline{\varrho_b}$ with $\varrho_b \overset{cbn}{\Rightarrow}_{R_1 \cup R_2} \overline{\varrho_b}$ such that $\overline{\varrho} = \omega\ \varrho_1 \cdots \varrho_{b-1}\ \overline{\varrho_b}\ \varrho_{b+1} \cdots \varrho_n$. We calculate:

$$
\begin{aligned}
&|nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, \omega\ \varrho_1 \cdots \varrho_n)|_\bullet \\
&\quad = \quad \text{(by normal form, definition of } |\,.\,|_\bullet \text{, and sum manipulation)} \\
&|nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, \varrho_b)|_\bullet + \sum_{d \in [n] \setminus \{b\}} |nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, \varrho_d)|_\bullet \\
&\quad = \quad \text{(by the induction hypothesis for } \varrho_b \overset{cbn}{\Rightarrow}_{R_1 \cup R_2} \overline{\varrho_b}) \\
&1 + |nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, \overline{\varrho_b})|_\bullet + \sum_{d \in [n] \setminus \{b\}} |nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, \varrho_d)|_\bullet \\
&\quad = \quad \text{(by normal form, definition of } |\,.\,|_\bullet \text{, and sum manipulation)} \\
&1 + |nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, \omega\ \varrho_1 \cdots \varrho_{b-1}\ \overline{\varrho_b}\ \varrho_{b+1} \cdots \varrho_n)|_\bullet \quad \square
\end{aligned}
$$

Example 4.6 (counting call-by-name steps for the mtts from Ex. 2.2) According to Definition 4.4, the mtts $M_{spine}^{\rightarrow\diamond}$ and $M_{pfx}^{\diamond\rightarrow\bullet}$ have rules

$$
\begin{aligned}
R_{spine}^{\rightarrow\diamond}: \quad & f_1\ (\alpha\ u_1)\ y_1 && \to \diamond\ (\alpha\ (f_1\ u_1\ y_1)) \\
& f_1\ (\beta\ u_1)\ y_1 && \to \diamond\ (f_2\ u_1\ y_1\ (\beta\ \epsilon)) \\
& f_1\ \epsilon\ y_1 && \to \diamond\ y_1 \\
& f_2\ (\alpha\ u_1)\ y_1\ y_2 && \to \diamond\ (\alpha\ (f_1\ u_1\ (\delta\ y_1\ y_2))) \\
& f_2\ (\beta\ u_1)\ y_1\ y_2 && \to \diamond\ (f_2\ u_1\ y_1\ (\beta\ (\beta\ y_2))) \\
& f_2\ \epsilon\ y_1\ y_2 && \to \diamond\ (\delta\ y_1\ y_2)
\end{aligned}
$$

and

$$
\begin{aligned}
R_{pfx}^{\diamond\rightarrow\bullet}: \quad & g\ (\delta\ v_1\ v_2)\ z_1 && \to \bullet\ (\delta\ (g\ v_1\ (g\ v_2\ z_1))) \\
& g\ (\alpha\ v_1)\ z_1 && \to \bullet\ (\alpha\ (g\ v_1\ z_1)) \\
& g\ (\beta\ v_1)\ z_1 && \to \bullet\ (\beta\ (g\ v_1\ z_1)) \\
& g\ \epsilon\ z_1 && \to \bullet\ (\epsilon\ z_1) \\
& g\ (\diamond\ v_1)\ z_1 && \to \bullet\ (g\ v_1\ z_1)
\end{aligned}
$$

respectively. Figure 4.c shows an example computation using these annotated rules, starting from the same input as in Figure 4.a. Note that the number of •-symbols in the final output is the same as the number of call-by-name reduction steps required in Figure 4.a. Further note that this final output is independent of the used reduction strategy: any other order of applying rules from $R^{\rightarrow\diamond}_{spine}$ and $R^{\diamond\rightarrow\bullet}_{pfx}$ would have yielded exactly the same normal form! □

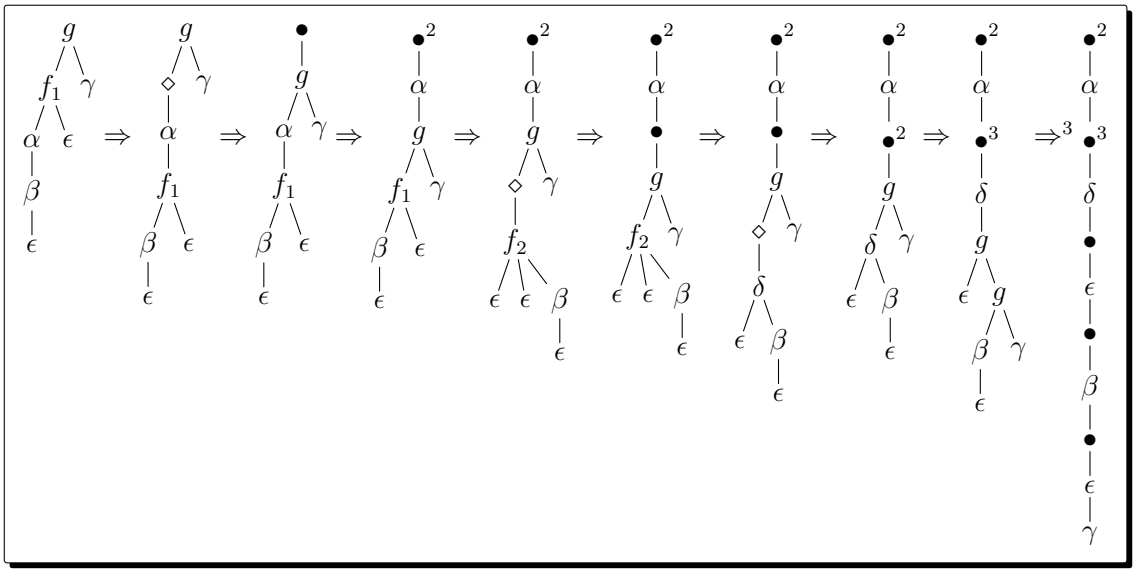

Figure 4.c: Reduction sequence using $\Rightarrow_{R^{\rightarrow\diamond}_{spine} \cup R^{\diamond\rightarrow\bullet}_{pfx}}$

For the composed program things are simpler than above because it consists of only a single mtt, so no propagation of annotated symbols from an intermediate to the final result is necessary. We just need to put some (other) special symbol on top of the right-hand side of every rule of $\overline{M_1;M_2}$. The following definition and lemma should thus hold no surprises.

Definition 4.7 ($\overline{M_1;M_2}^{\rightarrow\circ}$)
The mtt $\overline{M_1;M_2}^{\rightarrow\circ} = (H, \Sigma, \Omega \cup \{nil, \circ\}, e_{\overline{M_1;M_2}}, \overline{R_1;R_2}^{\rightarrow\circ})$ has rules as follows:

$$\frac{}{\begin{array}{l}\overline{R_1;R_2}^{\rightarrow\circ}: \\ \quad h\ (\sigma\ u_1 \cdots u_p) \cdots \rightarrow \circ\ rhs_{\overline{M_1;M_2},h,\sigma} \quad \forall h \in H, \sigma \in \Sigma^{(p)}\end{array}}$$

where on the left-hand sides the same context variables occur (in the same order) as in the corresponding rules of $\overline{M_1;M_2}$. □

Lemma 4.8 ($\overline{M_1;M_2}^{\rightarrow\circ}$ counts the call-by-name steps of $\overline{M_1;M_2}$)
$cbn_{\overline{M_1;M_2}} = \tau_{\overline{M_1;M_2}^{\rightarrow\circ}}\,;|\,\blacksquare\,|_\circ$ □

The proof is analogous to—but slightly simpler than—that of Lemma 4.5; we omit it.

4.4 Pushing annotation through composition

Through the characterization of $cbn_{M_1;M_2}$ and $cbn_{\overline{M_1;M_2}}$ in terms of $\tau_{M_1^{\rightarrow\diamond}}$, $\tau_{M_2^{\diamond\rightarrow\bullet}}$, and $\tau_{\overline{M_1;M_2}^{\rightarrow\circ}}$ we have for now freed ourselves from concerns about a particular reduction strategy (though we will return to this issue in Section 4.10 when considering the call-by-need case). Since the tree transduction induced by an mtt is defined without imposing call-by-name or another policy on reductions computing the normal forms of instantiations of its initial expression, we can reason about the outputs generated by the mtts $M_1^{\rightarrow\diamond}$, $M_2^{\diamond\rightarrow\bullet}$, and $\overline{M_1;M_2}^{\rightarrow\circ}$ with much more ease than about the lengths of very specific reduction sequences involving M_1, M_2, and $\overline{M_1;M_2}$. The externalization will thus prove fruitful.

One hurdle for analyzing the relation between the number of $\circ$-symbols produced by $\overline{M_1;M_2}^{\rightarrow\circ}$ and the number of $\bullet$-symbols produced by $M_1^{\rightarrow\diamond}$ followed by $M_2^{\diamond\rightarrow\bullet}$ is that the rule structure of the former mtt seems rather unconnected to that of the latter two on first sight. But of course, there is a tight connection in that $M_1^{\rightarrow\diamond}$ and $M_2^{\diamond\rightarrow\bullet}$ are annotated versions of the original mtts M_1 and M_2 while $\overline{M_1;M_2}^{\rightarrow\circ}$ was obtained by annotating the rules produced by our composition construction given those original mtts as input. The most promising approach to exploit this connection is by trying to get an alternative perspective on $\overline{M_1;M_2}^{\rightarrow\circ}$: rather than viewing it as an annotated version of the composed mtt, can we also view it as the composition of annotated versions of the original mtts? This would allow us to shift further attention completely to the level of (annotations of) the original program, after all the base on which we aim to formulate conditions for efficiency nondeterioration.

The quest for appropriately annotated versions of M_1 and M_2 whose composition according to Construction 3.1 would differ from $\overline{M_1;M_2}$ only by an added $\circ$-symbol on top of every (non-nil[1]) rule right-hand side clearly required a more thorough analysis of the internal workings of the composition construction and the interplay between its ingredients than the explanations in Section 3.1. The result, however, is surprisingly simple and elegant, and comes with a purely syntactic proof that it indeed has the desired characteristics. Moreover, it sheds new light on the roles of the states of the composed mtt as identified in Observation 3.3. The key is to analyze which constellations in the original program are responsible for a step of the composed program being performed.

Recall that a state $\overline{fg}$ of $\overline{M_1;M_2}$ essentially computes the output of state g of M_2 run on an intermediate result produced by state f of M_1. It consumes its recursion argument with the same granularity as f does, namely by pattern-matching against the root symbol. Hence, to chart steps of $\overline{fg}$ in terms of the original program, it makes sense to mark portions of the intermediate result produced by a single step of f (reusing the $\diamond$-symbol):

$$f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r \rightarrow \diamond\ (\cdots)$$

[1]Rules with the dummy symbol nil as right-hand side are guaranteed not to have an impact on the outcome of the composed mtt's computation. Hence, it is irrelevant how the corresponding right-hand side of the annotated version looks like.

and to record the consumption of such a portion by g as follows in the final output:

$$g\ (\diamond\ v_1)\ z_1 \cdots z_s \to \circ\ (g\ v_1\ z_1 \cdots z_s).$$

So far, the approach is the same as in [Voi02b] for the special case that one of the involved mtts is a tdtt; but for $\overline{k_f l_{g'}}$-states, which were not present in that special case, things become considerably more intricate. These states are used to determine the values of context parameters of M_2's state g' on reaching the positions of context parameters of M_1's state f in an intermediate result produced by f for a given input. To model such behavior, we explicitly record positions in the intermediate result where a context parameter of f was used by marking all context variables in rule right-hand sides of f with yet another dedicated symbol $\star$. The cost associated in the composed program with accessing a context parameter of g' on a thus marked context parameter of a state of M_1 is reflected by adding the following rule:

$$g'\ (\star\ v_1)\ z_1 \cdots z_{s'} \to g'\ v_1\ (\circ\ z_1)\ \cdots\ (\circ\ z_{s'}).$$

Its effect is illustrated in Figure 4.d; compare that to Figure 3.a. The chosen scheme ensures that if some of the context parameters of g' are not required for its computation on a concrete instantiation of v_1, e.g. on ϕ_k in the figure, then the $\circ$-symbols stored in the corresponding positions will not be included in the final count as they will be discarded along with the values they mark. That this is exactly what we want hinges on the fact that a particular $\overline{k_f l_{g'}}$-computation is necessary only if the corresponding value is really required in the replacement (2) for (1) in Section 3.1.

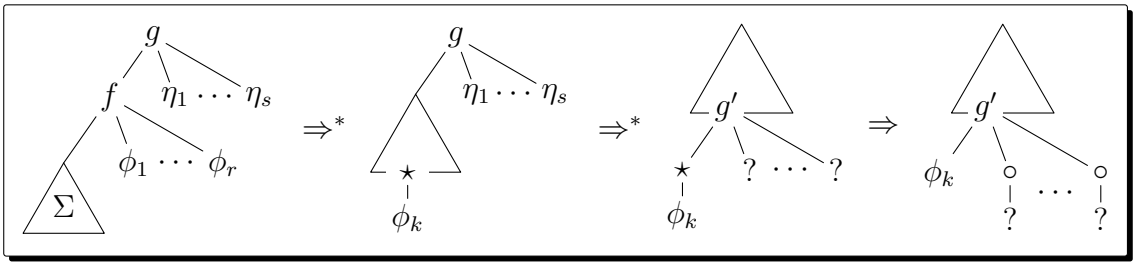

Figure 4.d: Effects of the $\star$-annotation and associated rules for the second mtt

Summarizing, we obtain the following annotated versions of M_1 and M_2.

Definition 4.9 ($M_1^{\to\diamond\star}$ and $M_2^{\diamond\star\to\circ}$)
The mtts $M_1^{\to\diamond\star} = (F, \Sigma, \Delta \cup \{\diamond, \star\}, e_{M_1}, R_1^{\to\diamond\star})$ and $M_2^{\diamond\star\to\circ} = (G, \Delta \cup \{\diamond, \star\}, \Omega \cup \{\circ\}, e_{M_2}, R_2^{\diamond\star\to\circ})$ have rules as follows:

$R_1^{\to\diamond\star}$:	
$f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r \to \diamond\ rhs_{M_1,f,\sigma}[y_k \leftarrow \star\ y_k \mid k \in [r]]$	$\forall f \in F^{(r+1)}, \sigma \in \Sigma^{(p)}$
$R_2^{\diamond\star\to\circ}$:	
$g\ (\delta\ v_1 \cdots v_q)\ \ z_1 \cdots z_s \to rhs_{M_2,g,\delta}$	$\forall g \in G^{(s+1)}, \delta \in \Delta^{(q)}$
$g\ (\diamond\ v_1)\ \ z_1 \cdots z_s \to \circ\ (g\ v_1\ z_1 \cdots z_s)$	$\forall g \in G^{(s+1)}$
$g\ (\star\ v_1)\ \ z_1 \cdots z_s \to g\ v_1\ (\circ\ z_1)\ \cdots\ (\circ\ z_s)$	$\forall g \in G^{(s+1)}$

Note that $M_1^{\rightarrow\diamond\star}$ is a tdtt iff M_1 is; likewise for context-linearity. Further, $M_2^{\diamond\star\rightarrow\circ}$ is a tdtt iff M_2 is; likewise for weakly single-useness. $\square$

By the remarks in Definition 4.9 applicability of Construction 3.1 to M_1 and M_2 implies its applicability also to $M_1^{\rightarrow\diamond\star}$ and $M_2^{\diamond\star\rightarrow\circ}$, yielding the mtt $\overline{M_1^{\rightarrow\diamond\star};M_2^{\diamond\star\rightarrow\circ}} = (H, \Sigma, \Omega \cup \{nil, \circ\}, e_{\overline{M_1^{\rightarrow\diamond\star};M_2^{\diamond\star\rightarrow\circ}}}, \overline{R_1^{\rightarrow\diamond\star};R_2^{\diamond\star\rightarrow\circ}})$, which we aim to show equivalent to $\overline{M_1;M_2}^{\rightarrow\circ}$. Note that the state set of $\overline{M_1^{\rightarrow\diamond\star};M_2^{\diamond\star\rightarrow\circ}}$ is the same H as for $\overline{M_1;M_2}$ (and thus for $\overline{M_1;M_2}^{\rightarrow\circ}$) because its definition in Construction 3.1 depends only on F and G. Likewise, the $nest_f$-functions and the rewrite systems Pre and $Pair$ are unchanged between the two applications of the composition construction to M_1 and M_2, respectively to $M_1^{\rightarrow\diamond\star}$ and $M_2^{\diamond\star\rightarrow\circ}$, assuming the same ordering of states in G is used in both cases. The following two auxiliary lemmata study the (non-)impact of the differences which do exist between the rule sets of the mtts involved in those two applications on certain computations to be performed when constructing the rule right-hand sides for the composed mtt.

Lemma 4.10 (auxiliary)
For every $f \in F^{(r+1)}$, $g \in G^{(s+1)}$, and $\sigma \in \Sigma$:

$$\begin{aligned} &nf(\Rightarrow_{R_2^{\diamond\star\rightarrow\circ} \cup Pre \cup Pair}, g\ rhs_{M_1,f,\sigma}[y_k \leftarrow \star\ y_k \mid k \in [r]]\ z_1 \cdots z_s) \\ = {} &nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, g\ rhs_{M_1,f,\sigma}\ z_1 \cdots z_s). \quad \square \end{aligned}$$

The proof (of a more general statement by simultaneous induction over the structure of right-hand sides of rules of M_1, using nested inductions over the structure of right-hand sides of rules of M_2 and over the reversed subset-order on the type of the third argument of $nest_f$-functions) can be found in the appendix. It relies on the facts that for every $\delta \in \Delta$ the δ-rules do not differ between $R_2^{\diamond\star\rightarrow\circ}$ and R_2, that $rhs_{M_1,f,\sigma}[y_k \leftarrow \star\ y_k \mid k \in [r]]$ contains no $\diamond$-symbols, that it contains $\star$-symbols only atop context variables y_k, and that a reduction

$$g\ (\star\ y_k)\ \varrho_1 \cdots \varrho_s \Rightarrow_{R_2^{\diamond\star\rightarrow\circ}} g\ y_k\ (\circ\ \varrho_1)\ \cdots\ (\circ\ \varrho_s) \Rightarrow_{Pre} y_{k,g}$$

for arbitrary g, y_k, and $\varrho_1, \ldots, \varrho_s$ has the same outcome as a $\Rightarrow_{Pre}$-reduction for a corresponding call to g on (unannotated) y_k.

Lemma 4.11 (auxiliary)
For every $f \in F^{(r+1)}$, $g \in G^{(s+1)}$, $\sigma \in \Sigma$, $\pi \in paths(rhs_{M_1,f,\sigma})$ with $lab(rhs_{M_1,f,\sigma}, \pi) \in Y_r$, and $l \in [s]$:

$$\begin{aligned} &nf(\Rightarrow_{R_2^{\diamond\star\rightarrow\circ} \cup Pre \cup Pair}, par_{M_2^{\diamond\star\rightarrow\circ}, rhs_{M_1^{\rightarrow\diamond\star},f,\sigma}}(1\pi, g, l)) \\ = {} &nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, par_{M_2, rhs_{M_1,f,\sigma}}(\pi, g, l)). \quad \square \end{aligned}$$

The proof (of a slightly more general statement by induction over the prefix-order of paths in $rhs_{M_1,f,\sigma}$, using a nested induction over the structure of right-hand sides of rules of M_2 and two statements from the proof of Lemma 4.10) can be found in the appendix. Beside the facts that for every $\delta \in \Delta$ the δ-rules are the same in $R_2^{\diamond\star\rightarrow\circ}$ and R_2, that no $\star$-symbols occur between the root of $rhs_{M_1^{\rightarrow\diamond\star},f,\sigma} = \diamond\ rhs_{M_1,f,\sigma}[y_k \leftarrow$

$\star\ y_k \mid k \in [r]]$ and the one at path 1π, and that the use of substitution $_[y_k \leftarrow \star\ y_k \mid k \in [r]]$ makes for no difference between computations using states from G with reduction relations $\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}$ and $\Rightarrow_{R_2 \cup Pre \cup Pair}$, respectively (see above), the key is that the recursive calls in $R_2^{\diamond\star\to\circ}$-rules at $\diamond$ are performed with unchanged context parameters.

We can now show that except for rules whose right-hand side is *nil* the mtts $\overline{M_1;M_2}^{\to\circ}$ and $\overline{M_1^{\to\diamond\star};M_2^{\diamond\star\to\circ}}$ are syntactically equivalent, taking into account that by Definition 4.7 the initial expression of the former mtt is $e_{\overline{M_1;M_2}}$ and that by Construction 3.1 the left-hand sides of the rules in $\overline{R_1^{\to\diamond\star};R_2^{\diamond\star\to\circ}}$ coincide with the left-hand sides of the $\overline{R_1;R_2}$-rules and thus by Definition 4.7 also with those of the $\overline{R_1;R_2}^{\to\circ}$-rules.

Lemma 4.12 ($\overline{M_1;M_2}^{\to\circ}$ and $\overline{M_1^{\to\diamond\star};M_2^{\diamond\star\to\circ}}$ are equal up to dummy rules)

1. $e_{\overline{M_1^{\to\diamond\star};M_2^{\diamond\star\to\circ}}} = e_{\overline{M_1;M_2}}$
2. For every $h \in H$ and $\sigma \in \Sigma$, if $rhs_{\overline{M_1^{\to\diamond\star};M_2^{\diamond\star\to\circ}},h,\sigma}$ is not equal to *nil*, then it is equal to $rhs_{\overline{M_1;M_2}^{\to\circ},h,\sigma}$.

Proof

1. We calculate as follows:

$$
\begin{aligned}
& e_{\overline{M_1^{\to\diamond\star};M_2^{\diamond\star\to\circ}}} \\
&\quad = \quad \text{(by Construction 3.1 and Definition 4.9)} \\
& nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pair}, e_{M_2}[v_1 \leftarrow e_{M_1}]) \\
&\quad = \quad \text{(since } e_{M_2}[v_1 \leftarrow e_{M_1}] \in T_{F\cup G\cup\Delta\cup\Omega}(U_1) \text{ contains no } \diamond\text{- or } \star\text{-symbols} \\
&\qquad\quad \text{and since for every } \delta \in \Delta \text{ the } \delta\text{-rules in } R_2^{\diamond\star\to\circ} \text{ coincide with} \\
&\qquad\quad \text{those in } R_2 \text{ by Definition 4.9)} \\
& nf(\Rightarrow_{R_2 \cup Pair}, e_{M_2}[v_1 \leftarrow e_{M_1}]) \\
&\quad = \quad \text{(by Construction 3.1)} \\
& e_{\overline{M_1;M_2}}
\end{aligned}
$$

2. For fixed $\sigma \in \Sigma$ by case analysis on $h \in H$:

   $\underline{h = \overline{fg}}$ for some $f \in F^{(r+1)}$ and $g \in G^{(s+1)}$:

$$
\begin{aligned}
& rhs_{\overline{M_1^{\to\diamond\star};M_2^{\diamond\star\to\circ}},\overline{fg},\sigma} \\
&\quad = \quad \text{(by Construction 3.1)} \\
& nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, g\ rhs_{M_1^{\to\diamond\star},f,\sigma}\ z_1 \cdots z_s) \\
&\quad = \quad \text{(by } rhs_{M_1^{\to\diamond\star},f,\sigma} = \diamond\ rhs_{M_1,f,\sigma}[y_k \leftarrow \star\ y_k \mid k \in [r]], \text{ cf. Definition 4.9)} \\
& nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, g\ (\diamond\ rhs_{M_1,f,\sigma}[y_k \leftarrow \star\ y_k \mid k \in [r]])\ z_1 \cdots z_s) \\
&\quad = \quad \text{(by } \Rightarrow_{R_2^{\diamond\star\to\circ}}, \text{ using that } rhs_{M_2^{\diamond\star\to\circ},g,\diamond} = \circ\ (g\ v_1\ z_1 \cdots z_s) \text{ by} \\
&\qquad\quad \text{Definition 4.9, and by normal form)} \\
& \circ\ nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, g\ rhs_{M_1,f,\sigma}[y_k \leftarrow \star\ y_k \mid k \in [r]]\ z_1 \cdots z_s) \\
&\quad = \quad \text{(by Lemma 4.10)}
\end{aligned}
$$

$\circ\ nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, g\ rhs_{M_1,f,\sigma}\ z_1 \cdots z_s)$
$=$ (by Construction 3.1)
$\circ\ rhs_{\overline{M_1;M_2},\overline{fg},\sigma}$
$=$ (by Definition 4.7)
$rhs_{\overline{M_1;M_2}^{\to\circ},\overline{fg},\sigma}$

$\underline{h = \overline{k_f l_g}}$ for some $f \in F^{(r+1)}$, $g \in G^{(s+1)}$, $k \in [r]$, and $l \in [s]$:
Note that $rhs_{M_1^{\to\diamond\star},f,\sigma} = \diamond\ rhs_{M_1,f,\sigma}[y_{k'} \leftarrow \star\ y_{k'} \mid k' \in [r]]$ by Definition 4.9.

- If $rhs_{M_1,f,\sigma}$ does not contain the context variable y_k, then $rhs_{M_1^{\to\diamond\star},f,\sigma}$ does not either, and hence by Construction 3.1:

$$rhs_{\overline{M_1^{\to\diamond\star};M_2^{\diamond\star\to\circ}},\overline{k_f l_g},\sigma} = nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, nil) = nil.$$

- Otherwise, if $lab(rhs_{M_1,f,\sigma}, \pi) = y_k$—and thus $sub(rhs_{M_1^{\to\diamond\star},f,\sigma}, 1\pi) = \star\ y_k$—for some $\pi \in paths(rhs_{M_1,f,\sigma})$, then:
  $rhs_{\overline{M_1^{\to\diamond\star};M_2^{\diamond\star\to\circ}},\overline{k_f l_g},\sigma}$
  $=$ (by Construction 3.1)
  $nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, par_{M_2^{\diamond\star\to\circ},rhs_{M_1^{\to\diamond\star},f,\sigma}}(1\pi 1, g, l))$
  $=$ (by definition of $par_{M_2^{\diamond\star\to\circ},rhs_{M_1^{\to\diamond\star},f,\sigma}}$ in Construction 3.1,
  using that $rhs_{M_2^{\diamond\star\to\circ},g,\star} = g\ v_1\ (\circ\ z_1)\ \cdots\ (\circ\ z_s)$ by Definition 4.9)
  $nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, \circ\ par_{M_2^{\diamond\star\to\circ},rhs_{M_1^{\to\diamond\star},f,\sigma}}(1\pi, g, l))$
  $=$ (by normal form)
  $\circ\ nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, par_{M_2^{\diamond\star\to\circ},rhs_{M_1^{\to\diamond\star},f,\sigma}}(1\pi, g, l))$
  $=$ (by Lemma 4.11)
  $\circ\ nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, par_{M_2,rhs_{M_1,f,\sigma}}(\pi, g, l))$
  $=$ (by Construction 3.1)
  $\circ\ rhs_{\overline{M_1;M_2},\overline{k_f l_g},\sigma}$
  $=$ (by Definition 4.7)
  $rhs_{\overline{M_1;M_2}^{\to\circ},\overline{k_f l_g},\sigma}$ $\square$

Using Lemma 4.12 and the fact that $\overline{M_1^{\to\diamond\star};M_2^{\diamond\star\to\circ}}$ was obtained by our **semantics-preserving** composition construction from $M_1^{\to\diamond\star}$ and $M_2^{\diamond\star\to\circ}$, we can express the semantics of $\overline{M_1;M_2}^{\to\circ}$—counting the call-by-name steps of the composed program—directly in terms of (the semantics of) these annotated versions of the original mtts. This implies that we do not even need to perform Construction 3.1 to analyze the efficiency of a composed program resulting from it. While the externalization by annotation in the previous section freed us from further concerns about the specifics of call-by-name reduction, the following lemma thus eliminates any necessity to consider the composition construction itself in the further development.

Lemma 4.13 ($M_1^{\to\diamond\star}$ and $M_2^{\diamond\star\to\circ}$ simulate $\overline{M_1;M_2}^{\to\circ}$)
$\tau_{\overline{M_1;M_2}^{\to\circ}} = \tau_{M_1^{\to\diamond\star}}\,;\tau_{M_2^{\diamond\star\to\circ}}$

Proof
The lemma follows from an instantiation of Theorem 3.2,

$$\tau_{\overline{M_1^{\rightarrow\diamond\star};M_2^{\diamond\star\rightarrow\circ}}} = \tau_{M_1^{\rightarrow\diamond\star}} ; \tau_{M_2^{\diamond\star\rightarrow\circ}} : T_\Sigma \longrightarrow T_{\Omega\cup\{\circ\}} ,$$

if for every $t \in T_\Sigma$ we can show that

$$\tau_{\overline{M_1^{\rightarrow\diamond\star};M_2^{\diamond\star\rightarrow\circ}}}(t) = nf(\Rightarrow_{\overline{R_1^{\rightarrow\diamond\star};R_2^{\diamond\star\rightarrow\circ}}}, e_{\overline{M_1^{\rightarrow\diamond\star};M_2^{\diamond\star\rightarrow\circ}}}[u_1 \leftarrow t])$$

is equal to

$$\tau_{\overline{M_1;M_2}^{\rightarrow\circ}}(t) = nf(\Rightarrow_{\overline{R_1;R_2}^{\rightarrow\circ}}, e_{\overline{M_1;M_2}}[u_1 \leftarrow t]).$$

Since the range of $\tau_{\overline{M_1^{\rightarrow\diamond\star};M_2^{\diamond\star\rightarrow\circ}}}$ contains no trees with *nil*-symbols, it must be possible to obtain the former normal form by applying only rules from $\overline{R_1^{\rightarrow\diamond\star};R_2^{\diamond\star\rightarrow\circ}}$ whose right-hand side is not equal to *nil*. But as shown in Lemma 4.12(2), all such rules are also contained in $\overline{R_1;R_2}^{\rightarrow\circ}$, which implies

$$e_{\overline{M_1^{\rightarrow\diamond\star};M_2^{\diamond\star\rightarrow\circ}}}[u_1 \leftarrow t] \Rightarrow^*_{\overline{R_1;R_2}^{\rightarrow\circ}} \tau_{\overline{M_1^{\rightarrow\diamond\star};M_2^{\diamond\star\rightarrow\circ}}}(t).$$

By Lemma 4.12(1) and the fact that every element of $T_{\Omega\cup\{\circ\}}$ is irreducible with respect to $\Rightarrow_{\overline{R_1;R_2}^{\rightarrow\circ}}$, this means that $\tau_{\overline{M_1^{\rightarrow\diamond\star};M_2^{\diamond\star\rightarrow\circ}}}(t)$ is a normal form of $e_{\overline{M_1;M_2}}[u_1 \leftarrow t]$ with respect to $\Rightarrow_{\overline{R_1;R_2}^{\rightarrow\circ}}$. The desired equality follows because normal forms with respect to $\Rightarrow_{\overline{R_1;R_2}^{\rightarrow\circ}}$ are unique. □

To appreciate the value of the previous lemma (in combination with Lemma 4.8), consider the following for the mtts from Example 2.2.

Example 4.14 (counting steps for composed program in running ex.)
The number of $\circ$-symbols produced using the annotated rules

$$\begin{array}{rlll}
R^{\rightarrow\diamond\star}_{spine} : & f_1\ (\alpha\ u_1)\ y_1 & \rightarrow \diamond\ (\alpha\ (f_1\ u_1\ (\star\ y_1))) \\
& f_1\ (\beta\ u_1)\ y_1 & \rightarrow \diamond\ (f_2\ u_1\ (\star\ y_1)\ (\beta\ \epsilon)) \\
& f_1\ \epsilon\ \ \ y_1 & \rightarrow \diamond\ (\star\ y_1) \\
& f_2\ (\alpha\ u_1)\ y_1\ y_2 & \rightarrow \diamond\ (\alpha\ (f_1\ u_1\ (\delta\ (\star\ y_1)\ (\star\ y_2)))) \\
& f_2\ (\beta\ u_1)\ y_1\ y_2 & \rightarrow \diamond\ (f_2\ u_1\ (\star\ y_1)\ (\beta\ (\beta\ (\star\ y_2)))) \\
& f_2\ \epsilon\ \ \ y_1\ y_2 & \rightarrow \diamond\ (\delta\ (\star\ y_1)\ (\star\ y_2))
\end{array}$$

of $M^{\rightarrow\diamond\star}_{spine}$ (cf. Definition 4.9) and the rules of $M^{\diamond\star\rightarrow\circ}_{pfx}$ (i.e. the rules from R_{pfx} plus $g\ (\diamond\ v_1)\ z_1 \rightarrow \circ\ (g\ v_1\ z_1)$ and $g\ (\star\ v_1)\ z_1 \rightarrow g\ v_1\ (\circ\ z_1)$) in the example computation in Figure 4.e is the same as the number of call-by-name reduction steps required in Figure 3.b for the same input. Actually, there even exists a more fine-grained correspondence regarding how many call-by-name reduction steps are required in Figure 3.b to produce which part of the output. Namely, one step is required there to output the top-most α-symbol, after two further steps one obtains (in one go) one δ- and one ϵ-symbol. Only after three more steps the next chunk of the output is produced (β and ϵ), before one last step gives the final γ. This information is exactly coded in the distribution of $\circ$-symbols over the output tree in Figure 4.e. The same kind of correspondence existed already between Figures 4.c and 4.a. □

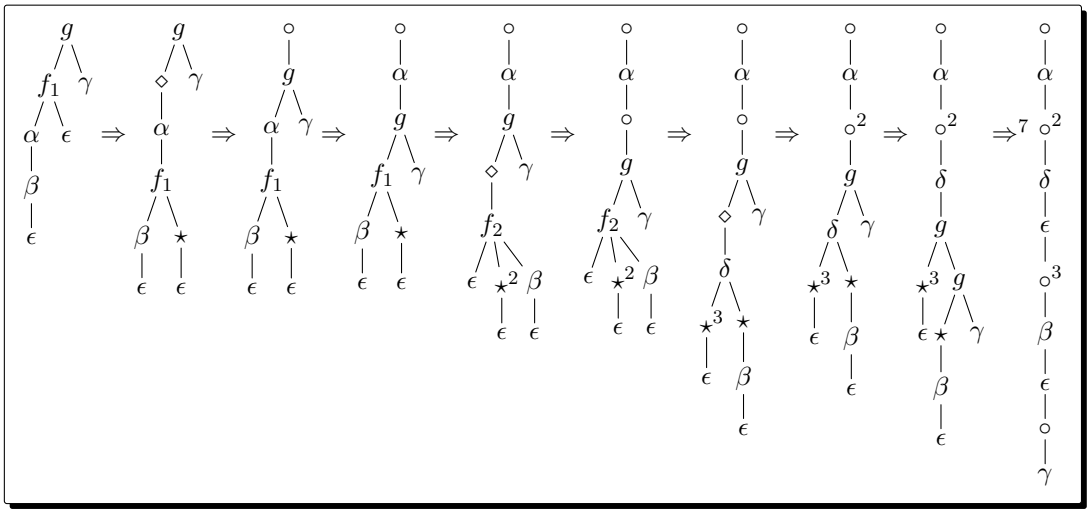

Figure 4.e: Reduction sequence using $\Rightarrow_{R^{\rightarrow\diamond\star}_{spine} \cup R^{\diamond\star\rightarrow\circ}_{pfx}}$

4.5 Combining annotations, and a first efficiency result

Summarizing the results from the previous two sections, we have that $M_1^{\rightarrow\diamond\star}$ and $M_2^{\diamond\star\rightarrow\circ}$ count the call-by-name steps of the composed program while $M_1^{\rightarrow\diamond}$ and $M_2^{\diamond\rightarrow\bullet}$ count the call-by-name steps of the original one. To determine the relative efficiency of the composed vs. the original program for a particular input, all we need to do is to compare the numbers of $\circ$- and $\bullet$-symbols, respectively, in the outputs produced by those different annotations of the mtts M_1 and M_2 for this input (e.g., to compare the values computed in Figures 4.e and 4.c). To compare these numbers on a more general level, independent from a concrete input tree, it is clearly beneficial to work with only one annotated version for each of M_1 and M_2. Quite naturally, "overlaying" the different annotation schemes leads to the following rules for $f \in F^{(r+1)}$, $\sigma \in \Sigma^{(p)}$, $g \in G^{(s+1)}$, and $\delta \in \Delta^{(q)}$:

$$
\begin{array}{lll}
f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r & \rightarrow \diamond\ rhs_{M_1,f,\sigma}[y_k \leftarrow \star\ y_k \mid k \in [r]] \\
g\ (\delta\ v_1 \cdots v_q)\ z_1 \cdots z_s & \rightarrow \bullet\ rhs_{M_2,g,\delta} \\
g\ (\diamond\ v_1)\ z_1 \cdots z_s & \rightarrow \circ\ (\bullet\ (g\ v_1\ z_1 \cdots z_s)) \\
g\ (\star\ v_1)\ z_1 \cdots z_s & \rightarrow g\ v_1\ (\circ\ z_1)\ \cdots\ (\circ\ z_s)
\end{array}
$$

Since we are only interested in whether there are more occurrences of one of the symbols $\circ$ and $\bullet$ than of the other, rather than in their absolute numbers, the adjacent $\circ$ and $\bullet$ in the g-rules at $\diamond$ can be omitted.[2] Since the new g-rules at

[2]One interpretation for this is that there exists a one-to-one correspondence between steps of $f \in F$ in the original program and steps of $\overline{fg_1}, \ldots, \overline{fg_\mu}$ in the composed one. In particular, this means that if the composed program contains no $\overline{k_f l_g}$-states—which is the case exactly if one of the two original mtts is a tdtt—, then the composed program does for no input perform more call-by-name steps than the original program, as formally established in Theorem 4.20 below. More

$\diamond$ will simply delete any $\diamond$-symbols appearing in the intermediate result, we can equivalently annotate as follows:

$$\begin{array}{ll} f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r & \to rhs_{M_1,f,\sigma}[y_k \leftarrow \star\ y_k \mid k \in [r]] \\ g\ (\delta\ v_1 \cdots v_q)\ \ z_1 \cdots z_s & \to \bullet\ rhs_{M_2,g,\delta} \\ g\ (\star\ v_1) \qquad\quad z_1 \cdots z_s & \to g\ v_1\ (\circ\ z_1)\ \cdots\ (\circ\ z_s) \end{array}$$

For the later analysis it will be beneficial to keep the δ-rules unchanged from M_2, i.e. devoid of any annotation. To achieve this, we instead of producing a $\bullet$-symbol whenever such a rule is applied arrange for every $\delta \in \Delta$ in the intermediate result to be explicitly marked with a $\bullet$-symbol that is then reproduced separately using appropriate new rules. This leads to the following annotated versions of M_1 and M_2.

Definition 4.15 ($M_1^{\to\star\bullet}$ and $M_2^{\star\bullet\to\circ\bullet}$)
The mtts $M_1^{\to\star\bullet} = (F, \Sigma, \Delta \cup \{\star, \bullet\}, e_{M_1^{\to\star\bullet}}, R_1^{\to\star\bullet})$ and $M_2^{\star\bullet\to\circ\bullet} = (G, \Delta \cup \{\star, \bullet\}, \Omega \cup \{\circ, \bullet\}, e_{M_2}, R_2^{\star\bullet\to\circ\bullet})$ are given as follows:

$R_1^{\to\star\bullet}$:

$$f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r \to rhs_{M_1,f,\sigma}[\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]\ [y_k \leftarrow \star\ y_k \mid k \in [r]] \quad \forall f \in F^{(r+1)}, \sigma \in \Sigma^{(p)}$$

$e_{M_1^{\to\star\bullet}} = e_{M_1}[\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]$

$R_2^{\star\bullet\to\circ\bullet}$:

$$\begin{array}{lll} g\ (\delta\ v_1 \cdots v_q)\ \ z_1 \cdots z_s & \to rhs_{M_2,g,\delta} & \forall g \in G^{(s+1)}, \delta \in \Delta^{(q)} \\ g\ (\star\ v_1) \qquad\quad z_1 \cdots z_s & \to g\ v_1\ (\circ\ z_1)\ \cdots\ (\circ\ z_s) & \forall g \in G^{(s+1)} \\ g\ (\bullet\ v_1) \qquad\quad z_1 \cdots z_s & \to \bullet\ (g\ v_1\ z_1 \cdots z_s) & \forall g \in G^{(s+1)} \end{array}$$

Note that $M_2^{\star\bullet\to\circ\bullet}$ is context-linear iff M_2 is. $\square$

To show that the output computed by $M_1^{\to\star\bullet}$ and $M_2^{\star\bullet\to\circ\bullet}$ correctly reflects the difference between the number of $\circ$-symbols produced by $M_1^{\to\diamond\star}$ and $M_2^{\diamond\star\to\circ}$ on the one hand and the number of $\bullet$-symbols produced by $M_1^{\to\diamond}$ and $M_2^{\diamond\to\bullet}$ on the other hand, we first need two auxiliary lemmata.

Lemma 4.16 (auxiliary)
For every $t \in T_\Sigma$:

1. $\tau_{M_1^{\to\diamond}}(t) = \tau_{M_1^{\to\diamond\star}}(t)[\star \twoheadleftarrow id]$
2. $\tau_{M_1^{\to\star\bullet}}(t) = \tau_{M_1^{\to\diamond\star}}(t)[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]$.

Proof
Point 1 is immediate from the facts that $M_1^{\to\diamond\star}$ and $M_1^{\to\diamond}$ have the same initial expression containing no $\star$-symbols and that for every $f \in F$ and $\sigma \in \Sigma$, $rhs_{M_1^{\to\diamond},f,\sigma} = rhs_{M_1^{\to\diamond\star},f,\sigma}[\star \twoheadleftarrow id]$. Similarly, point 2 follows from the facts that $e_{M_1^{\to\star\bullet}}$ can be obtained from the initial expression of $M_1^{\to\diamond\star}$ (e_{M_1}, containing no $\diamond$-symbols) via the substitutions $[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]$ and that for every $f \in F$ and $\sigma \in \Sigma$, $rhs_{M_1^{\to\star\bullet},f,\sigma} = rhs_{M_1^{\to\diamond\star},f,\sigma}[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]$. $\square$

precisely, the amount of improvement corresponds to the number of steps performed by states of M_2 in the original program.

The second auxiliary lemma is formulated using the function $|\,\cdot\,|_{\circ-\bullet} = |\,\cdot\,|_\circ - |\,\cdot\,|_\bullet$.

Lemma 4.17 (auxiliary)
For every $t' \in T_{\Delta\cup\{\diamond,\star\}}$:

$$|\tau_{M_2^{\diamond\star\to\circ}}(t')|_\circ - |\tau_{M_2^{\diamond\to\bullet}}(t'[\star \twoheadleftarrow id])|_\bullet = |\tau_{M_2^{\star\bullet\to\circ\bullet}}(t'[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet\cdot\delta \mid \delta\in\Delta])|_{\circ-\bullet}. \quad \square$$

The proof (of a more general statement by simultaneous induction over the structure of elements of $T_{\Delta\cup\{\diamond,\star\}}$, using a nested induction over the structure of right-hand sides of rules of—and of the initial expression of—M_2) can be found in the appendix; it holds no big surprises.

Simply plugging the two auxiliary lemmata together, we easily obtain the following (cf. the appendix for details).

Lemma 4.18 ($M_1^{\to\star\bullet}$ and $M_2^{\star\bullet\to\circ\bullet}$ count the difference of $\circ$- and $\bullet$-symbols)
$\tau_{M_1^{\to\diamond\star}} ; \tau_{M_2^{\diamond\star\to\circ}} ; |\,\cdot\,|_\circ - \tau_{M_1^{\to\diamond}} ; \tau_{M_2^{\diamond\to\bullet}} ; |\,\cdot\,|_\bullet = \tau_{M_1^{\to\star\bullet}} ; \tau_{M_2^{\star\bullet\to\circ\bullet}} ; |\,\cdot\,|_{\circ-\bullet}$ $\square$

Example 4.19 (instantiation of Lemma 4.18 to the running example)
Compare the output computed using the annotated rules

$$\begin{array}{rl}
R_{spine}^{\to\star\bullet}: & f_1\ (\alpha\ u_1)\ y_1 \to \bullet\ (\alpha\ (f_1\ u_1\ (\star\ y_1)))\\
& f_1\ (\beta\ u_1)\ y_1 \to f_2\ u_1\ (\star\ y_1)\ (\bullet\ (\beta\ (\bullet\ \epsilon)))\\
& f_1\ \epsilon\ y_1 \to \star\ y_1\\
& f_2\ (\alpha\ u_1)\ y_1\ y_2 \to \bullet\ (\alpha\ (f_1\ u_1\ (\bullet\ (\delta\ (\star\ y_1)\ (\star\ y_2)))))\\
& f_2\ (\beta\ u_1)\ y_1\ y_2 \to f_2\ u_1\ (\star\ y_1)\ (\bullet\ (\beta\ (\bullet\ (\beta\ (\star\ y_2)))))\\
& f_2\ \epsilon\ y_1\ y_2 \to \bullet\ (\delta\ (\star\ y_1)\ (\star\ y_2))
\end{array}$$

of $M_{spine}^{\to\star\bullet}$ (and the new initial expression $e_{M_{spine}^{\to\star\bullet}} = f_1\ u_1\ (\bullet\ \epsilon)$, cf. Definition 4.15) and the rules of $M_{pfx}^{\star\bullet\to\circ\bullet}$ (i.e. the rules from R_{pfx} plus $g\ (\star\ v_1)\ z_1 \to g\ v_1\ (\circ\ z_1)$ and $g\ (\bullet\ v_1)\ z_1 \to \bullet\ (g\ v_1\ z_1)$) in Figure 4.f to that from Figures 4.c and 4.e (for the same input). The continuity is that there is one more $\bullet$-symbol than there are $\circ$-symbols. $\square$

It is easy to see that the output computed by $M_1^{\to\star\bullet}$ and $M_2^{\star\bullet\to\circ\bullet}$ will contain no $\circ$-symbols if one of the two is a tdtt. This yields a first useful result of our efficiency analysis, albeit for a very special case.

Theorem 4.20 (efficiency nondeterioration if M_1 or M_2 is a tdtt)
If either of M_1 and M_2 is a tdtt, then $cbn_{\overline{M_1;M_2}}(t) \le cbn_{M_1;M_2}(t)$ for every $t \in T_\Sigma$.

Proof
We calculate

$$\begin{array}{l}
cbn_{\overline{M_1;M_2}} - cbn_{M_1;M_2}\\
\quad = \quad \text{(by Lemmata 4.8 and 4.5)}\\
\tau_{\overline{M_1;M_2}^{\to\circ}} ; |\,\cdot\,|_\circ - \tau_{M_1^{\to\diamond}} ; \tau_{M_2^{\diamond\to\bullet}} ; |\,\cdot\,|_\bullet\\
\quad = \quad \text{(by Lemma 4.13)}\\
\tau_{M_1^{\to\diamond\star}} ; \tau_{M_2^{\diamond\star\to\circ}} ; |\,\cdot\,|_\circ - \tau_{M_1^{\to\diamond}} ; \tau_{M_2^{\diamond\to\bullet}} ; |\,\cdot\,|_\bullet\\
\quad = \quad \text{(by Lemma 4.18)}\\
\tau_{M_1^{\to\star\bullet}} ; \tau_{M_2^{\star\bullet\to\circ\bullet}} ; |\,\cdot\,|_{\circ-\bullet},
\end{array}$$

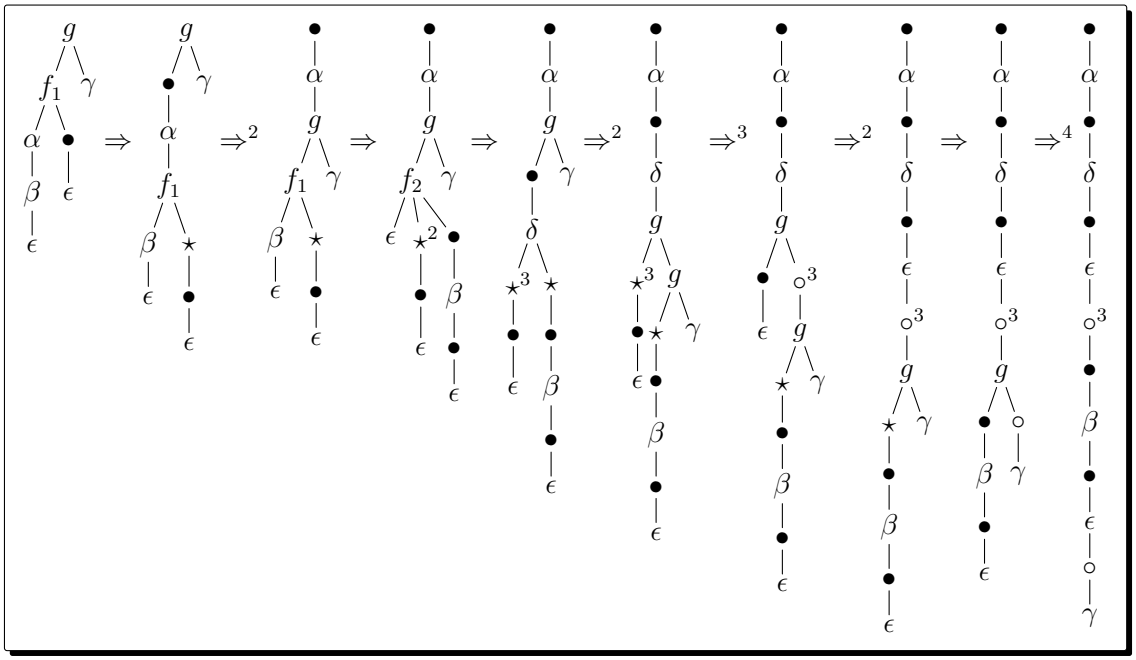

Figure 4.f: Reduction sequence using $\Rightarrow_{R^{\rightarrow\star\bullet}_{spine} \cup R^{\star\bullet\rightarrow\circ\bullet}_{pfx}}$

which implies

$$cbn_{\overline{M_1;M_2}}(t) - cbn_{M_1;M_2}(t) = |\tau_{M_2^{\star\bullet\rightarrow\circ\bullet}}(\tau_{M_1^{\rightarrow\star\bullet}}(t))|_\circ - |\tau_{M_2^{\star\bullet\rightarrow\circ\bullet}}(\tau_{M_1^{\rightarrow\star\bullet}}(t))|_\bullet$$

for every $t \in T_\Sigma$. The only possibility for $\tau_{M_2^{\star\bullet\rightarrow\circ\bullet}}(\tau_{M_1^{\rightarrow\star\bullet}}(t))$ to contain $\circ$-symbols is via applications of $\star$-rules of $M_2^{\star\bullet\rightarrow\circ\bullet}$. If M_1 is a tdtt, then $\tau_{M_1^{\rightarrow\star\bullet}}(t)$ contains no $\star$-symbols because there are no context variables in M_1's rule right-hand sides in this case. If M_2 is a tdtt, then the $\star$-rules of $M_2^{\star\bullet\rightarrow\circ\bullet}$ produce no $\circ$-symbols at all because $G = G^{(1)}$ in this case. In both cases we obtain $|\tau_{M_2^{\star\bullet\rightarrow\circ\bullet}}(\tau_{M_1^{\rightarrow\star\bullet}}(t))|_\circ = 0$. Since $|\tau_{M_2^{\star\bullet\rightarrow\circ\bullet}}(\tau_{M_1^{\rightarrow\star\bullet}}(t))|_\bullet$ is always nonnegative, the theorem follows. □

4.6 Unifying the treatment of ∘- and •-symbols

For the case that neither of the two mtts to be composed is a tdtt, we have—through Lemma 4.18—reduced the efficiency analysis problem to comparing the numbers of $\circ$- and $\bullet$-symbols in the final outputs produced by $M_1^{\rightarrow\star\bullet}$ and $M_2^{\star\bullet\rightarrow\circ\bullet}$. The most general and exact approach would be to try developing an algorithm that decides for a given pair of mtts M_1 and M_2 whether for every input the number of $\circ$-symbols in the output of $M_1^{\rightarrow\star\bullet}$ and $M_2^{\star\bullet\rightarrow\circ\bullet}$ is less or equal to the number of $\bullet$-symbols in that same output (plus some small constant). Given that relatively little is known about decidability problems for mtts, this seems a bit overambitious as such a decision procedure might not even exist.[3] Instead, we will make compromises on

[3]The two results that spring to mind as potentially applicable here, namely the (effective) closure of the class of regular tree languages under inverses of macro tree transductions [EV85]

generality and accept pessimistic approximations below for the sake of proceeding with the analysis. The resulting theorems, containing sufficient conditions only, will nevertheless be quite powerful.

While the emergence of •-symbols in the output of $M_1^{\rightarrow\star\bullet}$ and $M_2^{\star\bullet\rightarrow\circ\bullet}$ by adding one atop every symbol from Δ in the intermediate result and then simply reproducing them using rules

$$g\ (\bullet\ v_1)\ z_1 \cdots z_s \rightarrow \bullet\ (g\ v_1\ z_1 \cdots z_s)$$

for $g \in G^{(s+1)}$ is already quite handy, the completely different manner in which ∘-symbols are produced, namely through rules of the form

$$g\ (\star\ v_1)\ z_1 \cdots z_s \rightarrow g\ v_1\ (\circ\ z_1)\ \cdots\ (\circ\ z_s) \tag{3}$$

from ⋆-symbols in the intermediate result, is somewhat unwieldy. Clearly, it would facilitate the desired comparison if ∘-symbols were also to appear directly in right-hand sides of rules of the first mtt and to be reproduced by the second mtt using rules analogous to the ones for •. A first step in that direction is to move all ∘-symbols out of the context parameter positions in ⋆-rules as in the following replacement for (3):

$$g\ (\star\ v_1)\ z_1 \cdots z_s \rightarrow \underbrace{\circ\ (\cdots \circ}_{s \text{ times}}\ (g\ v_1\ z_1 \cdots z_s) \cdots) \tag{4}$$

We have to be careful, though, because this might change the total number of ∘-symbols produced. If, for example, g has the potential to delete one of its context parameters, then the ∘-symbol in that position in the right-hand side of (3) might be discarded while the corresponding outwards moved symbol in (4) is unnecessarily included in the count. Conversely, if g may copy one of its context parameters, then a ∘-symbol that is rightfully counted several times when using (3) would be counted only once when using (4) instead. While overestimating the number of ∘-symbols and thus the number of call-by-name steps performed by the composed program is acceptable when looking for sufficient (rather than necessary) conditions guaranteeing efficiency nondeterioration through our construction, any underestimation in that respect is of course unsafe. It is avoided by performing the replacement of (3) with (4) only if $M_2^{\star\bullet\rightarrow\circ\bullet}$ is context-linear.

The next step is to directly have sequences of ∘-symbols instead of single ⋆-symbols in the right-hand sides of rules of the first mtt and to reproduce those ∘-symbols one by one with an appropriate rule of the second mtt. Since states in G can have different numbers of context parameters, we have to replace every ⋆ by s_{max} times ∘ to be on the safe side, where s_{max} is the maximal such number. This yields for every $f \in F^{(r+1)}$ and $\sigma \in \Sigma^{(p)}$ the following rule:

$$f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r \rightarrow rhs_{M_1,f,\sigma}[\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta][y_k \leftarrow \circ^{s_{max}}\ y_k \mid k \in [r]].$$

and the decidability of the finiteness of ranges of compositions of macro tree transductions [DE98], fail due to the fact that the tree language $\{t \in T_{\{\circ,\bullet,\epsilon^{(0)}\}} \mid |t|_\circ \leq |t|_\bullet\}$ is not regular (by standard pumping arguments).

In the next section our analysis will proceed by finding conditions under which pairs of $\circ$- and $\bullet$-symbols appearing in one and the same annotated rule right-hand side of M_1 can be caused to cancel each other out without compromising safety. To gain flexibility in this endeavor, it would be beneficial if the $\circ$-symbols in question were not always fixed to appear cascaded above all context variables. Without changing the computed annotated intermediate result (as proved in Lemma 4.23 below), this can be achieved by, e.g., moving some of the s_{max} symbols from the top of all occurrences of context variable y_k in every right-hand side for f to the kth context parameter position of every call to f throughout right-hand sides and the initial expression. To provide for maximal freedom of choice, the amount of $\circ$-symbols to be relocated thus can differ for different f and k, as modeled in the following definition through parameterization over the mapping κ.

Definition 4.21 ($M_1^{\to\circ\bullet,\kappa}$ and $M_2^{\circ\bullet\to\circ\bullet}$)
Let $s_{max} = max(rank(G)) - 1$ and let $\kappa : \{(f,k) \mid f \in F^{(r+1)}, k \in [r]\} \longrightarrow \{0,\ldots,s_{max}\}$ be arbitrary. For every $f \in F^{(r+1)}$ and $k \in [r]$ abbreviate $\kappa_{f,k} = \kappa(f,k)$ and $\overline{\kappa}_{f,k} = s_{max} - \kappa(f,k)$. The mtts $M_1^{\to\circ\bullet,\kappa} = (F, \Sigma, \Delta \cup \{\circ,\bullet\}, e_{M_1^{\to\circ\bullet,\kappa}}, R_1^{\to\circ\bullet,\kappa})$ and $M_2^{\circ\bullet\to\circ\bullet} = (G, \Delta \cup \{\circ,\bullet\}, \Omega \cup \{\circ,\bullet\}, e_{M_2}, R_2^{\circ\bullet\to\circ\bullet})$ are given as follows:

$R_1^{\to\circ\bullet,\kappa}$:

$$f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r \to rhs_{M_1,f,\sigma} \qquad \forall f \in F^{(r+1)}, \sigma \in \Sigma^{(p)}$$
$$[\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]$$
$$[f' \twoheadleftarrow f' \cdot (id \times \circ^{\kappa_{f',1}} \cdots \times \circ^{\kappa_{f',r'}}) \mid f' \in F^{(r'+1)}]$$
$$[y_k \leftarrow \circ^{\overline{\kappa}_{f,k}}\ y_k \mid k \in [r]]$$

$$e_{M_1^{\to\circ\bullet,\kappa}} = e_{M_1}[\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta][f' \twoheadleftarrow f' \cdot (id \times \circ^{\kappa_{f',1}} \cdots \times \circ^{\kappa_{f',r'}}) \mid f' \in F^{(r'+1)}]$$

$R_2^{\circ\bullet\to\circ\bullet}$:

$$g\ (\delta\ v_1 \cdots v_q)\ \ z_1 \cdots z_s \to rhs_{M_2,g,\delta} \qquad \forall g \in G^{(s+1)}, \delta \in \Delta^{(q)}$$
$$g\ (\circ\ v_1)\ \ z_1 \cdots z_s \to \circ\ (g\ v_1\ z_1 \cdots z_s) \qquad \forall g \in G^{(s+1)}$$
$$g\ (\bullet\ v_1)\ \ z_1 \cdots z_s \to \bullet\ (g\ v_1\ z_1 \cdots z_s) \qquad \forall g \in G^{(s+1)}$$

Note that $M_2^{\circ\bullet\to\circ\bullet}$ is recursion-linear iff M_2 is; likewise for recursion-nondeletingness, context-linearity and -nondeletingness, and basicness, and hence also for atmostness and atleastness. $\square$

Example 4.22 ($M_{spine}^{\to\circ\bullet,\kappa}$ for $s_{max} = 1$ and a concrete κ)
Choose κ so that $\kappa_{f_1,1} = \kappa_{f_2,2} = 1$ and $\kappa_{f_2,1} = 0$, and hence $\overline{\kappa}_{f_1,1} = \overline{\kappa}_{f_2,2} = 0$ and $\overline{\kappa}_{f_2,1} = 1$. Then, the mtt $M_{spine}^{\to\circ\bullet,\kappa} = (\{f_1^{(2)}, f_2^{(3)}\}, \Sigma_{mon}, \Delta_{tree} \cup \{\circ,\bullet\}, e_{M_{spine}^{\to\circ\bullet,\kappa}}, R_{spine}^{\to\circ\bullet,\kappa})$ has the rules

$$\begin{aligned}
R_{spine}^{\to\circ\bullet,\kappa}:\ & f_1\ (\alpha\ u_1)\ y_1 && \to \bullet\ (\alpha\ (f_1\ u_1\ (\circ\ y_1)))\\
& f_1\ (\beta\ u_1)\ y_1 && \to f_2\ u_1\ y_1\ (\circ\ (\bullet\ (\beta\ (\bullet\ \epsilon))))\\
& f_1\ \epsilon\ \ y_1 && \to y_1\\
& f_2\ (\alpha\ u_1)\ y_1\ y_2 && \to \bullet\ (\alpha\ (f_1\ u_1\ (\circ\ (\bullet\ (\delta\ (\circ\ y_1)\ y_2)))))\\
& f_2\ (\beta\ u_1)\ y_1\ y_2 && \to f_2\ u_1\ (\circ\ y_1)\ (\circ\ (\bullet\ (\beta\ (\bullet\ (\beta\ y_2)))))\\
& f_2\ \epsilon\ \ y_1\ y_2 && \to \bullet\ (\delta\ (\circ\ y_1)\ y_2)
\end{aligned}$$

and the initial expression $e_{M_{spine}^{\to\circ\bullet,\kappa}} = f_1\ u_1\ (\circ\ (\bullet\ \epsilon))$. $\square$

To show that—irrespective of the choice of κ—the difference between the numbers of ∘- and •-symbols in the output computed by $M_1^{\to\circ\bullet,\kappa}$ and $M_2^{\circ\bullet\to\circ\bullet}$ for context-linear M_2 is greater or equal to the corresponding difference in the output produced by $M_1^{\to\star\bullet}$ and $M_2^{\star\bullet\to\circ\bullet}$, we first need two auxiliary lemmata.

Lemma 4.23 (auxiliary)
For every $\kappa : \{(f,k) \mid f \in F^{(r+1)}, k \in [r]\} \longrightarrow \{0,\ldots,s_{max}\}$ and $t \in T_\Sigma$:

$$\tau_{M_1^{\to\circ\bullet,\kappa}}(t) = \tau_{M_1^{\to\star\bullet}}(t)[\star \twoheadleftarrow \circ^{s_{max}}]. \quad \square$$

The proof (of a more general statement by simultaneous induction over the structure of elements of T_Σ, using a nested induction over the structure of right-hand sides of rules of—and of the initial expression of—M_1, annotated using $\cdot[\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]$) can be found in the appendix. The key observation is that when a call $f\ t\ (\circ^{\kappa_{f,1}}\ \theta_1)\ \cdots\ (\circ^{\kappa_{f,r}}\ \theta_r)$ is reduced using a rule from $R_1^{\to\circ\bullet,\kappa}$, additional ∘-symbols are put on top of each context parameter by applying $\circ^{\overline{\kappa}_{f,k}}$ to $(\circ^{\kappa_{f,k}}\ \theta_k)$ for every $k \in [r]$. Since $\overline{\kappa}_{f,k} + \kappa_{f,k} = s_{max}$ for every $k \in [r]$, this corresponds to the $\circ^{s_{max}}$ substituted eventually for the single $\star$-symbol put onto each context parameter when reducing a call to f on t using a rule from $R_1^{\to\star\bullet}$.

Lemma 4.24 (auxiliary)
If M_2 is context-linear, then for every $t' \in T_{\Delta\cup\{\star,\bullet\}}$:

$$|\tau_{M_2^{\star\bullet\to\circ\bullet}}(t')|_{\circ-\bullet} \leq |\tau_{M_2^{\circ\bullet\to\circ\bullet}}(t'[\star \twoheadleftarrow \circ^{s_{max}}])|_{\circ-\bullet}.$$

If additionally M_2 is context-nondeleting and $G = G^{(s_{max}+1)}$, then equality holds. $\square$

The proof (of a more general statement by simultaneous induction over the structure of elements of $T_{\Delta\cup\{\star,\bullet\}}$, using a nested induction over the structure of right-hand sides of rules of—and of the initial expression of—M_2, as well as Lemmata 2.8 and 2.9 for the mtt M_2) can be found in the appendix. The key idea is to quantify how often the ∘-symbols produced in a step

$$g\ (\star\ t'_1)\ \varrho_1 \cdots \varrho_s \Rightarrow_{R_2^{\star\bullet\to\circ\bullet}} g\ t'_1\ (\circ\ \varrho_1)\ \cdots\ (\circ\ \varrho_s)$$

will appear in the final output in terms of the number of occurrences of $z_1,\ldots,z_s$ in

$$nf(\Rightarrow_{R_2}, g\ t'_1[\star,\bullet \twoheadleftarrow id, id]\ z_1 \cdots z_s)\,,$$

which can be approximated using Lemma 2.9, and to compare this to the number of ∘-symbols produced when reducing a call to g on $(\star\ t'_1)[\star \twoheadleftarrow \circ^{s_{max}}]$ using (s_{max} times) the rule for g at $\circ$ in $R_2^{\circ\bullet\to\circ\bullet}$.

Note that in Lemma 4.24 our count gets imprecise for the first time through the use of an inequation. This stems from the (potential) proliferation of ∘-symbols for the sake of advancing in the analysis by forging the annotation scheme into a more tractable form, as discussed at the beginning of this section. To keep some grip on imprecisions introduced by overestimating the number of ∘-symbols—or underestimating the number of •-symbols—, we will make a habit of recording additional

conditions under which given approximations are exact, thus developing a sense of how frequently the pessimistic approximations will manifest in practice. For only showing that a composed program is more efficient than the original one, a count based on inequations (in the proper direction) is of course sufficient. The primary concern when evaluating the usefulness of sufficient conditions is then how many typical examples they cover, and in Subsection 7.5.2 we will argue that our efficiency analysis fares pretty well in this respect.

Simply plugging Lemmata 4.23 and 4.24 together, we easily obtain the following (cf. the appendix for details).

Lemma 4.25 ($M_1^{\to\circ\bullet,\kappa}$ and $M_2^{\circ\bullet\to\circ\bullet}$ compute a safe approximation)
For every $\kappa : \{(f,k) \mid f \in F^{(r+1)}, k \in [r]\} \longrightarrow \{0,\ldots,s_{max}\}$, if M_2 is context-linear, then

$$\tau_{M_1^{\to\star\bullet}} ; \tau_{M_2^{\star\bullet\to\circ\bullet}} ; |\,\cdot\,|_{\circ-\bullet} \leq \tau_{M_1^{\to\circ\bullet,\kappa}} ; \tau_{M_2^{\circ\bullet\to\circ\bullet}} ; |\,\cdot\,|_{\circ-\bullet}.$$

If additionally M_2 is context-nondeleting and $G = G^{(s_{max}+1)}$, then equality holds. □

Example 4.26 (instantiation of Lemma 4.25 to the running example)
To observe the effect of the transition from $M_1^{\to\star\bullet}$ and $M_2^{\star\bullet\to\circ\bullet}$ to $M_1^{\to\circ\bullet,\kappa}$ and $M_2^{\circ\bullet\to\circ\bullet}$ for the running example with the concrete κ from Example 4.22, compare the computation in Figure 4.g to that from Figure 4.f. In particular, note how as a consequence of replacing (3) with (4), $\circ$-symbols have "floated upwards" in the computed output as indicated by the dashed arrows. The difference between the numbers of $\circ$- and $\bullet$-symbols is the same in both outputs, as expected since M_{pfx} is not only context-linear but also context-nondeleting, and since all its states have the same rank as there is only one. □

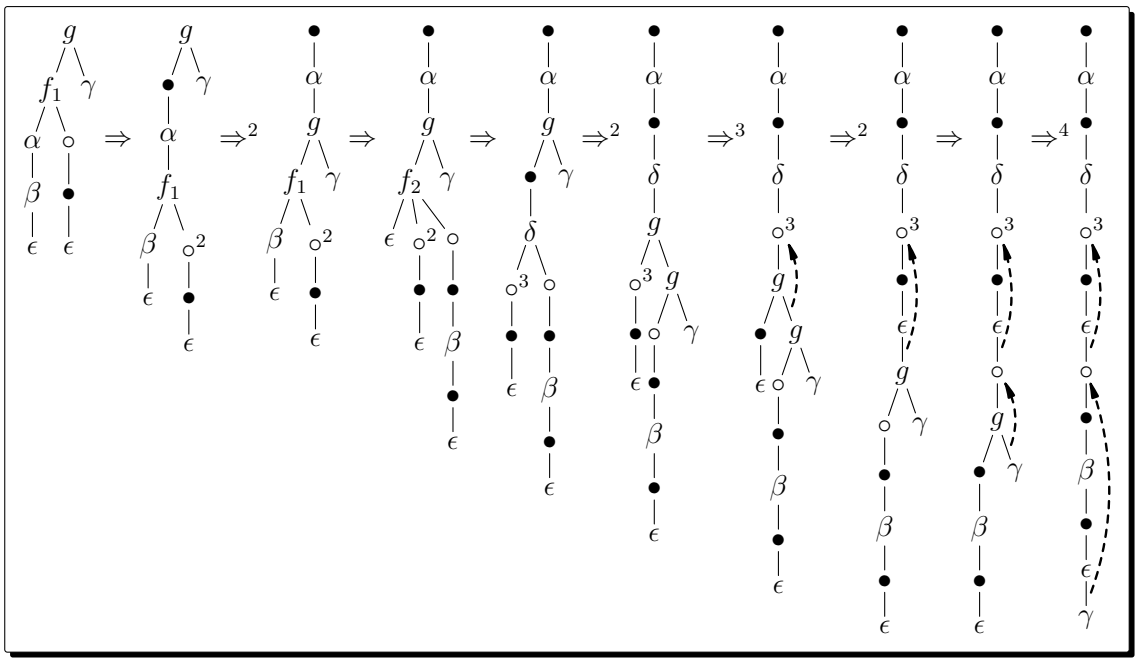

Figure 4.g: Reduction sequence using $\Rightarrow_{R_{spine}^{\to\circ\bullet,\kappa} \cup R_{pfx}^{\circ\bullet\to\circ\bullet}}$

4.7 Moving •-symbols to cancel out ∘-symbols

Our ultimate goal is now to put a constant upper bound, for every input, on the difference between the numbers of ∘- and •-symbols reproduced by $M_2^{\circ\bullet\to\circ\bullet}$ from an intermediate result produced by some $M_1^{\to\circ\bullet,\kappa}$. That could in principle be achieved similarly as in the proof of Theorem 4.20 by correspondingly bounding only the number of ∘-symbols and using that the subtracted number of •-symbols is always nonnegative. But clearly it is unlikely that there is indeed a constant upper bound for the number of ∘-symbols because the number of applications of rules of $M_1^{\to\circ\bullet,\kappa}$ in whose right-hand sides such symbols in general appear depends on the input, and so consequently does the number of ∘-symbols in the final output. We would stand much better chances if all ∘-symbols were eliminated from the rules of the mtt producing the annotated intermediate result. Fortunately, we can make this happen by continuing to manipulate the first mtt's annotation, as long as this does not lead to an underestimation of the difference between the numbers of ∘- and •-symbols. One obviously safe manipulation is to once again remove pairs of adjacent ∘- and •-symbols. If such symbols do not appear next to each other, we can attempt to first bring them closer together by moving either kind of symbols. The following definition lists the sensible choices for moving •-symbols around in annotated right-hand sides of rules of the first mtt.[4]

Definition 4.27 (symmetric up- and down-rules for •)
For every $\delta \in \Delta^{(q)}$ and $j \in [q]$ rewrite rules $\downarrow_{\delta,j}$ and $\uparrow_{\delta,j}$, and for every $f \in F^{(r+1)}$ and $k \in [r]$ rewrite rules $\downarrow_{f,k}$ and $\uparrow_{f,k}$ are given as follows:

$$\begin{array}{lll}
\downarrow_{\delta,j} & : \bullet\ (\delta\ v_1 \cdots v_q) & \to \delta\ v_1 \cdots (\bullet\ v_j) \cdots v_q \\
\uparrow_{\delta,j} & : \delta\ v_1 \cdots (\bullet\ v_j) \cdots v_q & \to \bullet\ (\delta\ v_1 \cdots v_q) \\
\downarrow_{f,k} & : \bullet\ (f\ u\ v_1 \cdots v_r) & \to f\ u\ v_1 \cdots (\bullet\ v_k) \cdots v_r \\
\uparrow_{f,k} & : f\ u\ v_1 \cdots (\bullet\ v_k) \cdots v_r & \to \bullet\ (f\ u\ v_1 \cdots v_r)
\end{array}$$ □

Such ↑- and ↓-rules may not be applied unconstrained. If, for example, $M_2^{\circ\bullet\to\circ\bullet}$ is not recursion-linear, then the application of a $\downarrow_{\delta,j}$-rule to move a •-symbol below an output symbol in the right-hand side of some rule of the first mtt—and as a consequence also in the annotated intermediate result—can lead to that •-symbol being counted more often after the manipulation than it was previously, in case the jth recursion variable occurs more than once in some δ-rule of M_2. That is, in the presence of a rule

$$g\ (\delta\ v_1\ v_2) \to \omega\ (g\ v_1)\ (g\ v_1)$$

of M_2 an application of the $\downarrow_{\delta,1}$-rule can have the effect that an $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$-reduction

$$g\ (\bullet\ (\delta\ \theta_1\ \theta_2)) \Rightarrow \bullet\ (g\ (\delta\ \theta_1\ \theta_2)) \Rightarrow \bullet\ (\omega\ (g\ \theta_1)\ (g\ \theta_1))$$

for some $\theta_1, \theta_2 \in T_{\Delta\cup\{\circ,\bullet\}}$ is replaced by the following:

[4] It turned out that dually moving also ∘-symbols would not add strength to the analysis.

$$g\ (\delta\ (\bullet\ \theta_1)\ \theta_2) \Rightarrow \omega\ (g\ (\bullet\ \theta_1))\ (g\ (\bullet\ \theta_1)) \Rightarrow^2 \omega\ (\bullet\ (g\ \theta_1))\ (\bullet\ (g\ \theta_1))\,,$$

thus overestimating the number of $\bullet$-symbols. In addition to avoiding unlegitimate duplication of a $\bullet$-symbol through several recursive calls on the subtree into which it was moved, also more subtle ways of duplication via context parameters must be barred. If, for example, M_2 has a rule

$$g\ (\delta\ v_1\ v_2) \to g'\ v_1\ (g\ v_2) \tag{5}$$

then an application of the $\downarrow_{\delta,2}$-rule can have the effect that an $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$-reduction

$$g\ (\bullet\ (\delta\ \theta_1\ \theta_2)) \Rightarrow \bullet\ (g\ (\delta\ \theta_1\ \theta_2)) \Rightarrow \bullet\ (g'\ \theta_1\ (g\ \theta_2))$$

for some $\theta_1, \theta_2 \in T_{\Delta\cup\{\circ,\bullet\}}$ is replaced by the following:

$$g\ (\delta\ \theta_1\ (\bullet\ \theta_2)) \Rightarrow g'\ \theta_1\ (g\ (\bullet\ \theta_2)) \Rightarrow g'\ \theta_1\ (\bullet\ (g\ \theta_2)).$$

If g' happens to copy its context parameter, then one $\bullet$-symbol is turned into more than one. To avoid this problem, we can either outlaw nested calls such as the one in the right-hand side of (5) or require that context parameters are never copied. Syntactic restrictions of mtts to these effects are (either of) basicness and context-linearity from Definition 2.5; the proper combination with recursion-linearity to guarantee a safe application of $\downarrow_{\delta,j}$-rules is given by what we dubbed **atmost** there. Completely dually, M_2 must be **atleast** in order to apply $\uparrow_{\delta,j}$-rules safely. In the case of $\downarrow_{f,k}$- and $\uparrow_{f,k}$-rules for a state $f \in F$, we additionally need to require context-linearity and -nondeletingness, respectively, of the first mtt to prevent unlegitimate proliferation of $\bullet$-symbols in the annotated intermediate result already. The following definition captures the gained insights by specifying a permissible subset $\mathcal{E}_{M_1,M_2}$ of the possible rewrite rules for moving $\bullet$-symbols and eliminating $\circ$-symbols.

Definition 4.28 ($\mathcal{E}_{M_1,M_2}$)
The rewrite system $\mathcal{E}_{M_1,M_2}$ contains the following rules:

- $\bullet\ (\circ\ v) \to v$,
- $\circ\ (\bullet\ v) \to v$,
- $\downarrow_{\delta,j}$ for every $\delta \in \Delta^{(q)}$ and $j \in [q]$ if M_2 is atmost,
- $\uparrow_{\delta,j}$ for every $\delta \in \Delta^{(q)}$ and $j \in [q]$ if M_2 is atleast,
- $\downarrow_{f,k}$ for every $f \in F^{(r+1)}$ and $k \in [r]$ if M_1 is context-linear and M_2 is atmost, and
- $\uparrow_{f,k}$ for every $f \in F^{(r+1)}$ and $k \in [r]$ if M_1 is context-nondel. and M_2 is atleast. □

Example 4.29 ($\mathcal{E}_{M_{spine},M_{pfx}}$, cf. Example 2.2)
Since M_{spine} is context-linear and context-nondeleting and M_{pfx} is atmost and atleast, the rewrite system $\mathcal{E}_{M_{spine},M_{pfx}}$ contains all the following rules:

$$\begin{array}{lll}
 & \bullet\ (\circ\ v) & \to v \\
 & \circ\ (\bullet\ v) & \to v \\
\downarrow_{\delta,1} & \bullet\ (\delta\ v_1\ v_2) & \to \delta\ (\bullet\ v_1)\ v_2 \\
\downarrow_{\delta,2} & \bullet\ (\delta\ v_1\ v_2) & \to \delta\ v_1\ (\bullet\ v_2) \\
\downarrow_{\alpha,1} & \bullet\ (\alpha\ v_1) & \to \alpha\ (\bullet\ v_1) \\
\downarrow_{\beta,1} & \bullet\ (\beta\ v_1) & \to \beta\ (\bullet\ v_1) \\
\uparrow_{\delta,1} & \delta\ (\bullet\ v_1)\ v_2 & \to \bullet\ (\delta\ v_1\ v_2) \\
\uparrow_{\delta,2} & \delta\ v_1\ (\bullet\ v_2) & \to \bullet\ (\delta\ v_1\ v_2)
\end{array}
\qquad
\begin{array}{lll}
\uparrow_{\alpha,1} & \alpha\ (\bullet\ v_1) & \to \bullet\ (\alpha\ v_1) \\
\uparrow_{\beta,1} & \beta\ (\bullet\ v_1) & \to \bullet\ (\beta\ v_1) \\
\downarrow_{f_1,1} & \bullet\ (f_1\ u\ v_1) & \to f_1\ u\ (\bullet\ v_1) \\
\downarrow_{f_2,1} & \bullet\ (f_2\ u\ v_1\ v_2) & \to f_2\ u\ (\bullet\ v_1)\ v_2 \\
\downarrow_{f_2,2} & \bullet\ (f_2\ u\ v_1\ v_2) & \to f_2\ u\ v_1\ (\bullet\ v_2) \\
\uparrow_{f_1,1} & f_1\ u\ (\bullet\ v_1) & \to \bullet\ (f_1\ u\ v_1) \\
\uparrow_{f_2,1} & f_2\ u\ (\bullet\ v_1)\ v_2 & \to \bullet\ (f_2\ u\ v_1\ v_2) \\
\uparrow_{f_2,2} & f_2\ u\ v_1\ (\bullet\ v_2) & \to \bullet\ (f_2\ u\ v_1\ v_2)
\end{array}
\qquad \square$$

We will later allow the rewrite rules from $\mathcal{E}_{M_1,M_2}$ to be applied arbitrarily often to rule right-hand sides, but also to the initial expression, of annotated versions of M_1. However, for the sake of proving that this indeed does not lead to an underestimation of the difference between the numbers of ∘- and •-symbols in the final output, it is more convenient to first consider at most one rewrite step on the initial expression and each right-hand side. This is described by a relation $\rightsquigarrow_{\mathcal{E}_{M_1,M_2}}$ on a set containing all $M_1^{\to\circ\bullet,\kappa}$ and all mtts possibly obtained from them via the rewrite rules in question.

Definition 4.30 ($\mathcal{M}_1^{\to\circ\bullet}$ and $\rightsquigarrow_{\mathcal{E}_{M_1,M_2}} \subseteq \mathcal{M}_1^{\to\circ\bullet} \times \mathcal{M}_1^{\to\circ\bullet}$)
Let $\mathcal{M}_1^{\to\circ\bullet}$ be the set of all mtts $M = (F, \Sigma, \Delta \cup \{\circ, \bullet\}, e, R)$ for which $e[\circ, \bullet \twoheadleftarrow id, id] = e_{M_1}$ and $rhs_{M,f,\sigma}[\circ, \bullet \twoheadleftarrow id, id] = rhs_{M_1,f,\sigma}$ for every $f \in F$ and $\sigma \in \Sigma$. Note that every $M_1^{\to\circ\bullet} \in \mathcal{M}_1^{\to\circ\bullet}$ is context-linear iff M_1 is; likewise for context-nondeletingness. For mtts $M_1^{\to\circ\bullet} = (F, \Sigma, \Delta \cup \{\circ, \bullet\}, e_{M_1^{\to\circ\bullet}}, R_1^{\to\circ\bullet})$ and $M_1^{\to\circ\bullet'} = (F, \Sigma, \Delta \cup \{\circ, \bullet\}, e_{M_1^{\to\circ\bullet'}}, R_1^{\to\circ\bullet'})$ in $\mathcal{M}_1^{\to\circ\bullet}$ we write $M_1^{\to\circ\bullet} \rightsquigarrow_{\mathcal{E}_{M_1,M_2}} M_1^{\to\circ\bullet'}$ if $e_{M_1^{\to\circ\bullet}} \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} e_{M_1^{\to\circ\bullet'}}$ and $rhs_{M_1^{\to\circ\bullet},f,\sigma} \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} rhs_{M_1^{\to\circ\bullet'},f,\sigma}$ for every $f \in F$ and $\sigma \in \Sigma$. $\square$

Example 4.31 (rewriting $M_{spine}^{\to\circ\bullet,\kappa}$ from Example 4.22 with $\rightsquigarrow_{\mathcal{E}_{M_{spine},M_{pfx}}}$)
By rewriting the initial expression and each right-hand side of a rule of $M_{spine}^{\to\circ\bullet,\kappa}$ with at most one step of $\Rightarrow_{\mathcal{E}_{M_{spine},M_{pfx}}}$, we can, e.g., obtain $M_{spine}^{\to\circ\bullet,\kappa} \rightsquigarrow_{\mathcal{E}_{M_{spine},M_{pfx}}} M_{spine}^{\to\circ\bullet'} = (\{f_1^{(2)}, f_2^{(3)}\}, \Sigma_{mon}, \Delta_{tree} \cup \{\circ, \bullet\}, f_1\ u_1\ (\circ\ (\bullet\ \epsilon)), R_{spine}^{\to\circ\bullet'})$ with rules

$$\begin{array}{lll}
R_{spine}^{\to\circ\bullet'}: & f_1\ (\alpha\ u_1)\ y_1 & \to \bullet\ (\alpha\ (f_1\ u_1\ (\circ\ y_1))) \\
 & f_1\ (\beta\ u_1)\ y_1 & \to f_2\ u_1\ y_1\ (\beta\ (\bullet\ \epsilon)) \\
 & f_1\ \epsilon\ y_1 & \to y_1 \\
 & f_2\ (\alpha\ u_1)\ y_1\ y_2 & \to \bullet\ (\alpha\ (f_1\ u_1\ (\circ\ (\delta\ (\bullet\ (\circ\ y_1))\ y_2)))) \\
 & f_2\ (\beta\ u_1)\ y_1\ y_2 & \to f_2\ u_1\ (\circ\ y_1)\ (\circ\ (\bullet\ (\beta\ (\bullet\ (\beta\ y_2))))) \\
 & f_2\ \epsilon\ y_1\ y_2 & \to \delta\ (\bullet\ (\circ\ y_1))\ y_2
\end{array}
\qquad \square$$

To show that rewriting an annotated version of M_1 according to $\rightsquigarrow_{\mathcal{E}_{M_1,M_2}}$ does not lead to a decrease in the difference between the numbers of ∘- and •-symbols reproduced by $M_2^{\circ\bullet\to\circ\bullet}$ from the annotated intermediate result, we first prove two lemmata.

The first one captures how the omission of $\bullet$-symbols from an annotated intermediate result is reflected in the final output produced by a state of an atmost and of an atleast M_2, respectively. In fact, the properties given here motivated the choice of names "atmost" and "atleast". The different orders in which θ and θ' appear in the left- and right-hand sides of the given inequations might appear counter-intuitive on first sight. This is caused by the fact that $\bullet$-symbols are counted negatively by $|\,.\,|_{\circ-\bullet}$ but positively by $|\,.\,|_\bullet$.

Lemma 4.32 (essential properties of atmost and atleast mtts, resp.)
For every $\theta, \theta' \in T_{\Delta\cup\{\circ,\bullet\}}$ with $\theta \Rightarrow^*_{\{\bullet\, v \to v\}} \theta'$, for every $g \in G^{(s+1)}$ and $\eta_1, \ldots, \eta_s \in T_{G\cup\Omega\cup\Delta\cup\{\circ,\bullet\}}$:

1. if M_2 is atmost, then
   $|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta'\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} \leq |\theta|_\bullet - |\theta'|_\bullet$
2. if M_2 is atleast, then
   $|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta'\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$
   $\geq |\theta|_\bullet - |\theta'|_\bullet$. $\square$

The proof (of more general statements by simultaneous induction over the structure of elements of $T_{\Delta\cup\{\circ,\bullet\}}$, using a nested induction over the structure of right-hand sides of rules of M_2, as well as Lemmata 2.8 and 2.9 for the mtt $M_2^{\circ\bullet\to\circ\bullet}$) can be found in the appendix. It uses $|\,.\,|_{\bullet-\circ} = |\,.\,|_\bullet - |\,.\,|_\circ$ instead of $|\,.\,|_{\circ-\bullet}$ to later enable a "recycling" of the performed inductions for the proof of Lemma 4.38. Another dimension of proof reuse is that again, as in the case of Lemma 2.9, we only need to explicitly prove the first statement, whereas the proof of the second one is completely dual (in the sense in which the definition of "atleast" is dual to that of "atmost").

The previous lemma is used in the following one to establish in which sense the $\uparrow$- and $\downarrow$-rules we have chosen to put into $\mathcal{E}_{M_1,M_2}$ are permissible.

Lemma 4.33 (essential properties of the rewrite system $\mathcal{E}_{M_1,M_2}$)
For every $g \in G^{(s+1)}$ and $\eta_1, \ldots, \eta_s \in T_{G\cup\Omega\cup\Delta\cup\{\circ,\bullet\}}$:

1. for every $\delta \in \Delta^{(q)}$, $j \in [q]$, and $\theta_1, \ldots, \theta_q \in T_{\Delta\cup\{\circ,\bullet\}}$:
   (a) if $\downarrow_{\delta,j} \in \mathcal{E}_{M_1,M_2}$, then
   $|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\delta\ \theta_1 \cdots \theta_q)\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$
   $\leq 1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\delta\ \theta_1 \cdots (\bullet\ \theta_j) \cdots \theta_q)\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$
   (b) if $\uparrow_{\delta,j} \in \mathcal{E}_{M_1,M_2}$, then
   $|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\delta\ \theta_1 \cdots \theta_q)\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$
   $\geq 1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\delta\ \theta_1 \cdots (\bullet\ \theta_j) \cdots \theta_q)\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$
2. for every $M_1^{\to\circ\bullet} = (F, \Sigma, \Delta\cup\{\circ,\bullet\}, e_{M_1^{\to\circ\bullet}}, R_1^{\to\circ\bullet}) \in \mathcal{M}_1^{\to\circ\bullet}$, $f \in F^{(r+1)}$, $k \in [r]$, $t \in T_\Sigma$, and $\theta_1, \ldots, \theta_r \in T_{\Delta\cup\{\circ,\bullet\}}$:
   (a) if $\downarrow_{f,k} \in \mathcal{E}_{M_1,M_2}$, then
   $|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ \theta_1 \cdots \theta_r)\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$
   $\leq 1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ \theta_1 \cdots (\bullet\ \theta_k) \cdots \theta_r)\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$

(b) if $\uparrow_{f,k} \in \mathcal{E}_{M_1,M_2}$, then
$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ \theta_1 \cdots \theta_r)\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$
$\geq 1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ \theta_1 \cdots (\bullet\ \theta_k) \cdots \theta_r)\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$.

Proof
Since from $\downarrow_{\delta,j} \in \mathcal{E}_{M_1,M_2}$ follows that M_2 is atmost and from $\uparrow_{\delta,j} \in \mathcal{E}_{M_1,M_2}$ follows that M_2 is atleast, the points 1a and 1b follow from Lemma 4.32 with $\theta = \delta\ \theta_1 \cdots (\bullet\ \theta_j) \cdots \theta_q$ and $\theta' = \delta\ \theta_1 \cdots \theta_q$, using that $\theta \Rightarrow_{\{\bullet\, v \to v\}} \theta'$ and $|\theta|_\bullet - |\theta'|_\bullet = 1$. Since from $\downarrow_{f,k} \in \mathcal{E}_{M_1,M_2}$ follows that M_1 is context-linear and M_2 is atmost and from $\uparrow_{f,k} \in \mathcal{E}_{M_1,M_2}$ follows that M_1 is context-nondeleting and M_2 is atleast, the points 2a and 2b follow from Lemma 4.32 with $\theta = nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ \theta_1 \cdots (\bullet\ \theta_k) \cdots \theta_r)$ and $\theta' = nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ \theta_1 \cdots \theta_r)$, using that:

1. $\theta \Rightarrow^*_{\{\bullet\, v \to v\}} \theta'$,
2. $|\theta|_\bullet - |\theta'|_\bullet \leq 1$ if M_1, and hence also $M_1^{\to\circ\bullet}$, is context-linear, and
3. $|\theta|_\bullet - |\theta'|_\bullet \geq 1$ if M_1, and hence also $M_1^{\to\circ\bullet}$, is context-nondeleting.

These auxiliary statements, in turn, follow immediately from the following equalities for θ and θ' (which hold because the $\theta_1, \ldots, \theta_r \in T_{\Delta\cup\{\circ,\bullet\}}$ are normal forms w.r.t. $\Rightarrow_{R_1^{\to\circ\bullet}}$):

$$\begin{aligned}\theta &= nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ y_1 \cdots y_r)[y_1, \ldots, y_r \leftarrow \theta_1, \ldots, \bullet\ \theta_k, \ldots, \theta_r]\\ \theta' &= nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ y_1 \cdots y_r)[y_1, \ldots, y_r \leftarrow \theta_1, \ldots, \theta_r]\end{aligned}$$

and Lemma 2.9 for the mtt $M_1^{\to\circ\bullet}$. □

Now, we are in a position to prove that rewriting according to $\rightsquigarrow_{\mathcal{E}_{M_1,M_2}}$ does not lead to an underestimation of the difference between the numbers of ∘- and •-symbols.

Lemma 4.34 (applying $\rightsquigarrow_{\mathcal{E}_{M_1,M_2}}$ leads to a safe approximation)
If $M_1^{\to\circ\bullet} \rightsquigarrow_{\mathcal{E}_{M_1,M_2}} M_1^{\to\circ\bullet'}$, then

$$\tau_{M_1^{\to\circ\bullet}}\,;\tau_{M_2^{\circ\bullet\to\circ\bullet}}\,;|\,\blacksquare\,|_{\circ-\bullet} \leq \tau_{M_1^{\to\circ\bullet'}}\,;\tau_{M_2^{\circ\bullet\to\circ\bullet}}\,;|\,\blacksquare\,|_{\circ-\bullet}.$$

If for every ↓- or ↑-rule used for rewriting the initial expression or right-hand sides of rules of $M_1^{\to\circ\bullet}$, $\mathcal{E}_{M_1,M_2}$ also contains the symmetric rule, then equality holds. □

The proof (of a more general statement by simultaneous induction over the structure of elements of T_Σ, using a nested simultaneous induction over the structure of right-hand sides of rules of—and of the initial expression of—$M_1^{\to\circ\bullet}$, nested inside which is an induction over the structure of right-hand sides of rules of—and of the initial expression of—M_2, as well as Lemma 4.33) can be found in the appendix. At its heart is an analysis of the different ways in which a rule right-hand side or initial expression of $M_1^{\to\circ\bullet}$ can be rewritten (or not) using $\Rightarrow^?_{\mathcal{E}_{M_1,M_2}}$. This leads to eleven cases to consider, but a lot of reuse is possible in resolving the associated proof obligations. Another interesting reuse aspect is the proof of the statement about equality in case that only ↓- or ↑-rules are used whose symmetric version is also contained in $\mathcal{E}_{M_1,M_2}$.

Example 4.35 (instantiation of Lemma 4.34 to the running example)
To observe the effect of the concrete application of $\rightsquigarrow_{\mathcal{E}_{M_{spine},M_{pfx}}}$ in Example 4.31, compare the computation in Figure 4.h to that from Figure 4.g. In particular, note how as a consequence of rewriting the right-hand side of the rule for f_2 at ϵ one $\bullet$-symbol has "floated downwards" in the computed output as indicated by the dashed arrow, and how the removal of adjacent $\circ$- and $\bullet$-symbols in the right-hand side of the rule for f_1 at β has led to the elimination of one such pair also from the computed output at the $\times$-marked position. The difference between the numbers of $\circ$- and $\bullet$-symbols is the same in both outputs, as expected since $\mathcal{E}_{M_{spine},M_{pfx}}$ contains all potential $\downarrow$- and $\uparrow$-rules. $\square$

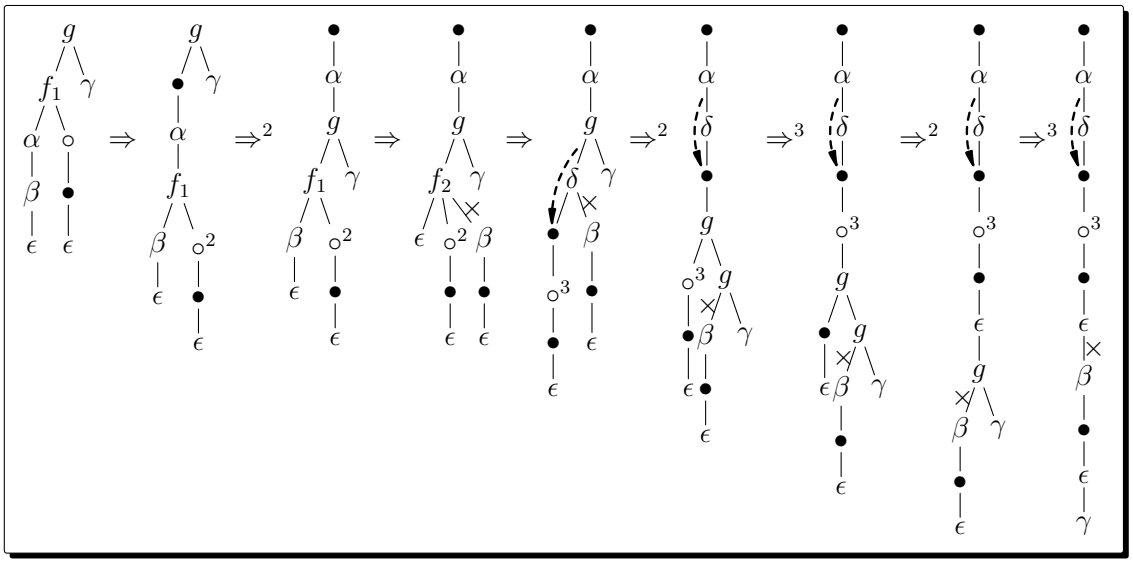

Figure 4.h: Reduction sequence using $\Rightarrow_{R^{\rightarrow\circ\bullet\prime}_{spine}\cup R^{\circ\bullet\rightarrow\circ\bullet}_{pfx}}$

4.8 Constantly bounding the number of $\circ$-symbols

As motivated at the beginning of the previous section, we want to manipulate some $M_1^{\rightarrow\circ\bullet,\kappa}$ (by eliminating $\circ$-symbols via $\rightsquigarrow_{\mathcal{E}_{M_1,M_2}}$) until we obtain an $M_1^{\rightarrow\circ\bullet} \in \mathcal{M}_1^{\rightarrow\circ\bullet}$ for which we can put a constant upper bound on $|\tau_{M_2^{\circ\bullet\rightarrow\circ\bullet}}(\tau_{M_1^{\rightarrow\circ\bullet}}(t))|_\circ$ for every $t \in T_\Sigma$. The most natural approach to establishing such an upper bound is to first consider conditions under which the number of $\circ$-symbols in the intermediate result $\tau_{M_1^{\rightarrow\circ\bullet}}(t)$ is constantly bounded. Requiring that no $\circ$-symbols occur anymore in right-hand sides of rules of $M_1^{\rightarrow\circ\bullet}$ is not quite enough because there might still be $\circ$-symbols present in its initial expression. In that case, we additionally impose context-linearity to be sure that such $\circ$-symbols are not duplicated during the computation.

Lemma 4.36 (bounding the number of ∘-symbols produced by $M_1^{\to\circ\bullet}$)
Let $M_1^{\to\circ\bullet} = (F, \Sigma, \Delta \cup \{\circ, \bullet\}, e_{M_1^{\to\circ\bullet}}, R_1^{\to\circ\bullet}) \in \mathcal{M}_1^{\to\circ\bullet}$ be such that the right-hand sides of the rules in $R_1^{\to\circ\bullet}$ contain no ∘-symbols. If $|e_{M_1^{\to\circ\bullet}}|_\circ = 0$ or M_1 is context-linear, then for every $t \in T_\Sigma$:

$$|\tau_{M_1^{\to\circ\bullet}}(t)|_\circ \leq |e_{M_1^{\to\circ\bullet}}|_\circ.$$

If $|e_{M_1^{\to\circ\bullet}}|_\circ = 0$ or additionally M_1 is context-nondeleting, then equality holds. □

The proof (of a more general statement by simultaneous induction over the structure of elements of T_Σ, using a nested induction over the structure of right-hand sides of rules of—and of the initial expression of—$M_1^{\to\circ\bullet}$, as well as Lemmata 2.8 and 2.9 for the mtt $M_1^{\to\circ\bullet}$) can be found in the appendix. The key observation is that if the right-hand sides of the rules in $R_1^{\to\circ\bullet}$ contain no ∘-symbols, then any ∘-symbols appearing in the output of a call to a state from F can only originate from copies of that call's context parameters, i.e., we can always decompose as follows:

$$|nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ \theta_1 \cdots \theta_r)|_\circ = \sum_{k\in[r]} |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ y_1 \cdots y_r)|_{y_k} * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \theta_k)|_\circ.$$

Given a constant bound for the number of ∘-symbols in $\tau_{M_1^{\to\circ\bullet}}(t)$, we could obtain a corresponding bound for $\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet}}(t))$ if we knew that $M_2^{\circ\bullet\to\circ\bullet}$ duplicates ∘-symbols found in its input by at most a constant factor. The latter is obviously the case if no such symbols are found in the input at all.

Lemma 4.37 ($M_2^{\circ\bullet\to\circ\bullet}$ does not create ∘-symbols out of nothing)
For every $\theta \in T_{\Delta\cup\{\circ,\bullet\}}$, if $|\theta|_\circ = 0$, then $|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\theta)|_\circ = 0$.

Proof
Since the initial expression $e_{M_2} \in RHS(G, \Omega, V_1, \emptyset)$ of $M_2^{\circ\bullet\to\circ\bullet}$ contains no ∘-symbols, such symbols in the output $\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\theta) = nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, e_{M_2}[v_1 \leftarrow \theta])$ could only originate from right-hand sides of rules in $R_2^{\circ\bullet\to\circ\bullet}$. But the only such right-hand sides containing ∘-symbols are those of the rules at input symbol ∘, which however are never applied if $|\theta|_\circ = 0$. □

If $M_2^{\circ\bullet\to\circ\bullet}$'s input **does** contain ∘-symbols, then we want to be sure that each of them is reproduced in the output at most a bounded number of times. One reason for unbounded reproduction could be recursion-nonlinearity because then a given ∘-symbol could be reached arbitrarily many times depending on its depth in the input tree. Another source for unbounded duplication could be that the output of a call that is nested inside a context parameter position of another call is duplicated when the outer call copies its parameters. This can be avoided by requiring basicness or context-linearity, just as we did in the previous section when discussing conditions for the safe applicability of $\downarrow_{\delta,j}$-rules. In fact, the restriction atmostness is once again what we are looking for. It guarantees that each ∘-symbol found in the input is reproduced in the output at most as many times as the input is plugged into the initial expression of $M_2^{\circ\bullet\to\circ\bullet}$.

Lemma 4.38 (bounding number of copies of ∘-symbols made by $M_2^{\circ\bullet\to\circ\bullet}$)
If M_2 is atmost, then for every $\theta \in T_{\Delta\cup\{\circ,\bullet\}}$:

$$|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\theta)|_\circ \leq |e_{M_2}|_{v_1} * |\theta|_\circ.$$

If additionally M_2 is atleast, then equality holds. □

The proof (by essentially "recycling" the inductions from the proof of Lemma 4.32 and additionally using Lemma 4.37 for $\theta[\circ \leftarrow id]$) can be found in the appendix. The close connection of this lemma to Lemma 4.32, which might not be apparent at first glance, stems from the fact that the omission of ∘-symbols from an input tree for an atmost and for an atleast $M_2^{\circ\bullet\to\circ\bullet}$, respectively, is reflected in the final output just in the same manner as in the case of •-symbols.

4.9 Assembling the call-by-name analysis

Together with Lemma 4.34, the three lemmata from the previous section can under appropriate preconditions be used to safely approximate the difference between the numbers of ∘- and •-symbols reproduced by $M_2^{\circ\bullet\to\circ\bullet}$ from an intermediate result produced by $M_1^{\to\circ\bullet,\kappa}$ for an input tree t in terms of $|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet}}(t))|_\bullet$ and the initial expressions of $M_1^{\to\circ\bullet}$ and $M_2^{\circ\bullet\to\circ\bullet}$ for some $M_1^{\to\circ\bullet}$ obtained from $M_1^{\to\circ\bullet,\kappa}$ via applications of $\rightsquigarrow_{\mathcal{E}_{M_1,M_2}}$. Since the still input-dependent, nonnegative value $|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet}}(t))|_\bullet$ occurs as subtrahend only, it can later be dropped. Alternatively, it could be used to attempt a further analysis by how much the composed program is more efficient than the original one, rather than just whether it is.[5]

Lemma 4.39 (the final puzzle piece)
Let $\kappa : \{(f,k) \mid f \in F^{(r+1)}, k \in [r]\} \longrightarrow \{0,\ldots,s_{max}\}$ and $M_1^{\to\circ\bullet} = (F,\Sigma,\Delta\cup\{\circ,\bullet\}, e_{M_1^{\to\circ\bullet}}, R_1^{\to\circ\bullet}) \in \mathcal{M}_1^{\to\circ\bullet}$ be such that $M_1^{\to\circ\bullet,\kappa} \rightsquigarrow^*_{\mathcal{E}_{M_1,M_2}} M_1^{\to\circ\bullet}$ and the right-hand sides of the rules in $R_1^{\to\circ\bullet}$ contain no ∘-symbols. If $|e_{M_1^{\to\circ\bullet}}|_\circ = 0$ or if M_1 is context-linear and M_2 is atmost, then for every $t \in T_\Sigma$:

$$|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet,\kappa}}(t))|_{\circ-\bullet} \leq |e_{M_2}|_{v_1} * |e_{M_1^{\to\circ\bullet}}|_\circ - |\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet}}(t))|_\bullet.$$

If for every ↓- or ↑-rule used in the rewriting process to obtain $e_{M_1^{\to\circ\bullet}}$ and $R_1^{\to\circ\bullet}$, $\mathcal{E}_{M_1,M_2}$ also contains the symmetric rule, and if $|e_{M_1^{\to\circ\bullet}}|_\circ = 0$ or additionally M_1 is context-nondeleting and M_2 is atleast, then equality holds.

Proof
By repeated applications of Lemma 4.34 and reflexivity and transitivity of $\leq$ we have $\tau_{M_1^{\to\circ\bullet,\kappa}} ; \tau_{M_2^{\circ\bullet\to\circ\bullet}} ; |\,\cdot\,|_{\circ-\bullet} \leq \tau_{M_1^{\to\circ\bullet}} ; \tau_{M_2^{\circ\bullet\to\circ\bullet}} ; |\,\cdot\,|_{\circ-\bullet}$, and hence for every $t \in T_\Sigma$:

$$|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet,\kappa}}(t))|_{\circ-\bullet} \leq |\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet}}(t))|_\circ - |\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet}}(t))|_\bullet.$$

[5] For our running example we will do so in Examples 4.42 and 4.46, but we have not formally pursued this direction in general because it is unclear how an optimizing compiler should benefit from the additional information about the **amount** of improvement. Usually, to establish that there **is** an improvement (or at least nondeterioration) will be enough.

If for every ↓- or ↑-rule used in the rewriting process to obtain $e_{M_1^{\to\circ\bullet}}$ and $R_1^{\to\circ\bullet}$, $\mathcal{E}_{M_1,M_2}$ also contains the symmetric rule, then the three occurrences of $\leq$ above can be replaced by $=$. Now, we reason as follows:

- If $|e_{M_1^{\to\circ\bullet}}|_\circ = 0$, then $|\tau_{M_1^{\to\circ\bullet}}(t)|_\circ = 0$ by Lemma 4.36, and as a consequence we have $|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet}}(t))|_\circ = 0 = |e_{M_2}|_{v_1} * |e_{M_1^{\to\circ\bullet}}|_\circ$ by Lemma 4.37 with $\theta = \tau_{M_1^{\to\circ\bullet}}(t)$.
- If M_1 is context-linear and M_2 is atmost, then:

  $|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet}}(t))|_\circ$
  $\leq$ (by Lemma 4.38 with $\theta = \tau_{M_1^{\to\circ\bullet}}(t)$)
  $|e_{M_2}|_{v_1} * |\tau_{M_1^{\to\circ\bullet}}(t)|_\circ$
  $\leq$ (by Lemma 4.36, using that the factor $|e_{M_2}|_{v_1} \geq 0$)
  $|e_{M_2}|_{v_1} * |e_{M_1^{\to\circ\bullet}}|_\circ$

  If additionally M_1 is context-nondeleting and M_2 is atleast, then Lemmata 4.36 and 4.38 give equalities instead of inequalities in the above calculation. □

Example 4.40 (further rewriting $M_{spine}^{\to\circ\bullet'}$ from Ex. 4.31 with $\leadsto^*_{\mathcal{E}_{M_{spine},M_{pfx}}}$)
By rewriting the initial expression and each right-hand side of a rule of $M_{spine}^{\to\circ\bullet'}$ arbitrarily often with some of the rules given in Example 4.29, we may obtain $M_{spine}^{\to\circ\bullet'} \leadsto^*_{\mathcal{E}_{M_{spine},M_{pfx}}} M_{spine}^{\to\circ\bullet} = (\{f_1^{(2)}, f_2^{(3)}\}, \Sigma_{mon}, \Delta_{tree} \cup \{\circ,\bullet\}, e_{M_{spine}^{\to\circ\bullet}}, R_{spine}^{\to\circ\bullet})$ with $e_{M_{spine}^{\to\circ\bullet}} = f_1\ u_1\ \epsilon$ and rules

$$
\begin{array}{lllll}
R_{spine}^{\to\circ\bullet}: & f_1\ (\alpha\ u_1)\ y_1 \to \alpha\ (f_1\ u_1\ y_1) & & f_2\ (\alpha\ u_1)\ y_1\ y_2 \to \alpha\ (f_1\ u_1\ (\delta\ y_1\ y_2)) \\
& f_1\ (\beta\ u_1)\ y_1 \to f_2\ u_1\ y_1\ (\beta\ (\bullet\ \epsilon)) & & f_2\ (\beta\ u_1)\ y_1\ y_2 \to f_2\ u_1\ y_1\ (\beta\ (\beta\ y_2)) \\
& f_1\ \epsilon \qquad y_1 \to y_1 & & f_2\ \epsilon \qquad y_1\ y_2 \to \delta\ y_1\ y_2
\end{array}
$$

For illustration, Figure 4.i gives a sequence of rewritings performed on the right-hand side of the rule for f_2 at β.

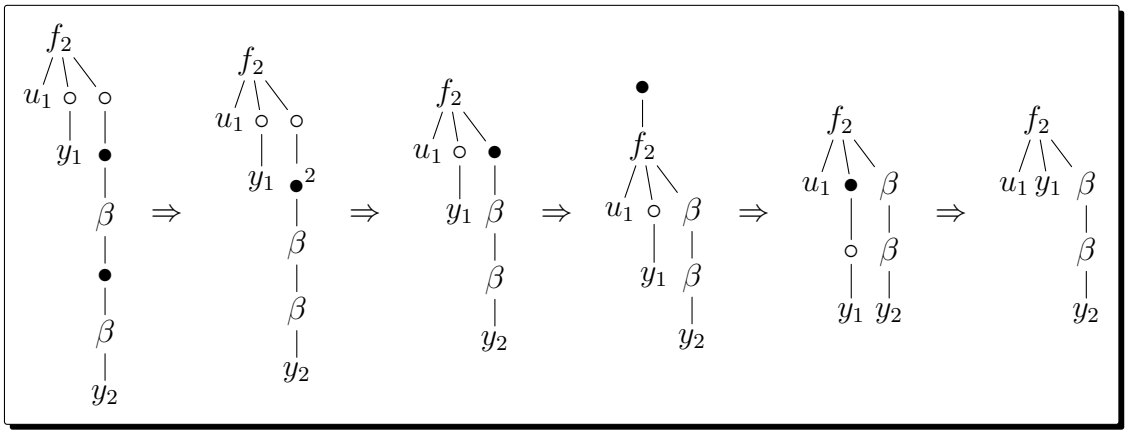

Figure 4.i: Rewriting $rhs_{M_{spine}^{\to\circ\bullet},\kappa,f_2,\beta} = rhs_{M_{spine}^{\to\circ\bullet'},f_2,\beta}$ using $\Rightarrow_{\mathcal{E}_{M_{spine},M_{pfx}}}$

As predicted by Lemma 4.39, given that $|e_{M^{\to\circ\bullet}_{spine}}|_\circ = 0$ and that $\mathcal{E}_{M_{spine},M_{pfx}}$ contains all potential $\downarrow$- and $\uparrow$-rules, the negative number of $\bullet$-symbols in the output computed in Figure 4.j using the above rules is equal to the difference of the numbers of $\circ$- and $\bullet$-symbols in the output computed in Figure 4.g for the same input. $\square$

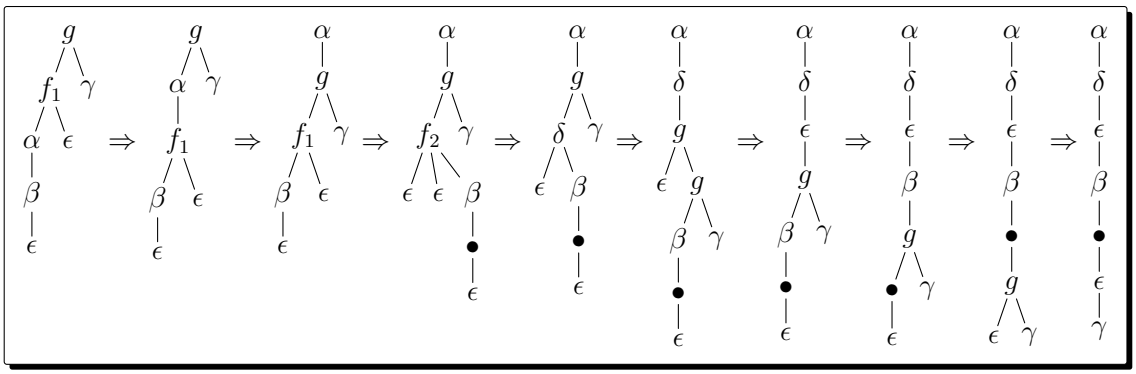

Figure 4.j: Reduction sequence using $\Rightarrow_{R^{\to\circ\bullet}_{spine} \cup R^{\circ\bullet\to\circ\bullet}_{pfx}}$

Finally, we can prove the main theorem about call-by-name efficiency for the case that neither of the two mtts to be composed is a tdtt.

Theorem 4.41 (putting together the pieces)
If M_2 is context-linear and it exists a mapping $\kappa : \{(f,k) \mid f \in F^{(r+1)}, k \in [r]\} \longrightarrow \{0,\ldots,s_{max}\}$ such that

- the right-hand sides of all rules in $R_1^{\to\circ\bullet,\kappa}$ can be rewritten with $\Rightarrow^*_{\mathcal{E}_{M_1,M_2}}$ such that all $\circ$-symbols are eliminated, and
- $e_{M_1^{\to\circ\bullet,\kappa}}$ can be rewritten with $\Rightarrow^*_{\mathcal{E}_{M_1,M_2}}$ such that all $\circ$-symbols are eliminated, or M_1 is context-linear and M_2 is atmost,

then there exists a constant $d \in \mathbb{N}$ such that $cbn_{\overline{M_1;M_2}}(t) \leq cbn_{M_1;M_2}(t) + d$ for every $t \in T_\Sigma$.

Proof
First, we recall

$$cbn_{\overline{M_1;M_2}} - cbn_{M_1;M_2} = \tau_{M_1^{\to\star\bullet}} ; \tau_{M_2^{\star\bullet\to\circ\bullet}} ; |\,\blacksquare\,|_{\circ-\bullet}$$

from the proof of Theorem 4.20. Since M_2 is context-linear, we have

$$\tau_{M_1^{\to\star\bullet}} ; \tau_{M_2^{\star\bullet\to\circ\bullet}} ; |\,\blacksquare\,|_{\circ-\bullet} \leq \tau_{M_1^{\to\circ\bullet,\kappa}} ; \tau_{M_2^{\circ\bullet\to\circ\bullet}} ; |\,\blacksquare\,|_{\circ-\bullet}$$

for every $\kappa : \{(f,k) \mid f \in F^{(r+1)}, k \in [r]\} \longrightarrow \{0,\ldots,s_{max}\}$ by Lemma 4.25, i.e.,

$$cbn_{\overline{M_1;M_2}}(t) - cbn_{M_1;M_2}(t) \leq |\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet,\kappa}}(t))|_{\circ-\bullet}$$

for every such κ and every $t \in T_\Sigma$. The statements about the particular κ whose existence is a precondition of the theorem imply that there is an mtt $M_1^{\to\circ\bullet} =$

$(F, \Sigma, \Delta \cup \{\circ, \bullet\}, e_{M_1^{\to\circ\bullet}}, R_1^{\to\circ\bullet})$ meeting the requirements of Lemma 4.39. As a consequence, we obtain

$$|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet,\kappa}}(t))|_{\circ-\bullet} \leq |e_{M_2}|_{v_1} * |e_{M_1^{\to\circ\bullet}}|_\circ - |\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet}}(t))|_\bullet$$

for every $t \in T_\Sigma$. Since $|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet}}(t))|_\bullet$ is always nonnegative, the previous two inequations imply that for every $t \in T_\Sigma$:

$$cbn_{\overline{M_1;M_2}}(t) - cbn_{M_1;M_2}(t) \leq |e_{M_2}|_{v_1} * |e_{M_1^{\to\circ\bullet}}|_\circ. \quad \square$$

Note that even if M_1 is context-linear and M_2 is atmost it makes sense to eliminate as many $\circ$-symbols as possible from $e_{M_1^{\to\circ\bullet,\kappa}}$ because that lowers the constant d up to which we can establish efficiency nondeterioration.

Example 4.42 (call-by-name analysis outcome for the running example) Given that M_{pfx} is context-linear and in the light of the rule right-hand sides and the initial expression of the mtt $M_{spine}^{\to\circ\bullet}$ from Example 4.40, the previous theorem yields that there exists a constant $d \in \mathbb{N}$ such that $cbn_{\overline{M_{spine};M_{pfx}}}(t) \leq cbn_{M_{spine};M_{pfx}}(t) + d$ for every $t \in T_{\Sigma_{mon}}$. Actually, the constant here is 0 because $|e_{M_{spine}^{\to\circ\bullet}}|_\circ = 0$. Since for the mtts M_{spine} and M_{pfx} the Lemmata 4.25 and 4.39 yield equalities instead of inequalities only (as observed in Examples 4.26 and 4.40), we could even conclude the stronger statement that for a given input t the composed program is more efficient than the original program by exactly $|\tau_{M_{pfx}^{\circ\bullet\to\circ\bullet}}(\tau_{M_{spine}^{\to\circ\bullet}}(t))|_\bullet$ steps. This is witnessed, for a concrete input, by the lengths of the call-by-name reduction sequences in Figures 3.b and 4.a on the one hand and the output computed in Figure 4.j on the other hand. By analyzing $|\tau_{M_{pfx}^{\circ\bullet\to\circ\bullet}}(\tau_{M_{spine}^{\to\circ\bullet}}(t))|_\bullet$, it is also not difficult to see that in general the amount of improvement corresponds to the number of β-blocks in the input tree (i.e. k in a decomposition as in Figure 2.a). To evaluate the benefits of our approach to efficiency analysis, the interested reader may ponder about how easy or difficult it would have been to work this out directly from the definitions of the original and the composed program as given in Examples 2.2 and 3.4, and whether such an ad-hoc reasoning could be entrusted to a compiler. $\square$

Note that in the course of our analysis up to this point we have in a sense "tunneled" all uses of context-linearity through Lemma 2.9. This means that if we were to use a syntactically more flexible restriction of mtts that still guarantees point 1 of Lemma 2.9, then Theorem 4.41 would still hold with respect to this generalized notion of "context-linear".[6] In [EM99] the desired property is called *finite copying in the parameters with parameter copying bound* 1, and following the strategy from the proof of Lemma 4.10(ii) in [EM03b] it is decidable for an mtt whether or not it has the property. While the applicability of Construction 3.1 itself would clearly not be increased through such a transition from "static" to "dynamic" context-linearity (because the use of "context-linear" in Construction 3.1 explicitly refers to

[6]The only thing that needs to be established additionally is that for such a generalized notion it would still be the case that context-linearity of M_1 implies that the inequation from point 1 of Lemma 2.9 holds also for every mtt in $\mathcal{M}_1^{\to\circ\bullet}$, and correspondingly so for M_2 and $M_2^{\circ\bullet\to\circ\bullet}$. But this is pretty obvious.

the syntactic notion from Definition 2.5), the class of programs—among those to which Construction 3.1 applies—for which our analysis guarantees efficiency non-deterioration could be enlarged by more than it might seem at first glance. This is so because also the definition of "atmost" could refer to the generalized notion of context-linearity without requiring any new proof to preserve Theorem 4.41, and so could the definition of $\mathcal{E}_{M_1,M_2}$. The other lemmata from group (iv) in Figure 4.b provide similar levers for generalizing the analysis results. In particular, alternative definitions of "atmost" and "atleast" complying to Lemmata 4.32 and 4.38 would also be acceptable, as would be a more liberal inclusion of $\uparrow$- and $\downarrow$-rules into $\mathcal{E}_{M_1,M_2}$ as long as Lemma 4.33 holds.

4.10 From call-by-name to call-by-need

Call-by-need differs from call-by-name reduction by representing terms as directed graphs rather than as trees to avoid—through sharing—repeated evaluation of the same subexpression [Wad71]. This improvement comes to bear only in situations where a call-by-name step would duplicate a subexpression containing a function call by substituting it for several occurrences of the same variable in the right-hand side of a rewrite rule. The standard way to transfer insights about call-by-name efficiency to the setting of call-by-need is thus to require all programs to be linear. Essentially, this is what [Küh99] does, and it also underlies accounts of efficiency nondeterioration for classical deforestation (cf. [San96a]).

Due to the known particular recursion scheme of mtts, we can do a bit better. First, we describe shapes of well-formed terms for the original program which ensure that in a call's context parameters no further calls occur unless context-linearity of the corresponding mtt guarantees that they are not duplicated during its reduction.

Definition 4.43 (WFT'_{M_1} and $WFT'_{M_1;M_2}$)
The set WFT'_{M_1} of *well-formed terms for M_1 without duplicable calls in context parameters* is the smallest set $WFT' \subseteq WFT_{M_1}$ satisfying the following conditions:

- for every $f \in F^{(r+1)}$, $t \in T_\Sigma$, and $\phi_1, \ldots, \phi_r \in WFT'$, if $\phi_1, \ldots, \phi_r \in T_\Delta$ or M_1 is context-linear, then $(f\ t\ \phi_1 \cdots \phi_r) \in WFT'$, and

- for every $\delta \in \Delta^{(q)}$ and $\phi_1, \ldots, \phi_q \in WFT'$: $(\delta\ \phi_1 \cdots \phi_q) \in WFT'$.

The set $WFT'_{M_1;M_2}$ of *well-formed terms for the original program without duplicable calls in context parameters* is the smallest set $WFT' \subseteq WFT_{M_1;M_2}$ satisfying the following conditions:

- for every $g \in G^{(s+1)}$, $\phi \in WFT'_{M_1}$, and $\varrho_1, \ldots, \varrho_s \in WFT'$, if $\varrho_1, \ldots, \varrho_s \in T_\Omega$ or M_2 is context-linear, then $(g\ \phi\ \varrho_1 \cdots \varrho_s) \in WFT'$, and

- for every $\omega \in \Omega^{(n)}$ and $\varrho_1, \ldots, \varrho_n \in WFT'$: $(\omega\ \varrho_1 \cdots \varrho_n) \in WFT'$. □

The following lemma establishes that only this particular kind of well-formed terms can occur during call-by-name reduction of $e_{M_2}[v_1 \leftarrow e_{M_1}[u_1 \leftarrow t]]$ for some $t \in T_\Sigma$ if each of the involved mtts is context-linear or basic.

Lemma 4.44 (auxiliary)
If M_1 is context-linear or basic and M_2 is context-linear or basic, then

1. for every $t \in T_\Sigma$ we have $e_{M_2}[v_1 \leftarrow e_{M_1}[u_1 \leftarrow t]] \in WFT'_{M_1;M_2}$, and
2. for every $\varrho \in WFT'_{M_1;M_2}$ and $\varrho' \in WFT_{M_1;M_2}$ with $\varrho \overset{cbn}{\Rightarrow}_{R_1 \cup R_2} \varrho'$ we have $\varrho' \in WFT'_{M_1;M_2}$.

Proof

1. This is immediate from the structure of the initial expressions of the involved mtts, in particular that initial expressions of basic mtts do not contain nested calls.
2. This is shown by a straightforward induction over the structure of elements of $WFT'_{M_1;M_2}$ (with nested inductions over the structure of right-hand sides of rules of M_1 and M_2), using the definition of $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ and that right-hand sides of rules of basic mtts do not contain nested calls. □

From the previous lemma and the definitions of atmostness and $\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ follows that for context-linear or basic M_1 and atmost M_2 every rewriting taking place (possibly deep inside some term) in the reduction sequence

$$e_{M_2}[v_1 \leftarrow e_{M_1}[u_1 \leftarrow t]] \left(\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}\right)^* \tau_{M_2}(\tau_{M_1}(t))$$

has one of the following two forms (with $f \in F^{(r+1)}$, $\sigma \in \Sigma^{(p)}$, $g \in G^{(s+1)}$, and $\delta \in \Delta^{(q)}$):

- $f\ (\sigma\ t_1 \cdots t_p)\ \phi_1 \cdots \phi_r \to rhs_{M_1,f,\sigma}[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \phi_1, \ldots, \phi_r]$, where
  - $t_1, \ldots, t_p \in T_\Sigma$ and
  - $\phi_1, \ldots, \phi_r \in T_\Delta$ or $|rhs_{M_1,f,\sigma}|_{y_k} \leq 1$ for every $k \in [r]$, or
- $g\ (\delta\ \phi_1 \cdots \phi_q)\ \varrho_1 \cdots \varrho_s \to rhs_{M_2,g,\delta}[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \phi_1, \ldots, \phi_q, \varrho_1, \ldots, \varrho_s]$, where
  - $|rhs_{M_2,g,\delta}|_{v_j} \leq 1$ for every $j \in [q]$ and
  - $\varrho_1, \ldots, \varrho_s \in T_\Omega$ or $|rhs_{M_2,g,\delta}|_{z_l} \leq 1$ for every $l \in [s]$.

It is thus easy to see that for such mtts no subexpression containing a call to some state is ever duplicated during the above call-by-name reduction sequence. Therefore, the number of reduction steps in this sequence, $cbn_{M_1;M_2}(t)$, does not differ from the one that would be obtained under a call-by-need strategy. Since, moreover, a call-by-need strategy always (in particular for the composed program) performs at most as many reduction steps as a call-by-name strategy, combining the above insight with Theorems 4.20 and 4.41 yields the following.

Theorem 4.45 (conditions for call-by-need efficiency nondeterioration) If

- M_1 is context-linear or basic,
- M_2 is atmost,
- M_1 is a tdtt or M_2 is context-linear,

and it exists a mapping $\kappa : \{(f,k) \mid f \in F^{(r+1)}, k \in [r]\} \longrightarrow \{0,\ldots,s_{max}\}$ such that

- the right-hand sides of all rules in $R_1^{\rightarrow\circ\bullet,\kappa}$ can be rewritten with $\Rightarrow^*_{\mathcal{E}_{M_1,M_2}}$ such that all $\circ$-symbols are eliminated, and
- $e_{M_1^{\rightarrow\circ\bullet,\kappa}}$ can be rewritten with $\Rightarrow^*_{\mathcal{E}_{M_1,M_2}}$ such that all $\circ$-symbols are eliminated, or M_1 is context-linear,

then there exists a constant $d \in \mathbb{N}$ such that for every input the composed program performs not more call-by-need steps to compute the final output than d more than the original program does. □

Example 4.46 (call-by-need analysis outcome for the running example) Given that M_{spine} is context-linear and M_{pfx} is linear and in the light of the rule right-hand sides of the mtt $M_{spine}^{\rightarrow\circ\bullet}$ from Example 4.40, the previous theorem yields that there exists a constant $d \in \mathbb{N}$ such that for every input $\overline{M_{spine};M_{pfx}}$ performs not more call-by-need steps than d more than the program consisting of expression $e_{M_{pfx}}[v_1 \leftarrow e_{M_{spine}}]$ and rules $R_{spine} \cup R_{pfx}$ does. Following the discussion from Example 4.42, the constant here is 0 and we could even conclude the stronger statement that the composed program is more efficient than the original program by at least as many steps as there are β-blocks in the input tree. Actually, the "at least" in the previous sentence can be replaced by "exactly" if we convince ourselves that for $\overline{M_{spine};M_{pfx}}$ there is no difference between the numbers of call-by-name and call-by-need steps, respectively, on every input. But this is obvious because the post-processed rules $\overline{R_{spine};R_{pfx}}$ given below Example 3.4 are all context-linear. □

Note that Theorem 4.45 in particular indicates a call-by-need efficiency nondeterioration if M_1 is a tdtt and M_2 is atmost, or if M_1 is context-linear or basic and M_2 is a linear tdtt. In both cases there is only a trivial choice for the mapping κ because its source is an empty set if M_1 is a tdtt, while its target is a singleton set if M_2 is a tdtt. It is easy to see that in each case neither $e_{M_1^{\rightarrow\circ\bullet,\kappa}}$ nor the right-hand sides of rules in $R_1^{\rightarrow\circ\bullet,\kappa}$ contain any $\circ$-symbols. It is also easy to see—from Theorem 4.20—that actually $d = 0$ in those two special cases.[7] They correspond to Theorem 1 of [Voi02b], except for the different formalization of initial expressions in that paper. The latter also contains results obtained by considering only the steps performed by the

[7] Drawing on the observation from Footnote 2 on page 45, we can even say that the amount of (call-by-need) improvement in these cases is at least the number of steps performed by states of M_2 in the original program.

second mtt in the original program, which allows weaker restrictions than context-linearity or basicness of M_1 and atmostness of M_2 to still guarantee, using a simpler annotation, that call-by-need steps are counted. The same approach could be followed if neither of the involved mtts is a tdtt. We refrain from doing so here and only provide the outcome for the special case considered in [Voi02b].

Theorem 4.47 (essentially Theorem 2 of [Voi02b])
If either of M_1 and M_2 is a tdtt, M_2 is context-linear or basic, $e_{M_1} = f\ u_1\ \theta_1 \cdots \theta_r$ and $e_{M_2} = g\ v_1\ \eta_1 \cdots \eta_s$ for some $f \in F^{(r+1)}$, $g \in G^{(s+1)}$, $\theta_1, \ldots, \theta_r \in T_\Delta$, and $\eta_1, \ldots, \eta_s \in T_\Omega$, and it exists a $\lambda \in \{0, 1\}$ such that the right-hand sides of all rules in

$R_1^{\rightarrow\circ\bullet,\lambda}$:

$$f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r \to \circ^{(1-\lambda)}\ rhs_{M_1,f,\sigma}\ [\delta\ \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]\ [f' \twoheadleftarrow \circ^\lambda \cdot f' \mid f' \in F] \qquad \forall f \in F^{(r+1)}, \sigma \in \Sigma^{(p)}$$

can be rewritten with $\Rightarrow^*_{\mathcal{E}_{M_1,M_2}}$ such that all $\circ$-symbols are eliminated, then for every input the composed program performs not more call-by-need steps to compute the final output than the original program does.[8] $\square$

Actually, in [Voi02b] no $\downarrow_{f,k}$-rules are available to rewrite right-hand sides of rules in $R_1^{\rightarrow\circ\bullet,\lambda}$. However, since we have proved Lemma 4.34 (corresponding to Lemma 7 in [Voi02b]) for the general case that $\mathcal{E}_{M_1,M_2}$ may contain $\downarrow_{f,k}$-rules, the theorem is also valid as given here. In any case, by Definition 4.28 such rules are contained in $\mathcal{E}_{M_1,M_2}$ only if M_1 is context-linear and M_2 is atmost, and under these conditions we already know—by specialization of Theorem 4.45 to the case that either of the involved mtts is a tdtt, see above—that the composed program never performs more call-by-need steps than the original program does.

Theorems 4.45 and 4.47 are not the end of the road. For example, the latter could be generalized by using a **mapping** λ from F to $\{0, 1\}$—similar to the parameterization of κ over context parameters of states of M_1 in Definition 4.21—, or one could indeed consider the approximation of counting only the steps of the second mtt in the original program, but therefor under weaker a priori restrictions, also for the general case of Construction 3.1. However, the theorems are already quite powerful in practice, both with respect to typical examples and in comparison to previous formal results regarding the effect of tree transducer composition on call-by-need efficiency (cf. Subsection 7.5.2).

[8] Actually, since the theorem is based on a count disregarding the steps of the first mtt, we can even say that the amount of improvement here is at least the number of steps performed by states of M_1 in the original program, minus λ.

Chapter 5

Efficiency of the analysis

At the heart of applying Theorems 4.41, 4.45, and 4.47 to establish call-by-name or call-by-need efficiency nondeterioration for a particular application of our composition construction is the need to check, given a rewrite system $\mathcal{E}$ containing $\bullet\,(\circ\,v) \to v$, $\circ\,(\bullet\,v) \to v$, and some of the rules from Definition 4.27[1], whether certain $\circ$- and $\bullet$-annotated versions of rule right-hand sides and the initial expression of M_1 can be rewritten with $\Rightarrow_{\mathcal{E}}^*$ such that all $\circ$-symbols are eliminated. That is, we have to be able to check the following predicate for some $\phi \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$:

$$\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).\ \phi \Rightarrow_{\mathcal{E}}^* \phi' \tag{6}$$

Since in general $\Rightarrow_{\mathcal{E}}$ is neither confluent nor terminating, we cannot simply solve this task by computing a unique normal form. However, given that none of the rewrite rules potentially in $\mathcal{E}$ introduces new symbols, we can enumerate all the (finitely many) trees to which ϕ rewrites via $\Rightarrow_{\mathcal{E}}^*$ and check whether at least one of them is free of $\circ$-symbols. This means that the conditions from all our theorems are decidable.[2] Clearly, it does not yet mean that they can be checked efficiently as is desirable for inclusion into a compiler.

There is much redundancy in considering all possible ways to rewrite a given ϕ via $\Rightarrow_{\mathcal{E}}^*$ if all we are aiming at is elimination of $\circ$-symbols. For example, it does not make sense to move a $\bullet$-symbol down into a subtree that contains no $\circ$-symbols at all, and it does not make sense to move a $\bullet$-symbol out of a subtree just to later move another $\bullet$-symbol back in. This suggests that there might be a way to check the above predicate more efficiently. In fact, in the remainder of this chapter we develop a procedure deciding (6) in only a single traversal over ϕ. A slightly

[1] We consider an arbitrary $\mathcal{E}$ of this kind rather than the fixed $\mathcal{E}_{M_1,M_2}$ from Definition 4.28 for two reasons. Firstly, we want to allow for potential generalizations as discussed at the end of Section 4.9. Secondly, as discussed later in Section 6.2, we have to change the constitution of $\mathcal{E}_{M_1,M_2}$ to analyze efficiency in the case that the final output of the mtts to be composed is consumed only partially.

[2] Note that applying Theorem 4.20 requires only a simple syntactic check, namely whether either of M_1 and M_2 is a tdtt. Similarly, applying Theorems 4.41, 4.45, and 4.47 requires syntactic checks for some of the restrictions introduced in Definition 2.5. The quantifications over κ and λ, respectively, in these theorems cause no problems because in each case there is only a finite number of possibilities to check.

less general variant of this procedure has already been implemented in [Reu03] for checking the conditions from Theorem 4.47. Here we also prove its correctness, thus justifying the earlier decision for the implementation. The enabling idea is to generalize the task to be solved. Instead of only considering whether a given ϕ can be rewritten using $\Rightarrow_{\mathcal{E}}^*$ such that all $\circ$-symbols are eliminated, we will also ask whether through this rewriting process some extra $\bullet$-symbols can be provided at the root or whether a successful elimination of $\circ$-symbols is possible if some $\bullet$-symbols are added on top of ϕ first. More precisely, we will decide the following predicate.

Definition 5.1 ($clean_{\mathcal{E}}$)
For every $\phi \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$ and $d \in \mathbb{Z}$:

$$clean_{\mathcal{E}}(\phi, d) = \exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).\ \begin{cases} \phi \Rightarrow_{\mathcal{E}}^* \bullet^d\ \phi' & \text{if } d > 0 \\ \bullet^{-d}\ \phi \Rightarrow_{\mathcal{E}}^*\ \phi' & \text{otherwise.} \end{cases} \quad \square$$

Note that (6) coincides with $clean_{\mathcal{E}}(\phi, 0)$. Our aim is to characterize $clean_{\mathcal{E}}$ inductively in terms of the structure of its first argument. First, we prove some properties of $\Rightarrow_{\mathcal{E}}^*$.

Lemma 5.2 (auxiliary)
For every $\phi, \phi' \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$:

$$\bullet\ \phi \Rightarrow_{\mathcal{E}}^* \bullet\ \phi' \ \text{ iff } \ \phi \Rightarrow_{\mathcal{E}}^*\ \phi'.$$

Proof
The "if"-direction is obvious. For the "only-if"-direction we prove by induction on $n \in \mathbb{N}$ that $\bullet\ \phi \Rightarrow_{\mathcal{E}}^n \bullet\ \phi'$ implies $\phi \Rightarrow_{\mathcal{E}}^*\ \phi'$ for every $\phi, \phi' \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$. The induction step[3] is by contradiction. Assume for given $n \in \mathbb{N}$ and $\phi, \phi' \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$ that $\bullet\ \phi \Rightarrow_{\mathcal{E}}^n \bullet\ \phi'$ but not $\phi \Rightarrow_{\mathcal{E}}^*\ \phi'$. Then, in $\bullet\ \phi \Rightarrow_{\mathcal{E}}^n \bullet\ \phi'$ the top-most $\bullet$-symbol must actively take part in some rewrite step. Consider the first such step. Then, there are $\theta, \theta' \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$ and $n_1 \in \mathbb{N}$ with $\phi \Rightarrow_{\mathcal{E}}^*\ \theta$, $\bullet\ \theta \Rightarrow_{\mathcal{E}}\ \theta'$, $\theta' \Rightarrow_{\mathcal{E}}^{n_1} \bullet\ \phi'$, and $n_1 < n$, where the rewrite step $\bullet\ \theta \Rightarrow_{\mathcal{E}}\ \theta'$ uses either the rule $\bullet\ (\circ\ v) \to v$ or one of the $\downarrow$-rules at the root. Using the following complete case distinction arising from this, we prove that $\theta \Rightarrow_{\mathcal{E}}^*\ \phi'$, which together with $\phi \Rightarrow_{\mathcal{E}}^*\ \theta$ gives a contradiction to $\phi \not\Rightarrow_{\mathcal{E}}^*\ \phi'$.

$\underline{\theta = \circ\ \theta'}$:
In this case, $\theta \Rightarrow_{\mathcal{E}}^*\ \phi'$ follows easily from $\theta' \Rightarrow_{\mathcal{E}}^{n_1} \bullet\ \phi'$, using the rule $\circ\ (\bullet\ v) \to v$.

$\underline{\theta = \delta\ \phi_1 \cdots \phi_q \text{ and } \theta' = \delta\ \phi_1 \cdots (\bullet\ \phi_j) \cdots \phi_q}$ for some $\delta \in \Delta^{(q)}$, $\phi_1, \ldots, \phi_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$, and $j \in [q]$ with $\downarrow_{\delta,j} \in \mathcal{E}$:
Somewhere in $\theta' \Rightarrow_{\mathcal{E}}^{n_1} \bullet\ \phi'$, an $\uparrow_{\delta,j'}$-rule for some $j' \in [q]$ must be applied at the root because there is no other way to get a $\bullet$-symbol into the top-most position. Consider the first such application. Then, in addition to $j' \in [q]$, there exist $\phi'_1, \ldots, \phi'_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$ and $n_2, n_3 \in \mathbb{N}$ with $\phi_1 \Rightarrow_{\mathcal{E}}^*\ \phi'_1, \ldots, \bullet\ \phi_j \Rightarrow_{\mathcal{E}}^{n_2} \phi'_j, \ldots, \phi_q \Rightarrow_{\mathcal{E}}^*\ \phi'_q$, $lab(\phi'_{j'}, \varepsilon) = \bullet$, $\uparrow_{\delta,j'} \in \mathcal{E}$, $\bullet\ (\delta\ \phi'_1 \cdots sub(\phi'_{j'}, 1) \cdots \phi'_q) \Rightarrow_{\mathcal{E}}^{n_3} \bullet\ \phi'$, and $n_2, n_3 < n_1$. By the induction hypothesis for $n_3 < n$ we get $\delta\ \phi'_1 \cdots sub(\phi'_{j'}, 1)$

[3]Note that following how induction over natural numbers is described in Section 2.3 there is no need for explicit consideration of a separate induction base case here.

$\cdots \phi'_q \Rightarrow^*_{\mathcal{E}} \phi'$, so for proving $\theta \Rightarrow^*_{\mathcal{E}} \phi'$ it suffices to prove that $\delta\ \phi'_1 \cdots \phi_j \cdots \phi'_q \Rightarrow^*_{\mathcal{E}} \delta\ \phi'_1 \cdots sub(\phi'_{j'}, 1) \cdots \phi'_q$. We do this by case distinction on j and j' being equal or not.

$\underline{j = j'}$: In this case it is enough to show $\phi_j \Rightarrow^*_{\mathcal{E}} sub(\phi'_j, 1)$, which follows by the induction hypothesis for $n_2 < n$ from $\bullet\ \phi_j \Rightarrow^{n_2}_{\mathcal{E}} \phi'_j$ and $lab(\phi'_j, \varepsilon) = \bullet$.

$\underline{j < j'}$:

$\delta\ \phi'_1 \cdots \phi_j \cdots \phi'_q$
$\Rightarrow_{\mathcal{E}}$ (by $lab(\phi'_{j'}, \varepsilon) = \bullet$ and $\uparrow_{\delta,j'} \in \mathcal{E}$)
$\bullet\ (\delta\ \phi'_1 \cdots \phi_j \cdots sub(\phi'_{j'}, 1) \cdots \phi'_q)$
$\Rightarrow_{\mathcal{E}}$ (by $\downarrow_{\delta,j} \in \mathcal{E}$)
$\delta\ \phi'_1 \cdots (\bullet\ \phi_j) \cdots sub(\phi'_{j'}, 1) \cdots \phi'_q$
$\Rightarrow^{n_2}_{\mathcal{E}}$(by $\bullet\ \phi_j \Rightarrow^{n_2}_{\mathcal{E}} \phi'_j$)
$\delta\ \phi'_1 \cdots sub(\phi'_{j'}, 1) \cdots \phi'_q$

$\underline{j > j'}$:
analogous to the previous case.

$\underline{\theta = f\ u_i\ \phi_1 \cdots \phi_r \text{ and } \theta' = f\ u_i\ \phi_1 \cdots (\bullet\ \phi_k) \cdots \phi_r}$ for some $f \in F^{(r+1)}$, $u_i \in U$, $\phi_1, \ldots, \phi_r \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$, and $k \in [r]$ with $\downarrow_{f,k} \in \mathcal{E}$:
analogous to the previous case. $\square$

Lemma 5.3 (auxiliary)
For every $\phi \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$, $\phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y)$, and $n \in \mathbb{N}$:

1. $\circ\ \phi \Rightarrow^*_{\mathcal{E}} \phi'$ iff $\phi \Rightarrow^*_{\mathcal{E}} \bullet\ \phi'$
2. $\bullet^{n+1}\ (\circ\ \phi) \Rightarrow^*_{\mathcal{E}} \phi'$ iff $\bullet^n\ \phi \Rightarrow^*_{\mathcal{E}} \phi'$.

Proof

1. The "if"-direction is obvious, using the rule $\circ\ (\bullet\ v) \to v$. For the "only-if"-direction, assume $\circ\ \phi \Rightarrow^*_{\mathcal{E}} \phi'$. Since ϕ' contains no $\circ$-symbols, somewhere in this reduction the rule $\circ\ (\bullet\ v) \to v$ must be applied at the root. Consider the first such application. Then, there exists $\theta \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$ with $\phi \Rightarrow^*_{\mathcal{E}} \bullet\ \theta$ and $\theta \Rightarrow^*_{\mathcal{E}} \phi'$. From this, $\phi \Rightarrow^*_{\mathcal{E}} \bullet\ \phi'$ follows easily.

2. The "if"-direction is obvious, using the rule $\bullet\ (\circ\ v) \to v$. For the "only-if"-direction, assume $\bullet^{n+1}\ (\circ\ \phi) \Rightarrow^*_{\mathcal{E}} \phi'$. Since ϕ' contains no $\circ$-symbols, somewhere in this reduction either of the rules $\bullet\ (\circ\ v) \to v$ and $\circ\ (\bullet\ v) \to v$ must be applied below a (potentially empty if $n = 0$) "tower" of $\bullet$-symbols at the root. Consider the first such application. Then, there exists $\theta \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$ with $\phi \Rightarrow^*_{\mathcal{E}} \theta$ and, depending on which of the two rules in question is applied, we know either $\bullet^n\ \theta \Rightarrow^*_{\mathcal{E}} \phi'$ or $lab(\theta, \varepsilon) = \bullet$ and $\bullet^{n+1}\ sub(\theta, 1) \Rightarrow^*_{\mathcal{E}} \phi'$. In the former case, the desired $\bullet^n\ \phi \Rightarrow^*_{\mathcal{E}} \phi'$ follows from $\phi \Rightarrow^*_{\mathcal{E}} \theta$ and $\bullet^n\ \theta \Rightarrow^*_{\mathcal{E}} \phi'$. The latter case reduces to the former because $lab(\theta, \varepsilon) = \bullet$ implies $\bullet^{n+1}\ sub(\theta, 1) = \bullet^n\ \theta$. $\square$

The previous two lemmata allow a straightforward proof (by essentially only expanding the definition of $clean_{\mathcal{E}}$; cf. the appendix for details) of the following one.

Lemma 5.4 (characteriz. of $clean_{\mathcal{E}}$ for first argument rooted by $\bullet$ or $\circ$)
For every $\phi \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$ and $d \in \mathbb{Z}$:

1. $clean_{\mathcal{E}}(\bullet\ \phi, d)$ iff $clean_{\mathcal{E}}(\phi, d-1)$
2. $clean_{\mathcal{E}}(\circ\ \phi, d)$ iff $clean_{\mathcal{E}}(\phi, d+1)$. $\square$

To establish similar characterizations of $clean_{\mathcal{E}}$ for the cases $\delta\ \phi_1 \cdots \phi_q$ and $f\ u_i\ \phi_1 \cdots \phi_r$, we first need another auxiliary lemma.

Lemma 5.5 (propagation of $clean_{\mathcal{E}}$ via $\Rightarrow^*_{\mathcal{E}}$)
For every $\phi, \theta \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$ and $d \in \mathbb{Z}$, if $\phi \Rightarrow^*_{\mathcal{E}} \theta$ and $clean_{\mathcal{E}}(\theta, d)$, then $clean_{\mathcal{E}}(\phi, d)$.

Proof
Assume $clean_{\mathcal{E}}(\theta, d)$, implying by Definition 5.1 the following:

$$\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).\ \begin{cases} \theta \Rightarrow^*_{\mathcal{E}} \bullet^d\ \phi' & \text{if } d > 0 \\ \bullet^{-d}\ \theta \Rightarrow^*_{\mathcal{E}} \phi' & \text{otherwise.} \end{cases}$$

If $\phi \Rightarrow^*_{\mathcal{E}} \theta$, then obviously also:

$$\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).\ \begin{cases} \phi \Rightarrow^*_{\mathcal{E}} \bullet^d\ \phi' & \text{if } d > 0 \\ \bullet^{-d}\ \phi \Rightarrow^*_{\mathcal{E}} \phi' & \text{otherwise}, \end{cases}$$

which by Definition 5.1 is equivalent to $clean_{\mathcal{E}}(\phi, d)$. $\square$

Now, we can prove the following (and likewise an analogous characterization for the case $f\ u_i\ \phi_1 \cdots \phi_r$).

Lemma 5.6 (characteriz. of $clean_{\mathcal{E}}$ for first argument rooted by $\delta \in \Delta$)
For every $\delta \in \Delta^{(q)}$, $\phi_1, \ldots, \phi_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$, and $d \in \mathbb{Z}$:

$$clean_{\mathcal{E}}(\delta\ \phi_1 \cdots \phi_q, d) \text{ iff } \exists d_1, \ldots, d_q \in \mathbb{Z}.\ d \leq \sum_{j \in [q]} d_j \wedge \forall j \in [q].\ clean_{\mathcal{E}}(\phi_j, d_j) \wedge (\downarrow_{\delta,j} \in \mathcal{E} \vee 0 \leq d_j) \wedge (\uparrow_{\delta,j} \in \mathcal{E} \vee 0 \geq d_j).$$

Proof
First, we prove for every $\delta \in \Delta^{(q)}$, $\phi_1, \ldots, \phi_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$, and $n, m \in \mathbb{N}$ the equivalence

$$\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).\ \bullet^n\ (\delta\ \phi_1 \cdots \phi_q) \Rightarrow^*_{\mathcal{E}} \bullet^m\ \phi' \tag{7}$$

iff

$$\exists d_1, \ldots, d_q \in \mathbb{Z}.\ m - n \leq \sum_{j \in [q]} d_j \wedge \forall j \in [q].\ clean_{\mathcal{E}}(\phi_j, d_j) \wedge (\downarrow_{\delta,j} \in \mathcal{E} \vee 0 \leq d_j) \wedge (\uparrow_{\delta,j} \in \mathcal{E} \vee 0 \geq d_j) \tag{8}$$

as follows.

– For the "if"-direction, assume that there exist $d_1, \ldots, d_q \in \mathbb{Z}$ with $m - n \leq \sum_{j \in [q]} d_j$ and

$$clean_{\mathcal{E}}(\phi_j, d_j) \wedge (\downarrow_{\delta,j} \in \mathcal{E} \vee 0 \leq d_j) \wedge (\uparrow_{\delta,j} \in \mathcal{E} \vee 0 \geq d_j) \tag{9}$$

for every $j \in [q]$. From the former follows

$$n + \sum_{j \in [q] \wedge d_j > 0} d_j - \sum_{j \in [q] \wedge d_j < 0} -d_j \geq m \tag{10}$$

which together with $m \geq 0$ gives

$$n + \sum_{j \in [q] \wedge d_j > 0} d_j \geq \sum_{j \in [q] \wedge d_j < 0} -d_j \tag{11}$$

From (9) we can for every $j \in [q]$ draw the following conclusions based on the value of d_j:

$\underline{d_j = 0}$: $\exists \phi'_j \in RHS(F, \Delta \cup \{\bullet\}, U, Y).\ \phi_j \Rightarrow^*_{\mathcal{E}} \phi'_j$

$\underline{d_j > 0}$: $\uparrow_{\delta,j} \in \mathcal{E}$ and $\exists \phi'_j \in RHS(F, \Delta \cup \{\bullet\}, U, Y).\ \phi_j \Rightarrow^*_{\mathcal{E}} \bullet^{d_j}\ \phi'_j$

$\underline{d_j < 0}$: $\downarrow_{\delta,j} \in \mathcal{E}$ and $\exists \phi'_j \in RHS(F, \Delta \cup \{\bullet\}, U, Y).\ \bullet^{-d_j}\ \phi_j \Rightarrow^*_{\mathcal{E}} \phi'_j$.

By (11), these imply that one can rewrite $\bullet^n\ (\delta\ \phi_1 \cdots \phi_q) \Rightarrow^*_{\mathcal{E}} \bullet^{n'}\ (\delta\ \phi'_1 \cdots \phi'_q)$, where $n' = n + \sum_{j \in [q] \wedge d_j > 0} d_j - \sum_{j \in [q] \wedge d_j < 0} -d_j$ and $\phi'_1, \ldots, \phi'_q \in RHS(F, \Delta \cup \{\bullet\}, U, Y)$. Since $n' \geq m$ by (10), this completes the proof of the "if"-direction.

– For the "only-if"-direction we prove—for fixed $\delta \in \Delta^{(q)}$, $m \in \mathbb{N}$, and $\phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y)$—by induction on $c \in \mathbb{N}$ that $\bullet^n\ (\delta\ \phi_1 \cdots \phi_q) \Rightarrow^c_{\mathcal{E}} \bullet^m\ \phi'$ implies (8) for every $n \in \mathbb{N}$ and $\phi_1, \ldots, \phi_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$. The induction step is by analyzing, for given $c, n \in \mathbb{N}$ and $\phi_1, \ldots, \phi_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$, the first (if existing) rewrite step in $\bullet^n\ (\delta\ \phi_1 \cdots \phi_q) \Rightarrow^c_{\mathcal{E}} \bullet^m\ \phi'$ that involves the top-most δ-symbol. Based on whether there is no such step or whether it uses a $\downarrow$- or $\uparrow$-rule, we perform the following complete case distinction:

$\underline{n \geq m \text{ and } \phi_1 \Rightarrow^*_{\mathcal{E}} \phi'_1, \ldots, \phi_q \Rightarrow^*_{\mathcal{E}} \phi'_q}$ for some $\phi'_1, \ldots, \phi'_q \in RHS(F, \Delta \cup \{\bullet\}, U, Y)$: In this case, (8) follows easily by choosing $d_1 = \cdots = d_q = 0$.

$\underline{n > 0,\ \phi_1 \Rightarrow^*_{\mathcal{E}} \phi'_1, \ldots, \phi_q \Rightarrow^*_{\mathcal{E}} \phi'_q,\ \downarrow_{\delta,j} \in \mathcal{E},\ \bullet^{n-1}\ (\delta\ \phi'_1 \cdots (\bullet\ \phi'_j) \cdots \phi'_q) \Rightarrow^{c'}_{\mathcal{E}} \bullet^m\ \phi'}$ for some $\phi'_1, \ldots, \phi'_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$, $j \in [q]$, and $c' \in \mathbb{N}$ with $c' < c$: By the induction hypothesis for $c' < c$ we conclude from $\bullet^{n-1}\ (\delta\ \phi'_1 \cdots (\bullet\ \phi'_j) \cdots \phi'_q) \Rightarrow^{c'}_{\mathcal{E}} \bullet^m\ \phi'$ that there exist $d'_1, \ldots, d'_q \in \mathbb{Z}$ such that $m - (n-1) \leq \sum_{j' \in [q]} d'_{j'}$, $clean_{\mathcal{E}}(\bullet\ \phi'_j, d'_j)$, $clean_{\mathcal{E}}(\phi'_{j'}, d'_{j'})$ for every $j' \in [q] \setminus \{j\}$, and $(\downarrow_{\delta,j'} \in \mathcal{E} \vee 0 \leq d'_{j'}) \wedge (\uparrow_{\delta,j'} \in \mathcal{E} \vee 0 \geq d'_{j'})$ for every $j' \in [q]$. The desired (8) follows by choosing $d_j = d'_j - 1$ and $d_{j'} = d'_{j'}$ for every $j' \in [q] \setminus \{j\}$, taking into account that $m - (n-1) \leq \sum_{j' \in [q]} d'_{j'}$ implies $m - n \leq d'_j - 1 + \sum_{j' \in [q] \setminus \{j\}} d'_{j'}$, that $clean_{\mathcal{E}}(\bullet\ \phi'_j, d'_j)$ and $\phi_j \Rightarrow^*_{\mathcal{E}} \phi'_j$ imply $clean_{\mathcal{E}}(\phi_j, d'_j - 1)$ by Lemmata 5.4(1) and 5.5, that $clean_{\mathcal{E}}(\phi'_{j'}, d'_{j'})$ and $\phi_{j'} \Rightarrow^*_{\mathcal{E}} \phi'_{j'}$ imply $clean_{\mathcal{E}}(\phi_{j'}, d'_{j'})$ for every $j' \in [q] \setminus \{j\}$ by Lemma 5.5, and that $(\uparrow_{\delta,j} \in \mathcal{E} \vee 0 \geq d'_j)$ implies $(\downarrow_{\delta,j} \in \mathcal{E} \vee 0 \leq d'_j - 1) \wedge (\uparrow_{\delta,j} \in \mathcal{E} \vee 0 \geq d'_j - 1)$ by $\downarrow_{\delta,j} \in \mathcal{E}$ and $d'_j \geq d'_j - 1$.

$\phi_1 \Rightarrow^*_{\mathcal{E}} \phi'_1, \ldots, \phi_j \Rightarrow^*_{\mathcal{E}} \bullet\ \phi'_j, \ldots, \phi_q \Rightarrow^*_{\mathcal{E}} \phi'_q, \uparrow_{\delta,j} \in \mathcal{E}, \bullet^{n+1}\ (\delta\ \phi'_1 \cdots \phi'_q) \Rightarrow^{c'}_{\mathcal{E}} \bullet^m\ \phi'$

for some $\phi'_1, \ldots, \phi'_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$, $j \in [q]$, and $c' \in \mathbb{N}$ with $c' < c$: By the induction hypothesis for $c' < c$ we conclude from $\bullet^{n+1}\ (\delta\ \phi'_1 \cdots \phi'_q) \Rightarrow^{c'}_{\mathcal{E}} \bullet^m\ \phi'$ that there exist $d'_1, \ldots, d'_q \in \mathbb{Z}$ such that $m - (n+1) \leq \sum_{j' \in [q]} d'_{j'}$ and $clean_{\mathcal{E}}(\phi'_{j'}, d'_{j'}) \wedge (\downarrow_{\delta,j'} \in \mathcal{E} \vee 0 \leq d'_{j'}) \wedge (\uparrow_{\delta,j'} \in \mathcal{E} \vee 0 \geq d'_{j'})$ for every $j' \in [q]$. The desired (8) follows by choosing $d_j = d'_j + 1$ and $d_{j'} = d'_{j'}$ for every $j' \in [q] \setminus \{j\}$, taking into account that $m - (n+1) \leq \sum_{j' \in [q]} d'_{j'}$ implies $m - n \leq d'_j + 1 + \sum_{j' \in [q] \setminus \{j\}} d'_{j'}$, that $clean_{\mathcal{E}}(\phi'_j, d'_j)$ and $\phi_j \Rightarrow^*_{\mathcal{E}} \bullet\ \phi'_j$ imply $clean_{\mathcal{E}}(\phi_j, d'_j + 1)$ by Lemmata 5.4(1) and 5.5, that $clean_{\mathcal{E}}(\phi'_{j'}, d'_{j'})$ and $\phi_{j'} \Rightarrow^*_{\mathcal{E}} \phi'_{j'}$ imply $clean_{\mathcal{E}}(\phi_{j'}, d'_{j'})$ for every $j' \in [q] \setminus \{j\}$ by Lemma 5.5, and that $(\downarrow_{\delta,j} \in \mathcal{E} \vee 0 \leq d'_j)$ implies $(\downarrow_{\delta,j} \in \mathcal{E} \vee 0 \leq d'_j + 1) \wedge (\uparrow_{\delta,j} \in \mathcal{E} \vee 0 \geq d'_j + 1)$ by $\uparrow_{\delta,j} \in \mathcal{E}$ and $d'_j \leq d'_j + 1$.

This completes the proof of the "only-if"-direction of the equivalence of (7) and (8).

Now, the lemma follows by Definition 5.1 and the equivalence of (7) and (8), choosing $n = 0$ and $m = d$ in case $d > 0$ and choosing $n = -d$ and $m = 0$ in case $d \leq 0$. □

The characterizations provided by Lemmata 5.4 and 5.6 (and the analogon of the latter for the case $f\ u_i\ \phi_1 \cdots \phi_r$) do not yet suffice as a decision procedure for $clean_{\mathcal{E}}$ because of the infinitely many choices for $d_1, \ldots, d_q \in \mathbb{Z}$ in Lemma 5.6. However, given that $clean_{\mathcal{E}}(\phi, d)$ implies $clean_{\mathcal{E}}(\phi, d')$ for every $d' < d$, it is enough to determine for given ϕ the **greatest** d such that $clean_{\mathcal{E}}(\phi, d)$. This motivates the following function definition, where the return value $-\infty$ indicates that for a given ϕ there is no such $d \in \mathbb{Z}$ at all.

Definition 5.7 ($sup_{\mathcal{E}}$)
The function $sup_{\mathcal{E}} : RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y) \longrightarrow \mathbb{Z} \cup \{-\infty\}$ is defined, with $y_k \in Y$, $\delta \in \Delta^{(q)}$, $f \in F^{(r+1)}$, $u_i \in U$, and $\phi_1, \ldots, \phi_{q/r} \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$, as follows:[4]

$$
\begin{aligned}
sup_{\mathcal{E}}(y_k) &= 0 \\
sup_{\mathcal{E}}(\bullet\ \phi_1) &= sup_{\mathcal{E}}(\phi_1) + 1 \\
sup_{\mathcal{E}}(\circ\ \phi_1) &= sup_{\mathcal{E}}(\phi_1) - 1 \\
sup_{\mathcal{E}}(\delta\ \phi_1 \cdots \phi_q) &= \sum_{j \in [q]} \begin{cases} -\infty & \text{if } \downarrow_{\delta,j} \notin \mathcal{E} \wedge 0 > sup_{\mathcal{E}}(\phi_j) \\ 0 & \text{if } \uparrow_{\delta,j} \notin \mathcal{E} \wedge 0 \leq sup_{\mathcal{E}}(\phi_j) \\ sup_{\mathcal{E}}(\phi_j) & \text{otherwise} \end{cases} \\
sup_{\mathcal{E}}(f\ u_i\ \phi_1 \cdots \phi_r) &= \sum_{k \in [r]} \begin{cases} -\infty & \text{if } \downarrow_{f,k} \notin \mathcal{E} \wedge 0 > sup_{\mathcal{E}}(\phi_k) \\ 0 & \text{if } \uparrow_{f,k} \notin \mathcal{E} \wedge 0 \leq sup_{\mathcal{E}}(\phi_k) \\ sup_{\mathcal{E}}(\phi_k) & \text{otherwise.} \end{cases}
\end{aligned}
$$

□

[4]The usual interpretation of arithmetic operations and comparison relations on $\mathbb{Z} \cup \{-\infty\}$ is used, in particular $-\infty + a = -\infty$ for every $a \in \mathbb{Z} \cup \{-\infty\}$ and $d > -\infty$ for every $d \in \mathbb{Z}$.

That $sup_{\mathcal{E}}$ indeed has the desired semantics is established in the following theorem. Since $sup_{\mathcal{E}}(\phi)$ can be computed in linear time with respect to the size of ϕ, the theorem establishes an efficient procedure for deciding (6).

Theorem 5.8 ($sup_{\mathcal{E}}$ can be used to decide $clean_{\mathcal{E}}$)
For every $\phi \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$ and $d \in \mathbb{Z}$:

$$clean_{\mathcal{E}}(\phi, d) \text{ iff } d \leq sup_{\mathcal{E}}(\phi).$$

Proof
By induction on $\phi \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$:

$\underline{\phi = y_k \in Y}$:

$clean_{\mathcal{E}}(y_k, d)$
iff (by Definition 5.1)
$\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).\ \begin{cases} y_k \Rightarrow_{\mathcal{E}}^* \bullet^d\ \phi' & \text{if } d > 0 \\ \bullet^{-d}\ y_k \Rightarrow_{\mathcal{E}}^* \phi' & \text{otherwise} \end{cases}$
iff (since no rule in $\mathcal{E}$ applies to y_k and since
$\bullet^{-d}\ y_k \in RHS(F, \Delta \cup \{\bullet\}, U, Y)$ if $d \leq 0$)
$d \leq 0$
iff (by Definition 5.7)
$d \leq sup_{\mathcal{E}}(y_k)$

$\underline{\phi = \bullet\ \phi_1}$ for some $\phi_1 \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$:

$clean_{\mathcal{E}}(\bullet\ \phi_1, d)$
iff (by Lemma 5.4)
$clean_{\mathcal{E}}(\phi_1, d-1)$
iff (by the induction hypothesis for ϕ_1)
$d - 1 \leq sup_{\mathcal{E}}(\phi_1)$
iff (by Definition 5.7 and arithmetics)
$d \leq sup_{\mathcal{E}}(\bullet\ \phi_1)$

$\underline{\phi = \circ\ \phi_1}$ for some $\phi_1 \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$:
analogous to the previous case.

$\underline{\phi = \delta\ \phi_1 \cdots \phi_q}$ for some $\delta \in \Delta^{(q)}$ and $\phi_1, \ldots, \phi_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$:

$clean_{\mathcal{E}}(\delta\ \phi_1 \cdots \phi_q, d)$
iff (by Lemma 5.6)
$\exists d_1, \ldots, d_q \in \mathbb{Z}.\ d \leq \sum_{j \in [q]} d_j \wedge \forall j \in [q].\ clean_{\mathcal{E}}(\phi_j, d_j) \wedge (\downarrow_{\delta,j} \in \mathcal{E} \vee 0 \leq d_j) \wedge (\uparrow_{\delta,j} \in \mathcal{E} \vee 0 \geq d_j)$
iff (by the induction hypotheses for the $\phi_1, \ldots, \phi_q$)

$\exists d_1, \ldots, d_q \in \mathbb{Z}.\ d \leq \sum_{j\in[q]} d_j \wedge \forall j \in [q].\ d_j \leq sup_{\mathcal{E}}(\phi_j) \wedge (\downarrow_{\delta,j} \in \mathcal{E} \vee 0 \leq d_j) \wedge (\uparrow_{\delta,j} \in \mathcal{E} \vee 0 \geq d_j)$

iff (see below)

$\exists d'_1, \ldots, d'_q \in \mathbb{Z} \cup \{-\infty\}.\ d \leq \sum_{j\in[q]} d'_j \wedge \forall j \in [q].\ d'_j = \begin{cases} -\infty & \text{if } \downarrow_{\delta,j} \notin \mathcal{E} \wedge 0 > sup_{\mathcal{E}}(\phi_j) \\ 0 & \text{if } \uparrow_{\delta,j} \notin \mathcal{E} \wedge 0 \leq sup_{\mathcal{E}}(\phi_j) \\ sup_{\mathcal{E}}(\phi_j) & \text{otherwise} \end{cases}$

iff (by applications of the one point rule of predicate logic)

$d \leq \sum_{j\in[q]} \begin{cases} -\infty & \text{if } \downarrow_{\delta,j} \notin \mathcal{E} \wedge 0 > sup_{\mathcal{E}}(\phi_j) \\ 0 & \text{if } \uparrow_{\delta,j} \notin \mathcal{E} \wedge 0 \leq sup_{\mathcal{E}}(\phi_j) \\ sup_{\mathcal{E}}(\phi_j) & \text{otherwise} \end{cases}$

iff (by Definition 5.7)

$d \leq sup_{\mathcal{E}}(\delta\ \phi_1 \cdots \phi_q)$

To prove the two implications required to close the above gap, it is enough to show two things; firstly, that for every $j \in [q]$ and $d_j \in \mathbb{Z}$,

$$d_j \leq sup_{\mathcal{E}}(\phi_j) \wedge (\downarrow_{\delta,j} \in \mathcal{E} \vee 0 \leq d_j) \wedge (\uparrow_{\delta,j} \in \mathcal{E} \vee 0 \geq d_j)$$

implies

$$d_j \leq \begin{cases} -\infty & \text{if } \downarrow_{\delta,j} \notin \mathcal{E} \wedge 0 > sup_{\mathcal{E}}(\phi_j) \\ 0 & \text{if } \uparrow_{\delta,j} \notin \mathcal{E} \wedge 0 \leq sup_{\mathcal{E}}(\phi_j) \\ sup_{\mathcal{E}}(\phi_j) & \text{otherwise;} \end{cases}$$

secondly, that for every $d \in \mathbb{Z}$ and $d'_1, \ldots, d'_q \in \mathbb{Z} \cup \{-\infty\}$,

$$d \leq \sum_{j\in[q]} d'_j \wedge \forall j \in [q].\ d'_j = \begin{cases} -\infty & \text{if } \downarrow_{\delta,j} \notin \mathcal{E} \wedge 0 > sup_{\mathcal{E}}(\phi_j) \\ 0 & \text{if } \uparrow_{\delta,j} \notin \mathcal{E} \wedge 0 \leq sup_{\mathcal{E}}(\phi_j) \\ sup_{\mathcal{E}}(\phi_j) & \text{otherwise} \end{cases}$$

implies

$$\forall j \in [q].\ d'_j \neq -\infty \wedge d'_j \leq sup_{\mathcal{E}}(\phi_j) \wedge (\downarrow_{\delta,j} \in \mathcal{E} \vee 0 \leq d'_j) \wedge (\uparrow_{\delta,j} \in \mathcal{E} \vee 0 \geq d'_j).$$

The first statement holds because $\downarrow_{\delta,j} \notin \mathcal{E} \wedge 0 > sup_{\mathcal{E}}(\phi_j)$ is in contradiction to $d_j \leq sup_{\mathcal{E}}(\phi_j) \wedge (\downarrow_{\delta,j} \in \mathcal{E} \vee 0 \leq d_j)$ and because $\uparrow_{\delta,j} \notin \mathcal{E}$ and $\uparrow_{\delta,j} \in \mathcal{E} \vee 0 \geq d_j$ imply $d_j \leq 0$. The second statement is also easily proved, using that a sum of values from $\mathbb{Z} \cup \{-\infty\}$ cannot be greater than or equal to an integer if one of its summands is $-\infty$, and that $\neg(\downarrow_{\delta,j} \notin \mathcal{E} \wedge 0 > sup_{\mathcal{E}}(\phi_j))$ and $\neg(\uparrow_{\delta,j} \notin \mathcal{E} \wedge 0 \leq sup_{\mathcal{E}}(\phi_j))$ imply $(\downarrow_{\delta,j} \in \mathcal{E} \vee 0 \leq sup_{\mathcal{E}}(\phi_j)) \wedge (\uparrow_{\delta,j} \in \mathcal{E} \vee 0 \geq sup_{\mathcal{E}}(\phi_j))$.

$\underline{\phi = f\ u_i\ \phi_1 \cdots \phi_r}$ for some $f \in F^{(r+1)}$, $u_i \in U$, and $\phi_1, \ldots, \phi_r \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$:

analogous to the previous case, using the analogon of Lemma 5.6 for $f\ u_i\ \phi_1 \cdots \phi_r$. $\square$

Chapter 6

Aspects of the transformation in practice

In this chapter we study some issues that surface when moving from the theoretical framework of tree transducers considered in Chapters 3 and 4 to a real functional language. In one case this pertains the correctness of our transformation in a richer semantic setting in which not all data structures are necessarily finite and completely defined; in two cases the concern is for efficiency aspects that have been disregarded in the analysis so far but may have an important impact in practice. In each case we first use an example to point out a problem and then go about a solution in general. The aspects of tree transducer composition to be studied here have not been treated before, so we see the developments in this chapter as vital for its success as a practical program transformation. The discussions will be mainly pragmatical, i.e., we will not exercise the same mathematical rigor by spelling out every formal detail as largely done so far. The conclusions drawn should be convincing enough, though.

As a "real functional language" we will relate to Haskell [Pey03], even though any lazy language would do. The bias towards Haskell is mainly caused by the fact that the implementation of our composition construction (and the preliminary efficiency analysis) in [Reu03] was performed by integration into a standard Haskell compiler. Note that the mentioned work already showed how to overcome the more basic obstacles when applying the theory to programs in a real language, for example, how to adjust our construction such that it also handles mtts with "mixed typing" (rather than a fixed input and a fixed output ranked alphabet to be used by all states alike). Hence, we will not consider such issues here.

6.1 Infinite and partial trees

So far, the data structures being considered as input and output of mtts have been finite trees. Since in lazy functional languages one also has the possibility to define infinite data structures, e.g. through recursive definitions such as

$$t_{inf} = \alpha\ t_{inf}\ ,$$

it is natural (and essential for practical use) to ask how tree transducer composition fares in such a setting. Theorem 3.2 is of no immediate use here because it establishes semantics preservation of our composition construction only for elements of T_Σ, i.e. for finite trees. In fact, its proof in [VK04b] uses structural induction, which is not applicable in the setting of infinite trees because there is then no induction base. The following simple example demonstrates that two mtts with the same semantics on all finite input trees need not necessarily behave identically on infinite trees.

Example 6.1 (equiv. on finite trees does not imply equiv. on infin. trees) Let $\Sigma_{nat} = \{\alpha^{(1)}, \epsilon^{(0)}\}$ and consider the two mtts $M_{rep} = (\{rep^{(1)}\}, \Sigma_{nat}, \Sigma_{nat}, rep\ u_1, R_{rep})$ and $M_{acc} = (\{acc^{(2)}\}, \Sigma_{nat}, \Sigma_{nat}, acc\ u_1\ \epsilon, R_{acc})$ with the following sets of rules:

$$\begin{aligned} R_{rep}: \quad & rep\ (\alpha\ u_1) && \to \alpha\ (rep\ u_1) \\ & rep\ \epsilon && \to \epsilon \end{aligned}$$

$$\begin{aligned} R_{acc}: \quad & acc\ (\alpha\ u_1)\ y_1 && \to acc\ u_1\ (\alpha\ y_1) \\ & acc\ \epsilon\ y_1 && \to y_1 \end{aligned}$$

It can be proved by structural induction that for every $t \in T_{\Sigma_{nat}}$: $\tau_{M_{acc}}(t) = \tau_{M_{rep}}(t)$ (see Section 7.6). Nevertheless, Figures 6.a and 6.b show that using the initial expressions and rule sets of M_{rep} and M_{acc}, respectively, on an infinite input tree leads to different outputs. While M_{rep} reproduces an infinite tree, M_{acc} never produces any accessible output. □

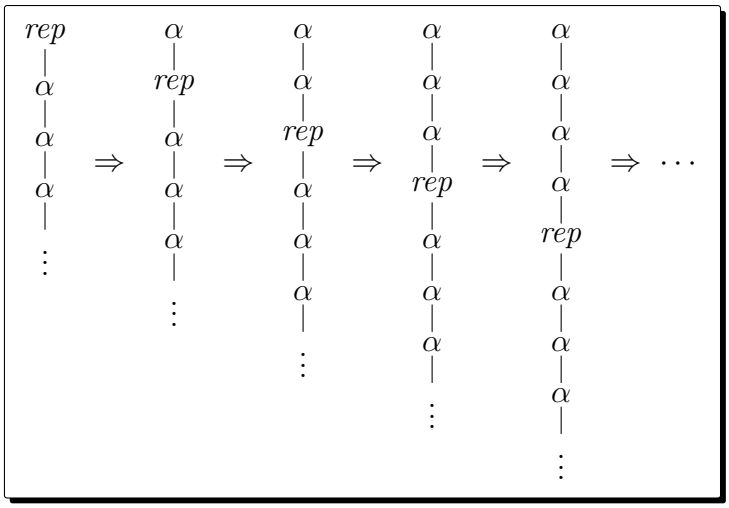

Figure 6.a: Computation of M_{rep} on t_{inf}

The example shows that it is dangerous to base assumptions about the general behavior of mtts on statements about their behavior on finite trees only. In fact, this could lead to the transformation of a terminating program into a nonterminating one, which is clearly unacceptable. In [Voi01, VK04a] a related issue was discussed for a certain post-processing step by pointing out that *elimination of copy-states* can transform a nonterminating program into a terminating program, but not vice

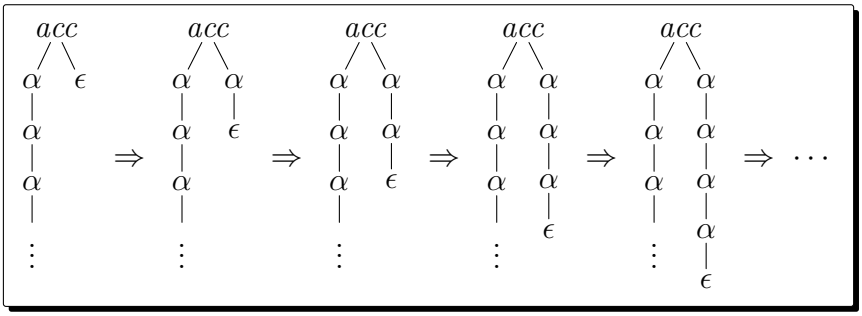

Figure 6.b: Computation of M_{acc} on t_{inf}

versa.[1] The (unspoken of) assumption underlying this discussion was that tree transducer composition itself does not change the termination behavior of programs on infinite trees. But this is far from obvious and was never justified. Similar reservations apply to other composition techniques for functions with accumulating parameters. For example, [Nis03] proposes a proof method for the alternative formulation (essentially) of our lazy composition transformation [Voi02c, Voi04b] in [Nis02], but fails when attempting to apply it to infinite inputs; the question of correctness in a setting with infinite data structures is left as an open problem. For tree transducer composition we will solve it in the remainder of this section.

The point when arguing the correctness of Construction 3.1 in the presence of infinite trees is not to somehow come up with a new model of mtts operating on infinite input (in the tradition of the transition from conventional finite automata to Büchi automata [Büc62]), but rather to examine the original vs. the composed program in the computational model of, say, Haskell. Even though there is yet no formal semantics for full Haskell, it is common practice to use a denotational style [Sch86] for reasoning about lazy functional programs. This means that one considers partial data structures potentially containing an undefined value $\bot$, along with an approximation relation $\sqsubseteq$ interpreted as "less than or equally as defined as". An infinite data structure is then the limit of an infinite approximation chain (see e.g. Figure 6.c), and by monotonicity and continuity the semantics of a function on an infinite input is considered to be the limit of the outputs of the function on the input's partial approximations. For a typical application of this approach to reasoning about Haskell programs, see [DJ04]. The following example demonstrates that—in contrast to completely defined, finite trees—partial trees can tell the difference between the

[1] The main reason for not considering elimination of copy-states in this thesis, however, is not this change in termination behavior, but rather that contrary to intuition this post-processing can actually lead to a deterioration of efficiency. This was observed in [Reu03] and attributed to the fact that elimination of copy-states may change the degree to which an input tree has to be evaluated in order to produce which part of the output. Such changes of evaluation order have to be treated with extreme care as they can have adverse effects on efficiency (see e.g. [GS01]). Hence, we do not anymore consider elimination of copy-states a post-processing that should be performed by default. For tree transducer composition itself, the results of this section guarantee that there is no dangerous change of evaluation degrees. Elimination of superfluous context parameters is also harmless in this respect.

two mtts from Example 6.1.

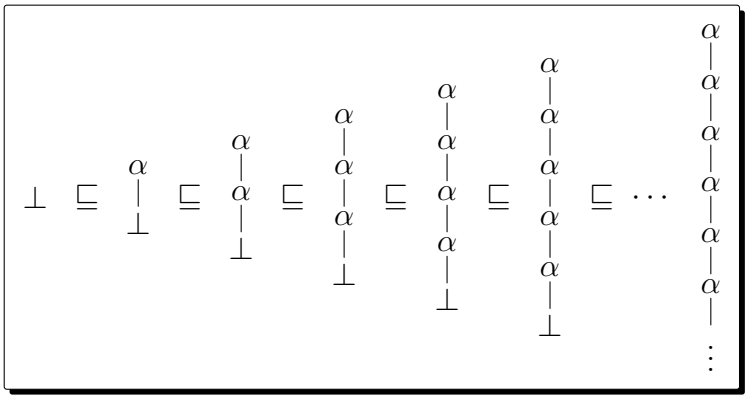

Figure 6.c: t_{inf} as the limit of a chain of partial trees

Example 6.2 (semantics on infin. trees defined by that on partial trees) Figures 6.d and 6.e show computations of M_{rep} and M_{acc}, respectively, on a particular partial tree from the approximation chain in Figure 6.c. Note that calls to *rep* or *acc* on recursion argument $\bot$ lead again to $\bot$ because pattern-matching on an undefined value fails. Generalizing from Figures 6.d and 6.e, it is easy to see that *rep* will reproduce every partial tree in the approximation chain for t_{inf}, whereas M_{acc} will output $\bot$ for every such tree. Hence, the semantics of the two mtts on the infinite tree is interpreted as $\tau_{M_{rep}}(t_{inf}) = t_{inf}$ and $\tau_{M_{acc}}(t_{inf}) = \bot$, which complies with the behavior observed in Example 6.1. □

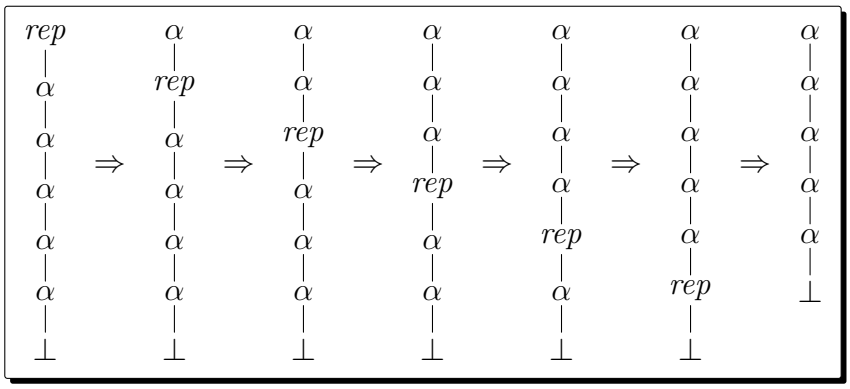

Figure 6.d: Computation of M_{rep} on a partial tree

Given the above discussion, it is clear that for arguing correctness of Construction 3.1 in the presence of infinite trees it is sufficient to show the equivalence of original and composed program on all partial trees. Actually, equivalence on partial trees is also in itself an interesting and important property (in particular in connection to Footnote 1 on page 79).

To discuss the (nonstrict) semantics of M_1, M_2, and $\overline{M_1;M_2}$ on partial trees, we explicitly introduce the undefined value as a new symbol. Let $\bot$ be a nullary symbol

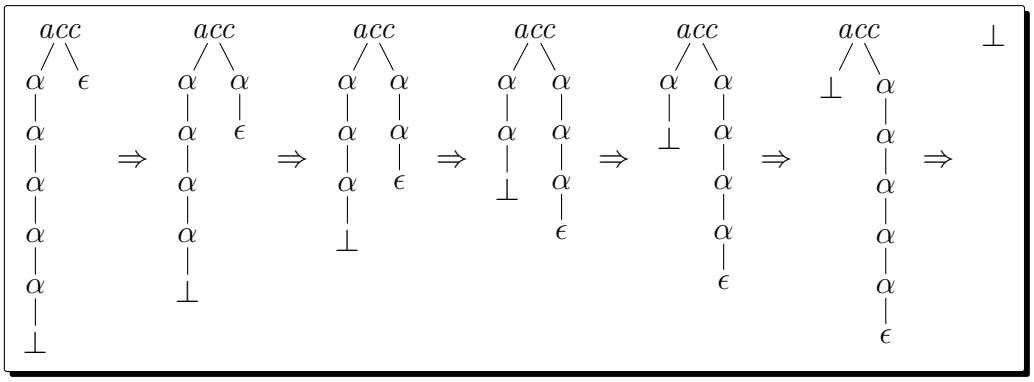

Figure 6.e: Computation of M_{acc} on a partial tree

not occurring in $\Sigma \cup \Delta \cup \Omega \cup F \cup G$. Then, a partial tree over Σ is just an element of $T_{\Sigma\cup\{\bot\}}$, and likewise for other ranked alphabets. The only way that $\bot$ can make a difference in the computation of an mtt is when it appears as recursion argument because then it leads to pattern-matching on an undefined value, whereas when it appears in a context parameter it is only copied around but not scrutinized. Hence, the following simple annotations of M_1, M_2, and $\overline{M_1;M_2}$ capture the behavior of the original and the composed program, respectively, on partial trees.

Definition 6.3 ($M_1^\bot$, $M_2^\bot$, and $\overline{M_1;M_2}^\bot$)
The mtt $M_1^\bot = (F, \Sigma \cup \{\bot\}, \Delta \cup \{\bot\}, e_{M_1}, R_1^\bot)$ has rules as follows:

$R_1^\bot$:		
$f\ (\sigma\ u_1 \cdots u_p)$	$y_1 \cdots y_r \to rhs_{M_1,f,\sigma}$	$\forall f \in F^{(r+1)}, \sigma \in \Sigma^{(p)}$
$f\ \bot$	$y_1 \cdots y_r \to \bot$	$\forall f \in F^{(r+1)}$

The mtts $M_2^\bot = (G, \Delta \cup \{\bot\}, \Omega \cup \{\bot\}, e_{M_2}, R_2^\bot)$ and $\overline{M_1;M_2}^\bot = (H, \Sigma \cup \{\bot\}, \Omega \cup \{nil, \bot\}, e_{\overline{M_1;M_2}}, \overline{R_1;R_2}^\bot)$ are defined analogously by adding $\bot$-rules whose right-hand sides are $\bot$. Note that $M_1^\bot$ is a tdtt iff M_1 is; likewise for context-linearity. Further, $M_2^\bot$ is a tdtt iff M_2 is; likewise for weakly single-useness. □

To relate the semantics of $M_1^\bot$, $M_2^\bot$, and $\overline{M_1;M_2}^\bot$, we follow the same strategy as for proving $\tau_{M_1^{\to\diamond\star}}; \tau_{M_2^{\diamond\star\to\circ}} = \tau_{\overline{M_1;M_2}^{\to\circ}}$ in Section 4.4. That is, we apply Construction 3.1 to $M_1^\bot$ and $M_2^\bot$ (which is possible due to the remarks in Definition 6.3), show that the mtt obtained is equal to $\overline{M_1;M_2}^\bot$ up to dummy rules, and appeal to semantics-preservation of the composition construction (for ordinary, finite trees!). Assuming that the same ordering of states in G is used in the two applications of the composition construction to M_1 and M_2, respectively to $M_1^\bot$ and $M_2^\bot$, we again observe that the state set of the composed mtt as well as the $nest_f$-functions and the rewrite systems *Pre* and *Pair* do not differ between the two applications. To show that except for rules whose right-hand side is *nil* the mtts $\overline{M_1^\bot;M_2^\bot} = (H, \Sigma\cup\{\bot\}, \Omega\cup\{nil, \bot\}, e_{\overline{M_1^\bot;M_2^\bot}}, \overline{R_1^\bot;R_2^\bot})$ and $\overline{M_1;M_2}^\bot$ are syntactically equivalent, we need a statement in the spirit of Lemma 4.11.

Lemma 6.4 (auxiliary)
For every $f \in F$ and $\sigma \in \Sigma$:

$$par_{M_2^\perp, rhs_{M_1,f,\sigma}} = par_{M_2, rhs_{M_1,f,\sigma}}.$$

Proof
The only difference between the definitions of $par_{M_2^\perp, rhs_{M_1,f,\sigma}}$ and $par_{M_2, rhs_{M_1,f,\sigma}}$ is that in the case $lab(rhs_{M_1,f,\sigma}, \pi) = \delta$ the former refers to the δ-rules of $M_2^\perp$, while the latter refers to the δ-rules of M_2. Since for every $\delta \in \Delta$ the δ-rules of $M_2^\perp$ are identical to those of M_2 (cf. Definition 6.3), the equality of the functions follows. (Note that the case $\delta = \perp$ is not possible because $rhs_{M_1,f,\sigma}$ contains no $\perp$-symbols.) $\square$

Taking into account that by Construction 3.1 for every $\sigma \in \Sigma$ the left-hand sides of the σ-rules in $\overline{R_1^\perp; R_2^\perp}$ coincide with the left-hand sides of the σ-rules in $\overline{R_1; R_2}$ and thus by Definition 6.3 also with those of the σ-rules in $\overline{R_1; R_2}^\perp$, and that also the left-hand sides of the $\perp$-rules in $\overline{R_1^\perp; R_2^\perp}$ coincide with those of the $\perp$-rules in $\overline{R_1; R_2}^\perp$, the desired syntactic correspondence is established as follows.

Lemma 6.5 ($\overline{M_1^\perp; M_2^\perp}$ and $\overline{M_1; M_2}^\perp$ are equal up to dummy rules)

1. $e_{\overline{M_1^\perp; M_2^\perp}} = e_{\overline{M_1; M_2}}$
2. For every $h \in H$ and $\sigma \in \Sigma \cup \{\perp\}$, if $rhs_{\overline{M_1^\perp; M_2^\perp}, h, \sigma}$ is not equal to nil, then it is equal to $rhs_{\overline{M_1; M_2}^\perp, h, \sigma}$.

Proof

1. Analogous to the proof of Lemma 4.12(1), using Definition 6.3.
2. By case analysis on $h \in H$ and $\sigma \in \Sigma \cup \{\perp\}$:

   $h = \overline{fg}$ and $\sigma \in \Sigma^{(p)}$ for some $f \in F^{(r+1)}$, $g \in G^{(s+1)}$, and $p \in \mathbb{N}$:

   $rhs_{\overline{M_1^\perp; M_2^\perp}, \overline{fg}, \sigma}$
   $=$ (by Construction 3.1 and Definition 6.3)
   $nf(\Rightarrow_{R_2^\perp \cup Pre \cup Pair}, g\ rhs_{M_1,f,\sigma}\ z_1 \cdots z_s)$
   $=$ (since $rhs_{M_1,f,\sigma} \in RHS(F, \Delta, U_p, Y_r)$ contains no $\perp$-symbols and since for every $\delta \in \Delta$ the δ-rules in $R_2^\perp$ coincide with those in R_2 by Definition 6.3)
   $nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, g\ rhs_{M_1,f,\sigma}\ z_1 \cdots z_s)$
   $=$ (by Construction 3.1 and Definition 6.3)
   $rhs_{\overline{M_1; M_2}^\perp, \overline{fg}, \sigma}$

   $h = \overline{k_f l_g}$ and $\sigma \in \Sigma^{(p)}$ for some $f \in F^{(r+1)}$, $g \in G^{(s+1)}$, $k \in [r]$, $l \in [s]$, and $p \in \mathbb{N}$:
   Note that $rhs_{M_1^\perp, f, \sigma} = rhs_{M_1, f, \sigma}$ by Definition 6.3.

– If $rhs_{M_1,f,\sigma}$ does not contain the context variable y_k, then $rhs_{\overline{M_1^\perp;M_2^\perp},\overline{k_f l_g},\sigma}$ $= nil$ by Construction 3.1.

– Otherwise, if $lab(rhs_{M_1,f,\sigma},\pi) = y_k$ for some $\pi \in paths(rhs_{M_1,f,\sigma})$, then:

$rhs_{\overline{M_1^\perp;M_2^\perp},\overline{k_f l_g},\sigma}$
$=$ (by Construction 3.1 and Lemma 6.4)
$nf(\Rightarrow_{R_2^\perp \cup Pre \cup Pair}, par_{M_2,rhs_{M_1,f,\sigma}}(\pi,g,l))$
$=$ (since $par_{M_2,rhs_{M_1,f,\sigma}}(\pi,g,l) \in T_{F\cup G\cup H\cup\Delta\cup\Omega\cup\{nil\}}(U_p \cup Y_r \cup Z_G)$ contains no $\perp$-symbols and since for every $\delta \in \Delta$ the δ-rules in $R_2^\perp$ coincide with those in R_2 by Definition 6.3)
$nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, par_{M_2,rhs_{M_1,f,\sigma}}(\pi,g,l))$
$=$ (by Construction 3.1 and Definition 6.3)
$rhs_{\overline{M_1;M_2}^\perp,\overline{k_f l_g},\sigma}$

$\underline{h = \overline{fg} \text{ and } \sigma = \perp}$ for some $f \in F$ and $g \in G$:
By Construction 3.1 and Definition 6.3 it is easy to see that $rhs_{\overline{M_1^\perp;M_2^\perp},\overline{fg},\perp} = \perp = rhs_{\overline{M_1;M_2}^\perp,\overline{fg},\perp}$.

$\underline{h = \overline{k_f l_g} \text{ and } \sigma = \perp}$ for some $f \in F^{(r+1)}$, $g \in G^{(s+1)}$, $k \in [r]$, and $l \in [s]$:
Since $rhs_{M_1^\perp,f,\perp} = \perp$ (cf. Definition 6.3) does not contain the context variable y_k, we have $rhs_{\overline{M_1^\perp;M_2^\perp},\overline{k_f l_g},\perp} = nil$ by Construction 3.1. □

In the same way in which Lemma 4.13 was proved from Lemma 4.12 and Theorem 3.2, we now obtain the following based on Lemma 6.5.

Theorem 6.6 (correctness of Constr. 3.1 in presence of partial trees) $\tau_{M_1^\perp} ; \tau_{M_2^\perp} = \tau_{\overline{M_1;M_2}^\perp}$ □

The proof is completely analogous to that of Lemma 4.13; we omit it.

Thus, another instance of externalization—this time of "undefinedness" rather than of "number of reduction steps"—combined with pushing an annotation through our composition construction has allowed us to reduce the question of semantics-preservation in the presence of partial (and thus in the presence of infinite) trees to the corresponding statement for the setting of completely defined, finite trees only. This suggests that the pattern followed here provides a general approach to analyzing Construction 3.1 (or other program transformations) that may have further applications in store as well. Indeed, it can be used to establish a correspondence between Construction 3.1 and the indirect constructions of [Küh98] and [CDPR99] with respect to their impact on call-by-name efficiency (cf. Footnote 3 on page 92). Moreover, Subsection 7.2.2 proposes how to use this pattern to approach a formal comparison of Construction 3.1 and classical deforestation with respect to the efficiency of transformed programs.

6.2 Partial demand

We start this section with a puzzle. Consider the htt $M_{top} = (\{top^{(1)}\}, \Omega_{mon}, \Omega_{mon} \setminus \{\beta\}, top\ u_1, R_{top})$ with set of rules

$$\begin{array}{rlll} R_{top}: & top\ (\alpha\ u_1) & \to \alpha\ (top\ u_1) \\ & top\ (\delta\ u_1) & \to \delta\ (top\ u_1) \\ & top\ (\epsilon\ u_1) & \to \epsilon\ (top\ u_1) \\ & top\ (\beta\ u_1) & \to \gamma \\ & top\ \gamma & \to \gamma \end{array}$$

Given an element of $T_{\Omega_{mon}}$, it outputs the top-most part until the first β is found. Now, assume that this htt is used to process the output produced by the original program (consisting of the mtts M_{spine} and M_{pfx}) from our running example. Figure 6.f shows the call-by-need reduction sequence of the complete program on a particular input, requiring ten steps altogether. According to Theorem 4.45, the composed mtt $\overline{M_{spine};M_{pfx}}$ never performs more call-by-need steps than the original program it replaces (cf. Example 4.46). Hence, we might expect the call-by-need reduction sequence of the thus transformed program on the same input in Figure 6.g (actually, we use the post-processed mtt $\overline{M_{spine};M_{pfx}}'$ given below Example 3.4, which makes no difference in terms of the number of reduction steps) to also require at most ten steps. Surprisingly, it requires one step more. Since in both cases *top* performs exactly three steps, we have to face the fact that the composition construction apparently decreased the efficiency here. What went amiss; is Theorem 4.45 wrong somehow?

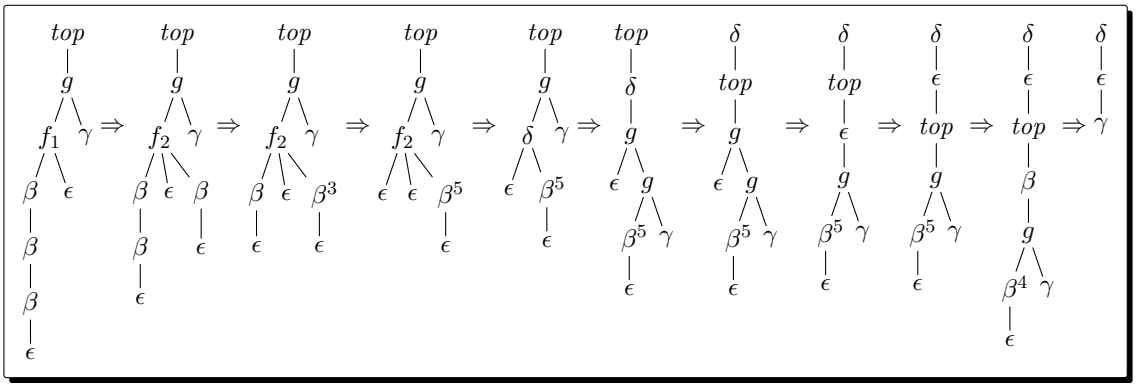

Figure 6.f: Call-by-need reduction sequence using $\Rightarrow_{R_{spine} \cup R_{pfx} \cup R_{top}}$

Theorem 4.45 is correct, of course; it is simply not applicable here. Just as the Theorems 4.20, 4.41, and 4.47, it makes a statement about the numbers of reduction steps required to compute the final output of the original and the composed program, respectively. But in Figures 6.f and 6.g this final output is never fully computed, only the part of it to be consumed by *top*. To compare the efficiency of original and composed program under such *partial demand*, we can leverage an observation made in Example 4.14. There, we noted that the output of $M_{spine}^{\to\diamond\star}$ and $M_{pfx}^{\diamond\star\to\circ}$

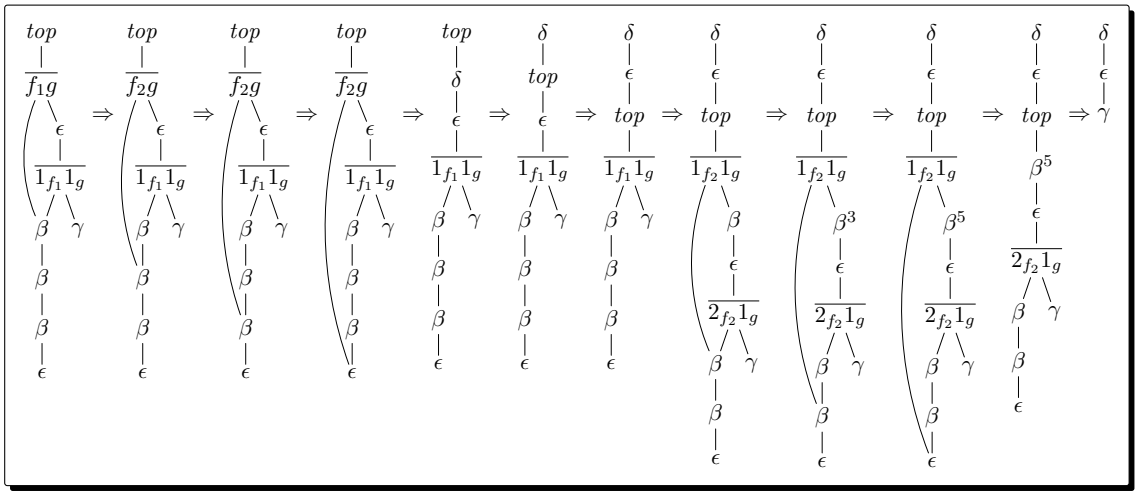

Figure 6.g: Call-by-need reduction sequence using $\Rightarrow_{\overline{R_{spine};R_{pfx}}'\cup R_{top}}$

for a particular input does not only provide insight into how many call-by-name reduction steps the transformed program requires to compute its full output, but also captures—through the distribution of $\circ$-symbols—how many steps are required to produce which part of the output. We also noted that the same is true for the distribution of $\bullet$-symbols over the output of $M_{spine}^{\rightarrow\diamond}$ and $M_{pfx}^{\diamond\rightarrow\bullet}$ with respect to call-by-name reduction steps of the original program. Figure 6.h shows these outputs for the input considered in Figures 6.f and 6.g. As we can see, there are altogether twelve $\bullet$-symbols and eleven $\circ$-symbols, meaning that on the particular input considered here the composed program is more efficient than the original program by one step if the whole output is computed. But if we only consider the initial fragments up to the first β-symbol, then there are seven $\bullet$-symbols vs. eight $\circ$-symbols, which is in line with the numbers of steps counted in Figures 6.f and 6.g, respectively, minus the three steps of *top* in both cases.[2]

Generalizing from the example, to study the efficiency of the composed vs. the original program under partial demand, we need to compare the numbers of $\circ$- and $\bullet$-symbols in (all) matching initial fragments of corresponding outputs of $M_1^{\rightarrow\diamond\star}$ and $M_2^{\diamond\star\rightarrow\circ}$ on the one hand, and of $M_1^{\rightarrow\diamond}$ and $M_2^{\diamond\rightarrow\bullet}$ on the other. The result will be exact with respect to call-by-name reduction, and a safe approximation with respect to call-by-need reduction under the restrictions from Section 4.10. Starting from these observations, we can examine the step-by-step development of our sufficient criteria to check which steps are valid also when the focus is on efficiency under partial rather than full demand on the final output. The combination and simplification of annotations in Section 4.5 is clearly harmless because the relative positions of $\circ$- and $\bullet$-symbols are not changed. In particular, Theorem 4.20 remains valid also under partial demand, as does the specialization of Theorem 4.45 to the case that one of the

[2]Note that the outputs in Figure 6.h reflect numbers of call-by-name reduction steps. But since M_{spine} is context-linear, M_{pfx} is atmost, and $\overline{M_{spine};M_{pfx}}'$ is context-linear, the numbers for call-by-need reduction are just the same.

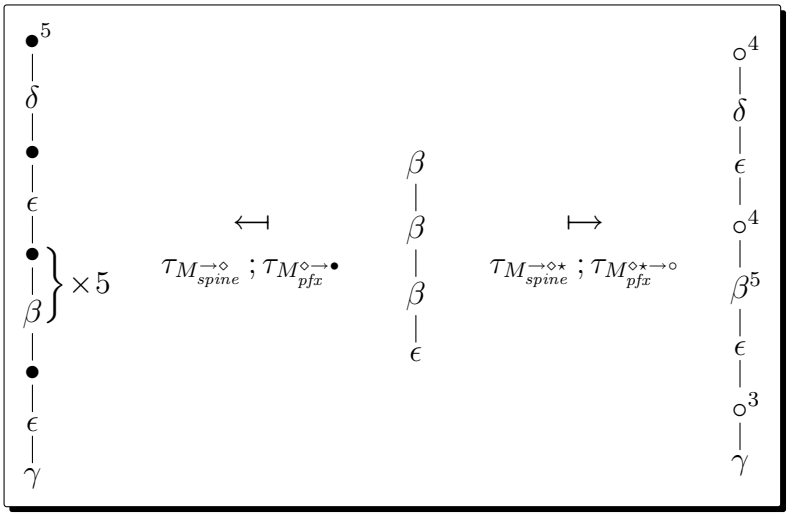

Figure 6.h: Annotated outputs for the same input

involved mtts is a tdtt. In the further manipulation of $M_1^{\rightarrow\star\bullet}$ and $M_2^{\star\bullet\rightarrow\circ\bullet}$, starting in Section 4.6, we have to check that $\circ$-symbols are never moved downwards and $\bullet$-symbols are never moved upwards. The former would mean that a step required by the composed program in the computation of a particular initial output fragment is potentially "hidden" from the comparison by moving it out of that fragment. The latter would mean that a step of the original program might be traded for a step of the composed one, even though the step of the original program is not really required in the computation of a particular initial fragment of the output that **does** require the respective step of the composed program. It is relatively easy to see that the only violations of these constraints come from the $\uparrow$-rules in $\mathcal{E}_{M_1,M_2}$. If those are outlawed, the conditions from Theorem 4.41—and hence also those from Theorem 4.45 in general—are sufficient for efficiency nondeterioration even if the possibility of partial demand is taken into account, and the same is true for Theorem 4.47.

Coming back to our running example consisting of M_{spine} and M_{pfx}, note that it is impossible to meet the conditions from either of the Theorems 4.41 and 4.45 if the $\uparrow_{f_2,2}$-rule is excluded from $\mathcal{E}_{M_{spine},M_{pfx}}$ because it is then not possible, for any κ, to rewrite the right-hand side $rhs_{M_{spine}^{\rightarrow\circ\bullet},\kappa},f_2,\beta$ with $\Rightarrow^*_{\mathcal{E}_{M_{spine},M_{pfx}}}$ such that all $\circ$-symbols are eliminated. If, on the other hand, we modify the mtt M_{spine} slightly into M'_{spine} by changing the right-hand side for f_2 at β into $rhs_{M'_{spine},f_2,\beta} = f_2\ u_1\ (\beta\ y_1)\ (\beta\ y_2)$, then for M'_{spine} and M_{pfx} the (adjusted) conditions from both Theorems 4.41 and 4.45 indicate an efficiency nondeterioration even under partial demand. And indeed, Figure 6.i shows that for the mtts $M'^{\rightarrow\star\bullet}_{spine}$ and $M^{\star\bullet\rightarrow\circ\bullet}_{pfx}$ there are never more $\circ$- than $\bullet$-symbols in any initial fragment of the final output.[3]

[3]To arrive at the general description of the mapping $\tau_{M'^{\rightarrow\star\bullet}_{spine}} ; \tau_{M^{\star\bullet\rightarrow\circ\bullet}_{pfx}}$ in Figure 6.i, one can for example compose the mtts $M'^{\rightarrow\star\bullet}_{spine}$ and $M^{\star\bullet\rightarrow\circ\bullet}_{pfx}$ using Construction 3.1 and study the resulting mtt.

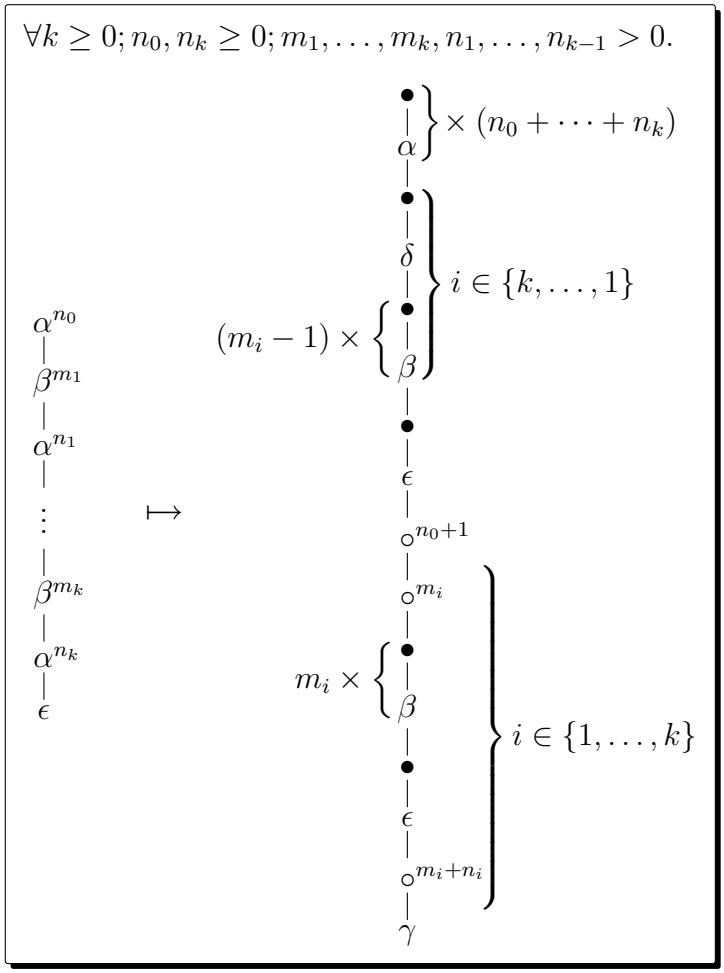

Figure 6.i: $\tau_{M'_{spine} \to \star\bullet}$; $\tau_{M^{\star\bullet \to \circ\bullet}_{pfx}}$

Given that there are now different criteria for efficiency analysis under full demand and for efficiency analysis under partial demand, where the latter are more pessimistic, which of them should be used in an actual implementation? The extension for the Glasgow Haskell Compiler (GHC; `http://www.haskell.org/ghc/`) implemented in [Reu03] by default performs an aggressive optimization by doing efficiency analysis under the assumption that the final output of a pair of mtts potentially to be composed is always completely evaluated. There is, however, a compiler switch allowing the user to fall back to the safer analysis taking the possibility of partial demand into account. A more sensitive choice would be possible by basing the decision on the context in which a potential composition pair appears in the program. That is, if a static analysis determines that in a given program context the output of the pair of mtts will be completely evaluated, then a more optimistic efficiency analysis is possible, leading to more optimization. The kind of analysis needed here is strictness detection in structured domains [Hug86, WH87]. It would be interesting to investigate whether the strictness analysis pass that already exists in GHC can be leveraged.

As a final note, we would like to point out that partial demand may not only have an adverse effect on tree transducer composition. Section 8.2 of [VK04a] contains an example where the transformed program is in general less efficient than the original one if the full output is demanded, whereas under particular partial demands an efficiency improvement is measured. Clearly, even when performing efficiency analysis with regard to partial demand, we cannot expect to profit from such situations because when considering all possible demands on the output full demand is included as a special case.

6.3 Tail calls

In [Reu03] some measurements are performed to evaluate the implementation of our composition construction based on concrete Haskell examples. One of those examples deserves particular attention, given that the measured results are counterintuitive on first sight, and also a bit disturbing.

Example 6.7 (M_{rev}, composed with itself)
Recall Σ_{mon} from Example 2.2. We define the mtt $M_{rev} = (\{rev^{(2)}\}, \Sigma_{mon}, \Sigma_{mon}, rev\ u_1\ \epsilon, R_{rev})$ with set of rules

$$\begin{aligned} R_{rev}: \quad & rev\ (\alpha\ u_1)\ y_1 \to rev\ u_1\ (\alpha\ y_1) \\ & rev\ (\beta\ u_1)\ y_1 \to rev\ u_1\ (\beta\ y_1) \\ & rev\ \epsilon \qquad y_1 \to y_1 \end{aligned}$$

Composing M_{rev} with itself according to Construction 3.1 and eliminating superfluous context parameters yields the mtt $\overline{M_{rev};M_{rev}}' = (\{\overline{rev\,rev}^{(2)}, \overline{1_{rev}1_{rev}}^{(2)}\}, \Sigma_{mon}, \Sigma_{mon} \cup \{nil\}, e_{\overline{M_{rev};M_{rev}}'}, \overline{R_{rev};R_{rev}}')$ with rules

$$
\begin{array}{lllll}
\overline{R_{rev};R_{rev}}' : & \overline{rev\,rev} & (\alpha\ u_1)\ y_{1,rev} & \to & \overline{rev\,rev}\ u_1\ y_{1,rev} \\
& \overline{rev\,rev} & (\beta\ u_1)\ y_{1,rev} & \to & \overline{rev\,rev}\ u_1\ y_{1,rev} \\
& \overline{rev\,rev} & \epsilon\ \ \ \ \ \ \ \ & \to & y_{1,rev} \\
& \overline{1_{rev}1_{rev}} & (\alpha\ u_1)\ z_{rev,1} & \to & \alpha\ (\overline{1_{rev}1_{rev}}\ u_1\ z_{rev,1}) \\
& \overline{1_{rev}1_{rev}} & (\beta\ u_1)\ z_{rev,1} & \to & \beta\ (\overline{1_{rev}1_{rev}}\ u_1\ z_{rev,1}) \\
& \overline{1_{rev}1_{rev}} & \epsilon\ \ \ \ \ \ \ \ & \to & z_{rev,1}
\end{array}
$$

and initial expression $e_{\overline{M_{rev};M_{rev}}'} = \overline{rev\,rev}\ u_1\ (\overline{1_{rev}1_{rev}}\ u_1\ \epsilon)$. $\square$

It is relatively easy to check that the conditions from Theorem 4.45 would opt for performing the transformation here (even under the adjustment for partial demand as in the previous section). And indeed, whatever the input, the original program consisting of expression $rev\ (rev\ u_1\ \epsilon)\ \epsilon$ and rules R_{rev} on the one hand and the transformed program consisting of expression $e_{\overline{M_{rev};M_{rev}}'}$ and rules $\overline{R_{rev};R_{rev}}'$ on the other require exactly the same number of call-by-need reduction steps to produce the final output. However, the measurements reveal that in practice the transformed program is considerably less efficient than the original one. In [Reu03] this is attributed to the phenomenon of *tail calls*, and we agree with that assessment.

A tail call is one that occurs at the outermost level of a function definition's right-hand side, without surrounding context. For example, the recursive calls to rev and $\overline{rev\,rev}$ in the rules given in Example 6.7 are tail calls, whereas the calls in the right-hand sides for $\overline{1_{rev}1_{rev}}$ at α and β are not. The difference this makes for efficiency stems from the fact that tail calls, if recognized and treated specially, require much less execution overhead than ordinary calls. Indeed, properly handling tail calls is one of the most important low-level optimization techniques in the implementation of compilers for functional languages (see e.g. [Bau03]). As such, it is also the most likely source for our abstraction from the different costs potentially inherent in different reduction steps to lead to imprecisions in practice. Hence, we want to briefly discuss this phenomenon in connection to our efficiency analysis, whereas an in-depth treatment is out of the scope of this thesis. In order not to depend on a precise quantitative statement by how much a tail call is more efficient than an ordinary call, we seek conditions under which in addition to the usual criterion with respect to the overall numbers of reduction steps it is also guaranteed that the transformed program never performs more non-tail calls than the original program does (up to a small additive constant depending on the program but not on the input tree). In other words, when comparing costs we may not trade a tail call of the original program for a non-tail call of the transformed program, whereas it is okay—though overly pessimistic—to trade a non-tail call of the original program for a tail call of the transformed program.

First, we consider tail calls in rules of the producer M_1. In Footnote 2 on page 45 we have observed a one-to-one correspondence between steps of $f \in F$ in the original program and steps of $\overline{fg_1}, \ldots, \overline{fg_\mu}$ in the composed one, which was used to let those steps annihilate each other. It is easy to see that whenever a step of f performs a tail call, the corresponding step of one of the $\overline{fg_1}, \ldots, \overline{fg_\mu}$ will also lead to a tail call. The reason is that if a right-hand side $rhs_{M_1,f,\sigma}$ is rooted by a call to some state f',

then also each translation of $rhs_{M_1,f,\sigma}$ with one of the states $g_1, \ldots, g_\mu$ (to obtain the right-hand sides for $\overline{fg_1}, \ldots, \overline{fg_\mu}$ at σ) is rooted by a call without surrounding context due to the *Pair*-rules. For illustration, consider the rules for f_1 and f_2 at β in Example 2.2 vs. the rules for $\overline{f_1 g}$ and $\overline{f_2 g}$ at β in Example 3.4, and the recursive rules for *rev* (in its role as producer) vs. those for $\overline{revrev}$ in Example 6.7.[4] As a consequence of the above observations we know that with respect to tail calls of the producer our analysis is already on the safe side.

With respect to tail calls in rules of the consumer M_2 we choose a quite rough approximation; they will not be included in the count at all. For Example 6.7 this means that the steps performed by *rev* in its role as consumer of α- and β-symbols will not be available to be weighed against non-tail calls of $\overline{1_{rev}1_{rev}}$ in the computation of $\overline{M_{rev};M_{rev}}'$. To discriminate tail calls of the consumer M_2 in general, we can change Definitions 4.15 and 4.21 such that $\bullet$-symbols are only inserted above δ-symbols for which there is no tail call in any associated rule of M_2. All remaining $\bullet$-symbols will for sure correspond to steps with non-tail calls only, and can thus safely be used to eliminate $\circ$-symbols. Since no elimination of $\circ$-symbols is necessary in the proof of Theorem 4.20, it stays intact even when the relative cheapness of tail calls is taken into account. The same is true for the specialization of Theorem 4.45 to the case that one of the involved mtts is a tdtt. To preserve Theorem 4.41—and thus also the general case of Theorem 4.45—in the setting where tail calls make a difference, the annotation scheme for $M_1^{\rightarrow\circ\bullet,\kappa}$ is adjusted as mentioned above. For our running example this still allows to conclude efficiency nondeterioration, given that there are no tail calls at all in the rules of M_{pfx}. For Example 6.7, however, we are only allowed to add $\bullet$-symbols above ϵ, which is not enough to eliminate all $\circ$-symbols. Composition of M_{rev} with itself will thus rightfully be rejected by the tail call-aware analysis. For Theorem 4.47 the same condition on which δ-symbols to annotate with a $\bullet$-symbol as above can be used to make sure that tail calls of M_2 are excluded from the count. There is then no advantage taken of how tail calls of M_1 induce tail calls of $\overline{M_1;M_2}$. Hence, a more sensitive analysis would be desirable, using for example special annotations for rules with tail calls. The latter is also true for Theorems 4.41 and 4.45, of course.

Finally, it is worth pointing out that the issues of tail calls and of partial demand (as discussed in the previous section) are completely orthogonal. That is, to get an analysis that is aware of both aspects, one can combine the restricted way of introducing $\bullet$-symbols with the restriction on how they may be moved.

[4]Note that the elimination of superfluous context parameters has no impact on a call being a tail call or not. Hence, it does not matter whether we consider the immediate outcome of the composition construction or the post-processed version here.

Chapter 7

Other techniques and avenues for future research

In this chapter we survey several works—some our own, some by others—that are relevant to the subject of this thesis. In particular, we consider different approaches to tree transducer composition, as well as other transformation techniques for functional programs, and how they relate to the approach featured in this thesis. We also provide pointers to the literature about formal efficiency analysis for functional programs and evaluate the strength of our analysis for tree transducer composition based on the examples considered in [KV01] and [VK04a] and by comparing our results to those of [Küh99] and [Höf99]. The chapter is more than just an overview of related work because at several places we also offer new insights, point out potential extensions on previous results, and propose directions for future research.

7.1 Tree transducer composition

Composition of tree transducers was first considered in [Rou70] for two tdtts to be composed into a single one. Using a (reversible) decomposition of mtts into tdtts and certain substitution devices [Eng80], the composition of tdtts was reused in [Eng81] to compose a tdtt and an mtt (in this order) into a single mtt. A more direct construction for this composition was later given in [Küh99]. In [EV85] an mtt and a tdtt (in this order) are composed into a single mtt. Our Construction 3.1 generalizes all these composition results. More precisely, if both M_1 and M_2 are tdtts, then it corresponds to the construction in the proof of Theorem 2 in [Rou70]; if M_1 is a tdtt, then it corresponds to Transformation 11 in [Küh99]; and if M_2 is a tdtt, then it corresponds to the construction in the proof of Theorem 4.12 in [EV85].

Composition results were also studied for other classes of tree transducers, in particular for *high-level tree transducers* [EV88], *MSO tree transducers* [BE00], and *pebble tree transducers* [EM03a]. Of those, the first ones are most relevant here because they directly correspond to functional programs and because arbitrarily many mtts can be composed into a single high-level tree transducer. However, as the name suggests, the resulting program will in general be higher-order, which

may have adverse effects on efficiency as going to be discussed in the context of shortcut deforestation in Section 7.4. Hence, we in this section restrict attention to composition techniques where mtts are composed into mtts.

7.1.1 Composition via attributed tree transducers

The potential application of tree transducer composition for optimizing functional programs was first recognized in [Küh97] and [Küh98]. These were also the first works to deal with the composition (into a single mtt) of two mtts neither of which is a tdtt. This was done using an indirection via attributed tree transducers (for short *atts*). Essentially the same approach was developed independently in [CDPR98, CDPR99], though not in the syntactical framework of tree transducers. The key idea is to transform the functions from the original program—corresponding to restricted mtts—into atts or attribute grammars (either using a further indirection via tdtts and substitution devices or using a direct construction in the spirit of [CF82]), to compose these using a construction based on ideas from [Fül81, Gan83, GG84, Gie88], and to transform the result back into an mtt using a construction from [Fra82].[1] Our direct construction is applicable to more programs because the restrictions on the original mtts are less severe. More precisely, the technique from [Küh97, Küh98] requires M_1 to be context-linear and weakly single-use, and M_2 to be weakly single-use, using a stronger notion of "weakly single-use" than we do (as indicated in Footnote 3 on page 14). In [KV01] the indirect transformation was generalized by weakening the condition on M_2 to a certain *restricted-use* restriction. This condition is incomparable with—i.e. neither strictly more nor strictly less general than—our weaker notion of "weakly single-use", but it is not too hard to see that Construction 3.1 also applies if M_1 is (only) context-linear and M_2 fulfills a straightforward generalization (by dropping the condition on the initial expression from the definition of "restricted-use") of both "restricted-use" and our notion of "weakly single-use".

Interestingly, the annotation scheme from Definition 4.9 works also for the indirect transformation via atts, i.e., an analogon of Lemma 4.12 holds, implying that any conditions for efficiency nondeterioration derived from it apply to the indirect transformation as well.[2] Put differently, we obtain that in terms of the number of call-by-name reduction steps the output programs produced by the indirect and the direct composition construction, respectively, perform identically.[3] This gives a neg-

[1] A restricted instance of attribute grammar composition is also the base for the local transformation rule proposed in [KGF02b] to eliminate intermediate results in compositions of *map*-style list transformers and for the integration of this rule with shortcut deforestation in [KGF02a].

[2] This much improves on preliminary results for the indirect transformation in [Mal01], where the main theorem about the relative efficiency of the original vs. the transformed program is still input-dependent.

[3] This is another elegant instance of the "pushing annotation through" pattern to analyze program transformations. In particular, it would be quite complicated to establish an explicit syntactic connection between Construction 3.1 and the indirect construction because in the case of the latter that would require to consider several consecutive constructions at once. Instead, one can arrive at

ative answer to the question, implicitly raised in [KV01], whether the fact that the output programs can be syntactically different provides a practical motivation for holding on to the indirect transformation via atts, even though a direct construction (with the obvious attendant benefits, e.g. for an implementor) is available. With respect to call-by-name efficiency it is now clear that there is no point in doing so. Any advantage that a program obtained by the indirect construction might, on the other hand, hypothetically enjoy under call-by-need evaluation would have to come from a somehow better use of sharing than in the program produced by the direct construction. But such an advantage is highly unlikely if the transformation result is to be an mtt in both cases. Instead, one would have to deviate from the principle that the final program should be an mtt again by using a different mapping from atts back to functional programs. The most promising candidate would be the mapping into a circular program applied in [Joh87] and [KS87]. We conjecture that this would lead to an overall transformation corresponding to our lazy composition transformation from [Voi02c, Voi04b] (cf. also the related technique in [Nis02, Nis04]), discussed in Section 7.3.

7.1.2 Composition for functions of linear size increase

In Theorem 2 of [Man03] the impressive result is proved that a composition of arbitrary many mtts can be transformed into a single one if the function computed by the composition is of linear size increase, which is decidable. The heart of the associated algorithm is to compose two mtts, the first of which is *finite copying* [EM99] and the second of which is recursion-linear. This is again done using an indirection via atts, applying essentially the composition construction from [Küh98] (cum grano salis, as expressed in [Man03], because the atts are additionally equipped with regular look-ahead). It would be interesting to investigate a direct composition construction also for this setting. Formal efficiency analysis of the final mtt vs. the original program, however, would be complicated by the fact that before we even get hands at the finite copying and the recursion-linear mtt to compose, a fair number of constructions from [Man02], [EV85], and [EM03b] have to be applied.

Note that neither is the composition technique from [Man03] more general than our Construction 3.1, nor the other way round. While in [Man03] more than two mtts can be composed and there are no a priori syntactic restrictions on their rules, our construction can also handle mtts the composition of which computes a function that is not of linear size increase. In the following we argue that there is not much room for generalizing the approach based on size increase properties rather than syntactic restrictions: already for functions of quadratic size increase it must fail. This is shown using a counterexample.

the desired correspondence by modularly threading the annotation scheme through each of these constructions individually.

Example 7.1 (M_{paths} and M_{prev})
Let $\Sigma_{tree} = \{\sigma^{(2)}, \alpha^{(0)}\}$ and $\Sigma'_{tree} = \Sigma_{tree} \cup \{1^{(1)}, 2^{(1)}\}$. We define the mtts $M_{paths} = (\{f^{(2)}\}, \Sigma_{tree}, \Sigma'_{tree}, f\ u_1\ \alpha, R_{paths})$ with set of rules

$$\begin{aligned} R_{paths}:\ & f\ (\sigma\ u_1\ u_2)\ y_1 \to \sigma\ (f\ u_1\ (1\ y_1))\ (f\ u_2\ (2\ y_1)) \\ & f\ \alpha \qquad\qquad y_1 \to y_1 \end{aligned}$$

and $M_{prev} = (\{g_1^{(1)}, g_2^{(2)}\}, \Sigma'_{tree}, \Sigma'_{tree}, g_1\ v_1, R_{prev})$ with set of rules

$$\begin{aligned} R_{prev}:\ & g_1\ (\sigma\ v_1\ v_2) \to \sigma\ (g_1\ v_1)\ (g_1\ v_2) && g_2\ (\sigma\ v_1\ v_2)\ z_1 \to \alpha \\ & g_1\ (1\ v_1) \to g_2\ v_1\ (1\ \alpha) && g_2\ (1\ v_1)\ z_1 \to g_2\ v_1\ (1\ z_1) \\ & g_1\ (2\ v_1) \to g_2\ v_1\ (2\ \alpha) && g_2\ (2\ v_1)\ z_1 \to g_2\ v_1\ (2\ z_1) \\ & g_1\ \alpha \to \alpha && g_2\ \alpha\ z_1 \to z_1 \end{aligned}$$

□

The mtt M_{paths} inserts above every leaf in a given tree the reversal of the path leading from the root to that leaf, encoded as a monadic tree over 1 and 2. The mtt M_{prev} takes such a tree and reverses the monadic parts, so that in the end the actual paths—rather than the reversed paths—are encoded. For an example tree, this is shown in Figure 7.a. It can be considered folklore knowledge that the general insertion of the actual paths above every leaf is not induced by any (single) mtt.[4] Since $size(\tau_{M_{prev}}(\tau_{M_{paths}}(t))) \leq size(t)^2$ for every $t \in T_{\Sigma_{tree}}$, this means that already for functions of quadratic size increase the mtt composition hierarchy does not collapse.

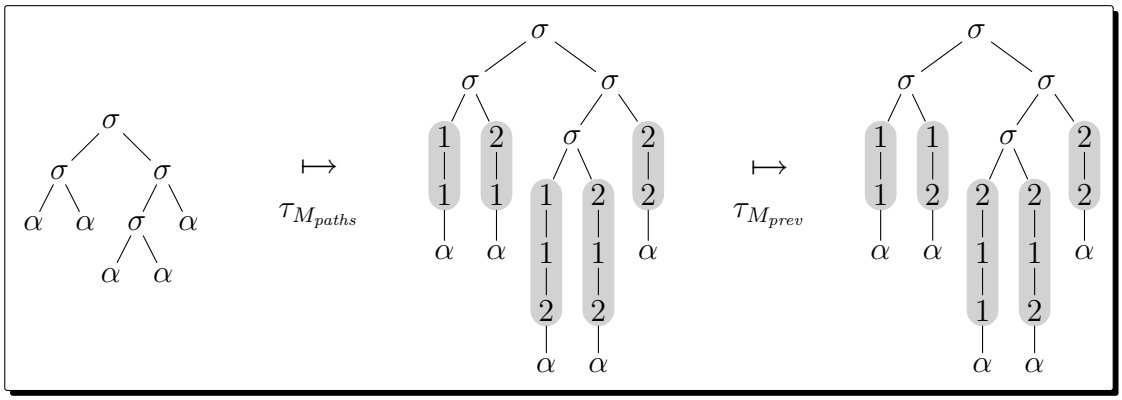

Figure 7.a: $\tau_{M_{paths}} ; \tau_{M_{prev}}$ on an example tree

Finally, we would like to point out that a variation of Example 7.1 indicates that the result from [Man03] about the collapse of the composition hierarchy for

[4]For example, the introduction of [KV94] uses exactly this operation as motivating example for generalizing mtts by inventing the more powerful concept of *macro attributed tree transducers*. However, it is interesting that the folklore statement resists formal proof attempts. In particular, the standard tools for disproving tree transducer realizability, i.e., *size*- or *height*-properties like Lemma 2.7, pumping lemmata for output languages [Küh96, Küh97], and the closure of the class of regular tree languages under inverses of macro tree transductions [EV85], are all doomed to failure here.

nondeterministic mtts under the assumptions that the relation computed by a composition is a function and is of linear size increase cannot be generalized by dropping the first assumption. In a nondeterministic mtt the rule set may contain more than one rule for a given combination of state and input symbol. The nondeterministic variant of M_{paths} with an additional unary state f' and the rule set

$$\begin{array}{rlll} R'_{paths}: & f\ (\sigma\ u_1\ u_2)\ y_1 & \to \sigma\ (f\ u_1\ (1\ y_1))\ (f'\ u_2) \\ & f\ (\sigma\ u_1\ u_2)\ y_1 & \to \sigma\ (f'\ u_1)\ (f\ u_2\ (2\ y_1)) \\ & f\ \alpha\ y_1 & \to y_1 \\ & f'\ (\sigma\ u_1\ u_2) & \to \sigma\ (f'\ u_1)\ (f'\ u_2) \\ & f'\ \alpha & \to \alpha \end{array}$$

nondeterministically inserts the appropriate reversed path above exactly one, but arbitrary, leaf rather than above every leaf of a given tree. Clearly, it computes a relation rather than a function. Composing this relation with $\tau_{M_{prev}}$ gives a (relational) transduction of linear size increase that is not induced by any nondeterministic mtt.[5]

7.1.3 Composition in a categorical framework

In [Jür04] composition of tree transducers is formulated in the language of *category theory*. Two models are proposed as generalizations of tree transducers: *algebraic transducers* and *monadic transducers*. The main result arising from the study of algebraic transducers is that for the case of tdtts tree transducer composition (using the construction from [Rou70], which is a special case of Construction 3.1) is equivalent to shortcut deforestation (see [JV04c]). To handle mtts that are no tdtts—in particular, to handle context parameters—monadic transducers are introduced, which operate over the structure of *free monads* (modeling trees along with substitution). A *fusion theorem* is proved for monadic transducers in general and can then be applied to particular tree transducers.

However, even though it is shown how arbitrary mtts can be represented as monadic transducers, there seems to be no general way to map the fusion result back into a functional program. At least, no such general mapping is provided in [Jür04], casting doubt on the claim that all constructions are "down-to-earth". Instead, the missing link is established only for subclasses of mtts that are (syntactically) characterized by the *pattern* component of monadic transducers. Such pattern characterizations are provided for htts, tdtts, *simple basic* mtts[6], basic mtts, and unrestricted mtts. Only if the monadic transducer obtained by fusion has a pattern that corresponds to one of these classes it can be mapped back into a functional

[5] As in Footnote 4 on page 94, this nonrealizability statement stems from folklore rather than from formal proof. It would be a challenging direction for future research to develop suitable tools to prove such statements.

[6] A simple basic mtt is a basic mtt such that in every rule right-hand side there are no output symbols inside context parameters of recursive calls.

program.[7] By calculating with patterns and comparing the patterns of the monadic transducers related by the fusion theorem, [Jür04] derives a number of composition results for tree transducer classes. All these results are special cases of the composition results for the case that one of two mtts is a tdtt, going back to [Eng81] and [EV85] (and being captured by our Construction 3.1), in the sense that applying the constructions from these papers to transducers as restricted in [Jür04] already yields transducers in the same classes as observed in the latter work.

In order to evaluate whether the approach via monadic transducers could also capture the cases of our Construction 3.1 where none of the involved mtts is a tdtt, it seems reasonable to ask whether context-linearity and weakly single-useness can be characterized by patterns, as that would be a necessary starting point. A statement from [Jür04] can be used to give a negative answer here. Namely, it is asserted that if a class of tree transducers can be characterized by patterns, then the associated semantic class is closed under composition (from either side) with the class of tree transductions induced by htts. By giving counterexamples, we show that neither the class of transductions induced by context-linear mtts nor that of those induced by weakly single-use mtts is closed thus. Hence, neither of the two restrictions of interest can be characterized by patterns.

Example 7.2 (M_{exp}, M_{bin}, and M_{count})
Recall Σ_{nat} and Σ_{tree} from Examples 6.1 and 7.1, respectively. We define the mtts $M_{exp} = (\{exp^{(2)}\}, \Sigma_{nat}, \Sigma_{nat}, exp\ u_1\ \epsilon, R_{exp})$, $M_{bin} = (\{bin^{(1)}\}, \Sigma_{nat}, \Sigma_{tree}, bin\ u_1, R_{bin})$, and $M_{count} = (\{count^{(2)}\}, \Sigma_{tree}, \Sigma_{nat}, count\ u_1\ \epsilon, R_{count})$ with the following sets of rules:

$$
\begin{array}{lllll}
R_{exp}: & exp & (\alpha\ u_1) & y_1 \to & exp\ u_1\ (exp\ u_1\ y_1) \\
 & exp & \epsilon & y_1 \to & \alpha\ y_1 \\
R_{bin}: & bin & (\alpha\ u_1) & \to & \sigma\ (bin\ u_1)\ (bin\ u_1) \\
 & bin & \epsilon & \to & \alpha \\
R_{count}: & count & (\sigma\ u_1\ u_2)\ y_1 & \to & \alpha\ (count\ u_1\ (count\ u_2\ y_1)) \\
 & count & \alpha & y_1 \to & \alpha\ y_1
\end{array}
$$

Clearly, M_{exp} is context-linear, M_{bin} is an htt, and M_{count} is weakly single-use. □

Theorem 7.3 (pattern noncharacterizability)

1. There is no context-linear mtt M with $\tau_M = \tau_{M_{exp}} ; \tau_{M_{bin}}$.

2. There is no weakly single-use mtt M with $\tau_M = \tau_{M_{bin}} ; \tau_{M_{count}}$.

[7] This means that in order to effectively compose two arbitrary mtts one would have to find a class of programs that corresponds to the class of patterns obtained by composing the patterns of two general mtts. In [Jür04] it is hinted that this might be the class of high-level tree transducers, but there is no real evidence. We instead conjecture that one would obtain programs like the one given in Example 7.8 as result of applying shortcut deforestation to two mtts (i.e. programs with "inner lambdas").

Proof
The proof is indirect. Assume there were a context-linear mtt M_{cl} with $\tau_{M_{cl}} = \tau_{M_{exp}} ; \tau_{M_{bin}}$ or a weakly single-use mtt M_{wsu} with $\tau_{M_{wsu}} = \tau_{M_{bin}} ; \tau_{M_{count}}$. In the former case, by Construction 3.1 and Theorem 3.2 there would be an mtt $\overline{M_{cl};M_{count}}$ with $\tau_{\overline{M_{cl};M_{count}}} = \tau_{M_{cl}} ; \tau_{M_{count}}$. In the latter case, by the same construction and theorem there would be an mtt $\overline{M_{exp};M_{wsu}}$ with $\tau_{\overline{M_{exp};M_{wsu}}} = \tau_{M_{exp}} ; \tau_{M_{wsu}}$. Hence, in both cases there would be an mtt M' with $\tau_{M'} = \tau_{M_{exp}} ; \tau_{M_{bin}} ; \tau_{M_{count}}$. By combining the last equality with the following three properties (each of which can be proved by induction):

- $height(\tau_{M_{exp}}(t)) = 1 + 2^{height(t)-1}$ for every $t \in T_{\Sigma_{nat}}$,
- $size(\tau_{M_{bin}}(t')) = 2^{height(t')} - 1$ for every $t' \in T_{\Sigma_{nat}}$, and
- $height(\tau_{M_{count}}(t'')) = size(t'') + 1$ for every $t'' \in T_{\Sigma_{tree}}$,

we could conclude that for every $t \in T_{\Sigma_{nat}}$: $height(\tau_{M'}(t)) = 2^{1+2^{height(t)-1}}$. But this double exponential height increase (for arbitrarily large input) clearly is in contradiction to Lemma 2.7. One could also use two lemmata of a similar flavor as Lemma 2.7, but for context-linear and weakly single-use mtts, respectively, to directly show that neither of M_{cl} and M_{wsu} can exist. □

Beside the above technical observations there are also more intuitive reasons for why the approach from [Jür04] is unlikely to deliver something like our Construction 3.1 or generalizations thereof. Both in the fusion of algebraic and in the fusion of monadic transducers the key ingredient is the translation of the rule right-hand sides of the producer using the rules of the consumer.[8] While this agrees well with the idea of shortcut deforestation and also with tree transducer composition as long as at least one of the involved mtts is a tdtt, the case that both the producer and the consumer have context parameters requires additional, completely different machinery. In particular, it is unclear what in the setting of [Jür04] could correspond to the "walking upwards"-scheme embodied in the definition of the *par*-functions in Construction 3.1. This also pertains the assertion that a category theory framework gives more elegant proofs by replacing explicit induction with the use of *universal properties*. While this strategy indeed works very well as long as induction means (some form of) **structural** induction, different modes of induction do not seem to be in reach so easily. For example, the correctness proof for Construction 3.1 requires induction over the prefix-order of paths in a tree and induction over the reversed subset-order on some (finite) power set, and so do other proofs about the construction, in particular those of Lemmata 4.10 and 4.11. In summary, our feeling is that the advantages of categorical semantics for studying program transformations

[8] In the fusion of algebraic transducers this manifests itself in the composition of the concrete functors corresponding to the rule sets of tdtts. In the fusion of monadic transducers it manifests itself in the composition of the producer's *rule* component and the consumer's *generalized semantics*, lifted using the producer's pattern.

unfold only if there is a close to perfect match between how the transformation proceeds and how ordinary programs compute. For the general case of tree transducer composition this is clearly not so.

7.2 Classical deforestation

Classical deforestation is based on the *define/instantiate/unfold/fold*-strategy from [BD77]. While variants of classical deforestation are applicable to arbitrary first-order programs and even higher-order variants have been devised, we restrict attention to mtts here for the sake of comparison. To this purpose, we give a description of classical deforestation tailored to (unrestricted) mtts. It uses implicit abstraction of context parameters by **let**-expressions [Küh99] to avoid termination problems of the original deforestation algorithm for definitions with terms in parameter positions that are no variables. The key idea is again a translation of the rule right-hand sides of M_1 using rules of M_2. However, the crucial step when a state of M_2 hits a state of M_1 is handled using a less intricate scheme than embodied in the *Pair*-rules of Construction 3.1. Since context parameters of M_1 are simply preserved, rather than precomputing their translations with states of M_2, there is no equivalent of the *Pre*-rules in the following construction. There is also no need for the auxiliary $\overline{k_f l_g}$-states.

Construction 7.4 (classical deforestation for mtts)
Let $M_1 = (F, \Sigma, \Delta, e_{M_1}, R_1)$ and $M_2 = (G, \Delta, \Omega, e_{M_2}, R_2)$ be mtts adhering to the naming conventions from Construction 3.1. Classical deforestation for the program $R_1 \cup R_2$ and the expression $e_{M_2}[v_1 \leftarrow e_{M_1}]$ delivers a program consisting of the set of functions $H \cup F \cup G$, the initial expression $e_{\overline{M_1;M_2}^D}$, and the rule set $\overline{R_1;R_2}^D \cup R_1 \cup R_2$, where

$$H = \{\overline{fg}^{(r+s+1)} \mid f \in F^{(r+1)}, g \in G^{(s+1)}\},$$

$$e_{\overline{M_1;M_2}^D} = nf(\Rightarrow_{R_2 \cup Fold}, e_{M_2}[v_1 \leftarrow e_{M_1}]), \text{ and}$$

$\overline{R_1;R_2}^D$ contains for every $f \in F^{(r+1)}$, $g \in G^{(s+1)}$, and $\sigma \in \Sigma^{(p)}$ the rule

$$\overline{fg}\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r\ z_1 \cdots z_s \to nf(\Rightarrow_{R_2 \cup Fold}, g\ rhs_{M_1,f,\sigma}\ z_1 \cdots z_s).$$

The right-hand side for $\overline{fg}$ at σ in $\overline{R_1;R_2}^D$ is also denoted as $rhs_{\overline{M_1;M_2}^D,\overline{fg},\sigma}$. The rewrite system *Fold* (over $F \cup G \cup H$ and $\{u\} \cup Y' \cup Z$) is defined as follows:

$$Fold : g\ (f\ u\ y'_1 \cdots y'_r)\ z_1 \cdots z_s \to \overline{fg}\ u\ y'_1 \cdots y'_r\ z_1 \cdots z_s \quad \forall g \in G^{(s+1)}, f \in F^{(r+1)}$$

The combined reduction relation $\Rightarrow_{R_2 \cup Fold}$ is confluent and terminating, hence the above use of normal forms with respect to it is justified. □

Note that classical deforestation preserves the original program because rules for the new states in H may still contain calls to functions from F and G. Clearly, the deforested program is not an mtt in general.

Example 7.5 (classical deforestation for the running example)
Applying Construction 7.4 to the mtts M_{spine} and M_{pfx} yields the new functions $H = \{\overline{f_1 g}^{(3)}, \overline{f_2 g}^{(4)}\}$ with associated rules

$$
\begin{array}{lll}
\overline{R_{spine};R_{pfx}}^{D}\colon & \overline{f_1 g}\ (\alpha\ u_1)\ y_1\ z_1 & \to \alpha\ (\overline{f_1 g}\ u_1\ y_1\ z_1) \\
& \overline{f_1 g}\ (\beta\ u_1)\ y_1\ z_1 & \to \overline{f_2 g}\ u_1\ y_1\ (\beta\ \epsilon)\ z_1 \\
& \overline{f_1 g}\ \epsilon\ y_1\ z_1 & \to g\ y_1\ z_1 \\
& \overline{f_2 g}\ (\alpha\ u_1)\ y_1\ y_2\ z_1 & \to \alpha\ (\overline{f_1 g}\ u_1\ (\delta\ y_1\ y_2)\ z_1) \\
& \overline{f_2 g}\ (\beta\ u_1)\ y_1\ y_2\ z_1 & \to \overline{f_2 g}\ u_1\ y_1\ (\beta\ (\beta\ y_2))\ z_1 \\
& \overline{f_2 g}\ \epsilon\ y_1\ y_2\ z_1 & \to \delta\ (g\ y_1\ (g\ y_2\ z_1))
\end{array}
$$

and the initial expression $e_{\overline{M_{spine};M_{pfx}}^{D}} = \overline{f_1 g}\ u_1\ \epsilon\ \gamma$. $\square$

Figure 7.b shows a call-by-need reduction sequence for the deforested program from the previous example with the same input as in Figure 4.a. Note that the deforested program still produces parts of the intermediate result (the shaded areas in Figure 7.b), but fewer than the original program does. More precisely, exactly those parts of the intermediate result were removed that occurred "outside" recursive calls in the original rules R_{spine}.

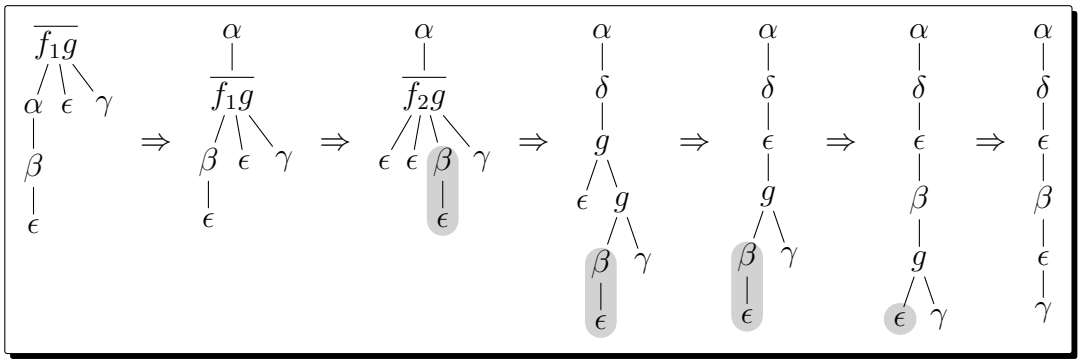

Figure 7.b: Call-by-need reduction sequence using $\Rightarrow_{\overline{R_{spine};R_{pfx}}^{D} \cup R_{spine} \cup R_{pfx}}$

The reason why classical deforestation does not reach intermediate results "inside" context parameters whereas Construction 3.1 does so, lies in the different treatment of the $y'_1, \ldots, y'_r$ by the *Fold*- and *Pair*-rules. While classical deforestation simply copies them without manipulation, our composition construction sends the states of M_2 into these context parameters of M_1. However, this does not mean that the composed program will always be more efficient than the deforested one. Since the elimination of intermediate results inside accumulating parameters comes at the price of using the auxiliary $\overline{k_f l_g}$-states, which perform additional computation, it is not too surprising that the composed program can actually be less efficient

than the deforested one, even if it is more efficient than the original program. In fact, our running example and simple variations thereof demonstrate that all conceivable relations between the composed and the deforested program with respect to call-by-need efficiency (under full demand and without special consideration of tail calls) can occur.

7.2.1 Efficiency comparison based on examples

Let us start with the running example itself. It is not all too difficult to see that compared to the original program consisting of M_{spine} and M_{pfx} the program from Example 7.5 saves exactly one reduction step for every α-symbol in the input, plus one additional step if the input ends in a β-block, i.e. if $k > 0$ and $n_k = 0$ in a decomposition as in Figure 2.a. Clearly, this number is always at least as big as the number of β-blocks k, which according to Example 4.46 is the improvement achieved by the composed program. Hence, for our running example the deforested program is on every input at least as efficient as the composed program. Moreover, a comparison of Figures 3.b and 7.b yields that the deforested program is actually more efficient on some inputs.[9]

Now, consider a variant M^{α}_{spine} of M_{spine} by changing the right-hand side for f_1 at α into $rhs_{M^{\alpha}_{spine},f_1,\alpha} = \alpha\ (f_1\ u_1\ (\alpha\ y_1))$. For the composition construction (with M_{pfx} as consumer) the additional α makes no real difference. The transformed program will simply have an additional α on top of $y_{1,g}$ in the right-hand side for $\overline{f_1 g}$ at α, without any impact on the number of reduction steps performed for given input. For classical deforestation, however, the additional α means an additional fragment of the intermediate result that is not eliminated because it is produced inside an accumulating parameter. This leads to as many additional steps of the deforested program as there are α-symbols in the input, minus the number of those α-symbols that are immediate successors of a β (because there is no change to the rule for f_2 at α). The result is exactly the number of steps by which the deforested program was better than the composed program for M_{spine} and M_{pfx} above. Hence, for M^{α}_{spine} and M_{pfx} the deforested and the composed program are equally efficient on every input with respect to the number of call-by-need reduction steps.

Further varying M^{α}_{spine} to $M^{\alpha\beta}_{spine}$ by changing the right-hand side for f_2 at β into $rhs_{M^{\alpha\beta}_{spine},f_2,\beta} = f_2\ u_1\ y_1\ (\beta\ (\beta\ (\beta\ y_2)))$ again has no relevant impact on the composed program (apart from replacing occurrences of $\beta\ (\beta\ y_{2,g})$ by $\beta\ (\beta\ (\beta\ y_{2,g}))$), but leads to as many additional steps of the deforested program as there are β-symbols in the input, minus k (because there is no change to the rule for f_1 at β). Hence, for $M^{\alpha\beta}_{spine}$ and M_{pfx} the composed program is on every input at least as efficient as the deforested program, and there are inputs for which it is actually more efficient.

[9] This comparison of course ignores that the deforested program requires additional work for allocation and eventual deallocation of memory cells for those parts of the intermediate result that were not eliminated. If those costs were also to be taken into account, the scale would tip more in favor of the composition construction.

Finally, a variant M^{β}_{spine} of M_{spine} in which the above change to the rule for f_2 at β is done, but not that to the rule for f_1 at α, leads to an example where the relative efficiency of the composed vs. the deforested program depends on the particular input provided. More precisely, for M^{β}_{spine} and M_{pfx} we obtain—along the same lines of reasoning as above—that the composed program performs fewer call-by-need reduction steps than the deforested program if and only if the number of β-symbols in the input is greater than the number of α-symbols therein, plus 1 if the input ends in a β-block; the composed and the deforested program are equally efficient if and only if the two numbers in question are equal, and the composed program is less efficient than the deforested one if and only if the former number is smaller than the latter.

7.2.2 An annotation approach to formal comparison

Even though formal efficiency analysis for classical deforestation is not an objective of this thesis, we would like to offer some thoughts in that direction. Along the same lines as in Section 4.3 the call-by-name steps of the deforested program can be reflected in the output as $\square$-symbols (where $\square$ is a special unary symbol, assumed not to occur in $\Sigma \cup \Delta \cup \Omega \cup F \cup G$) using the following annotated rules:

$$
\begin{array}{llll}
\overline{fg}\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r\ z_1 \cdots z_s & \to \square\ rhs_{\overline{M_1;M_2}^{D},\overline{fg},\sigma} & \forall f \in F^{(r+1)}, g \in G^{(s+1)}, \sigma \in \Sigma^{(p)} \\
f\ \ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r & \to \diamond\ rhs_{M_1,f,\sigma} & \forall f \in F^{(r+1)}, \sigma \in \Sigma^{(p)} \\
g\ \ (\delta\ v_1 \cdots v_q)\ \ z_1 \cdots z_s & \to \square\ rhs_{M_2,g,\delta} & \forall g \in G^{(s+1)}, \delta \in \Delta^{(q)} \\
g\ \ (\diamond\ v_1)\ \ \ z_1 \cdots z_s & \to \square\ (g\ v_1\ z_1 \cdots z_s) & \forall g \in G^{(s+1)}
\end{array}
$$

Following the development in Section 4.4 for the composition construction, we seek an annotation of the original program such that classical deforestation applied to a thus annotated version yields an equivalent of the above. To distinguish symbols of the intermediate result that will be eliminated from those that will not, we introduce a copy of Δ, namely $\underline{\Delta} = \{\underline{\delta}^{(q)} \mid \forall \delta \in \Delta^{(q)}\}$. Likewise, a copy $\underline{F}$ of F is defined to distinguish calls of the first mtt's states that are reachable by classical deforestation from those that are not. We arrive at the following enrichments of the original mtts' rules:

$$
\begin{array}{lll}
\underline{f}\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r \to \diamond\ mark(rhs_{M_1,f,\sigma}) & \forall f \in F^{(r+1)}, \sigma \in \Sigma^{(p)} \\
f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r \to \diamond\ rhs_{M_1,f,\sigma} & \forall f \in F^{(r+1)}, \sigma \in \Sigma^{(p)} \\
g\ (\underline{\delta}\ v_1 \cdots v_q)\ \ z_1 \cdots z_s \to rhs_{M_2,g,\delta} & \forall g \in G^{(s+1)}, \delta \in \Delta^{(q)} \\
g\ (\delta\ v_1 \cdots v_q)\ \ z_1 \cdots z_s \to \square\ rhs_{M_2,g,\delta} & \forall g \in G^{(s+1)}, \delta \in \Delta^{(q)} \\
g\ (\diamond\ v_1)\ \ \ z_1 \cdots z_s \to \square\ (g\ v_1\ z_1 \cdots z_s) & \forall g \in G^{(s+1)}
\end{array}
$$

where $mark : RHS(F, \Delta, U, Y) \longrightarrow RHS(F \cup \underline{F}, \Delta \cup \underline{\Delta}, U, Y)$ is defined, with $y_k \in Y$, $\delta \in \Delta^{(q)}$, $f \in F^{(r+1)}$, $u_i \in U$, and $\phi_1, \ldots, \phi_{q/r} \in RHS(F, \Delta, U, Y)$, as follows:

$$
\begin{array}{ll}
mark(y_k) & = y_k \\
mark(\delta\ \phi_1 \cdots \phi_q) & = \underline{\delta}\ mark(\phi_1)\ \cdots\ mark(\phi_q) \\
mark(f\ u_i\ \phi_1 \cdots \phi_r) & = \underline{f}\ u_i\ \phi_1\ \cdots\ \phi_r.
\end{array}
$$

Without formal proof, we assert that applying Construction 7.4 to $e_{M_2}[v_1 \leftarrow \mathit{mark}(e_{M_1})]$ and the above rules yields a program that is equivalent, up to function names and superfluous rules, to $e_{\overline{M_1;M_2}^D}$ with the annotated version of the deforested program further above. This provides the starting point for a study of call-by-name efficiency of the deforested vs. the composed program by combining the above with the annotation scheme from Definition 4.9 and then proceeding similarly to the development in Sections 4.5–4.9. To transfer the results to the call-by-need setting, we would have to use conditions under which call-by-name and call-by-need are equivalent for the deforested program or for the composed program. It is easy to see that for linear mtts M_1 and M_2 also the deforested program produced by Construction 7.4 is linear. Hence, linearity of the input program allows to conclude from call-by-name improvement of the composed over the deforested program also a corresponding call-by-need improvement. However, we conjecture that the weaker restrictions considered in Section 4.10 are also sufficient here.

By comparing the above enriched program to that from Definition 4.4, it is easy to see that there are always at most as many $\boxdot$- as $\bullet$-symbols produced. This agrees with the fact that classical deforestation is in general known to be efficiency nondeteriorating with respect to call-by-name reduction steps [San96a]. Using the facts established in Section 4.10 (which are independent of whether Construction 3.1 is applicable to M_1 and M_2 or not), we can conclude that also with respect to call-by-need the program produced by Construction 7.4 is at least as efficient as the original program if M_1 is context-linear or basic and M_2 is atmost. This generalizes a statement in Section 4.6 of [Voi02b], where a sufficient condition for efficiency nondeterioration by tree transducer composition was combined with the fact that classical deforestation and tree transducer composition are equivalent if the producer is a tdtt (see [Küh99] or compare Constructions 3.1 and 7.4 for this case) to conclude that classical deforestation is call-by-need efficiency nondeteriorating for a tdtt and an atmost mtt. Along the same lines, it was shown there (based on Theorem 4.47) that classical deforestation is call-by-need efficiency nondeteriorating for a producing tdtt and a context-linear or basic mtt, which was also not previously known.

Finally, some remarks regarding partial demand and tail calls in connection to classical deforestation. The proofs in [San96a] establish improvement in any program context, hence the concerns from Section 6.2 are not an issue when comparing the original and the deforested program. When comparing the deforested and the composed program, however, partial demand can make a difference. For example, in the case of the mtt M'_{spine} from Section 6.2 as producer and the mtt M_{pfx} as consumer it strongly depends on the degree to which the final output is evaluated whether the composed or the deforested program is preferable. In a formal efficiency analysis this could again be tackled by restricting the direction in which symbols may be moved during the manipulation of annotations. We are not aware of any efficiency analysis for classical deforestation that takes tail calls into account. However, similar arguments about tail calls in rules of M_1 being reproduced as tail calls in rules for $\overline{fg}$-functions as in Section 6.3, plus the observation that any step of M_2 is either eliminated by classical deforestation or does not change its nature of leading

to a tail call or not, are enough to see that at least for the special case of classical deforestation handled in Construction 7.4 tail calls do not change the picture of whether the original or the deforested program is more efficient. For the comparison of the deforested and the composed program, the situation is probably again more complicated.

7.3 Lazy composition

We have seen that the elimination of intermediate results by tree transducer composition comes at the price of using the auxiliary $\overline{k_f l_g}$-states as introduced in the *Pair*-rules of our construction. Consider, e.g., one such rule from the application of Construction 3.1 to our running example consisting of the mtts M_{spine} and M_{pfx}, as derived in Section 3.3:

$$\begin{aligned} g\ (f_2\ u\ y_1'\ y_2')\ z_1 \to \overline{f_2 g}\ u\ (g\ y_1'\ (\overline{1_{f_2} 1_g}\ u\ nil\ (g\ y_2'\ (\overline{2_{f_2} 1_g}\ u\ nil\ nil\ z_1))\ z_1)) \\ (g\ y_2'\ (\overline{2_{f_2} 1_g}\ u\ (g\ y_1'\ (\overline{1_{f_2} 1_g}\ u\ nil\ nil\ z_1))\ nil\ z_1))\ z_1 \end{aligned} \tag{12}$$

On the right-hand side there are five occurrences of u. In the construction of a concrete rule right-hand side for $\overline{M_{spine};M_{pfx}}$ not all of these occurrences are necessarily going to be preserved because some of them might be deleted during translation of the replacements for y_1' and y_2' with g. However, the initial expression and the rules of $\overline{M_{spine};M_{pfx}}$ in Example 3.4 show that at least some of the additional calls introduced in this way make it into the composed program, where they lead to multiple traversals of the input data structure, even though the original program performed only one such traversal (plus one traversal of the intermediate result that is avoided in the composed program). A standard technique for avoiding multiple traversals is *tupling* of several functions into a single one [Pet77, Bir84, CGK99]. However, simply applying standard tupling to the program resulting from our composition construction does not eliminate all superfluous traversals in general, due to the intricate scheme of nesting translations in *Pair*-rules like the one above (and also in the definition of $par_{M_2,\phi}$-functions). For example, the right-hand side of (12) contains two calls to $\overline{1_{f_2} 1_g}$ on u. Since they differ in their context parameters, there seems to be no apparent way to avoid the multiple traversals here. But actually we know that both these calls compute the same value, namely the context parameter of g on reaching the first context parameter of f_2's output on the input substituted for u. The only reason for their context parameters to be different is to keep the nesting of translations of finite depth as discussed in [VK04a]. Indeed, the two calls could be shared, but it is unlikely that an off-the-shelf tupling transformation that is ignorant of our composition construction would be able to figure this out on its own. The solution is of course to integrate the tupling transformation with the composition construction.

Assume that we enriched our model of mtts by allowing sharing through the use of directed acyclic graphs, instead of trees, as right-hand sides of rules. Making sharing

explicit through **let**-expressions, we could then replace (12) with the following:

$$\begin{aligned} g\ (f_2\ u\ y_1'\ y_2')\ z_1 \rightarrow \mathbf{let}\ & x_1 = \overline{1_{f_2} 1_g}\ u\ nil\ (g\ y_2'\ (\overline{2_{f_2} 1_g}\ u\ nil\ nil\ z_1))\ z_1 \\ \mathbf{in}\ & \overline{f_2 g}\ u\ (g\ y_1'\ x_1)\ (g\ y_2'\ (\overline{2_{f_2} 1_g}\ u\ (g\ y_1'\ x_1)\ nil\ z_1))\ z_1 \end{aligned} \tag{13}$$

The replacement of the call $\overline{1_{f_2} 1_g}\ u\ nil\ nil\ z_1$ by x_1 is justified by the fact that it differs from $\overline{1_{f_2} 1_g}\ u\ nil\ (g\ y_2'\ (\overline{2_{f_2} 1_g}\ u\ nil\ nil\ z_1))\ z_1$ only in a position filled with a *nil*-symbol, which is guaranteed (by the correctness proof for Construction 3.1) not to have an impact on the computed output for any input. There still remain two calls to $\overline{2_{f_2} 1_g}$ on u. Introducing sharing of these calls as above leads to the following:

$$\begin{aligned} g\ (f_2\ u\ y_1'\ y_2')\ z_1 \rightarrow \mathbf{let}\ & x_1 = \overline{1_{f_2} 1_g}\ u\ nil\ (g\ y_2'\ x_2)\ z_1 \\ & x_2 = \overline{2_{f_2} 1_g}\ u\ (g\ y_1'\ x_1)\ nil\ z_1 \\ \mathbf{in}\ & \overline{f_2 g}\ u\ (g\ y_1'\ x_1)\ (g\ y_2'\ x_2)\ z_1 \end{aligned} \tag{14}$$

Note that now we have a circular binding because x_1 depends on x_2 and vice versa. This will have a tremendous impact on the transformed program because it will necessitate lazy evaluation for ensuring termination, whereas mtts and extensions thereof by noncircular **let**-expressions are guaranteed to terminate under any evaluation strategy (on completely defined, finite input). However, we claim that introducing circularities is unavoidable to maximize sharing here. In particular, when insisting on acyclic graphs as rule right-hand sides we would have to use either (13) or the following replacement for (12):

$$\begin{aligned} g\ (f_2\ u\ y_1'\ y_2')\ z_1 \rightarrow \mathbf{let}\ & x_2 = \overline{2_{f_2} 1_g}\ u\ (g\ y_1'\ (\overline{1_{f_2} 1_g}\ u\ nil\ nil\ z_1))\ nil\ z_1 \\ \mathbf{in}\ & \overline{f_2 g}\ u\ (g\ y_1'\ (\overline{1_{f_2} 1_g}\ u\ nil\ (g\ y_2'\ x_2)\ z_1))\ (g\ y_2'\ x_2)\ z_1 \end{aligned} \tag{15}$$

Using (13) may lead to a superfluous traversal with $\overline{2_{f_2} 1_g}$, whereas using (15) may lead to a superfluous traversal with $\overline{1_{f_2} 1_g}$. In general, it is impossible to decide statically which of the two options to use because it can depend on the concrete input provided for u which is preferable with respect to efficiency. Hence, our integration of tree transducer composition and tupling is based on the choice (14). Thus, it shares ideas with the *circular tupling* transformation of [Bir84].

In [Voi02c, Voi04b] we have formally specified a transformation called *lazy composition* that eliminates intermediate results inside accumulating parameters but avoids the introduction of multiple traversals. Here we only want to illustrate lazy composition by applying it to our running example without presenting the general transformation in detail. Actually, in [Voi02c, Voi04b] only the special case was considered that the producer and the consumer have exactly one state each, but from the example below it should become clear how the technique can be extended to handle mutual recursion at least in the producer.[10] For the output program,

[10]While being restricted to one state each for the producer and the consumer, the programs in [Voi02c, Voi04b] had a slightly more liberal syntactic scheme for rule right-hand sides with respect to the use of recursion variables and with respect to external calls. These papers also proposed some other generalizations which are not relevant for the key ideas of lazy composition and hence will not be considered here.

which clearly will not be an mtt anymore, we use Haskell notation [Pey03] but take some freedom with regard to symbol names, which may appear in different types and with different ranks. Just like in tree transducer composition, the key idea is to translate the right-hand sides of M_1's rules with the state g of M_2 and to provide an $\overline{fg}$-state with translations of all context parameters of f with g. But instead of introducing auxiliary states to compute the appropriate context parameters of g on the context parameters of f, this information is additionally provided by $\overline{fg}$ itself. More precisely, $\overline{fg}$ is tupled together with all $\overline{k_f l_g}$-states for $k \in [rank_F(f) - 1]$ and $l \in [rank_G(g) - 1]$. To see how this works in the transformation, let us consider the construction of a concrete rule for our running example.

Assume we want to construct the rule for $\overline{f_1 g}$ at input symbol β. Part of its right-hand side can be obtained by translating $rhs_{M_{spine},f_1,\beta}$ with g, but additionally we want to compute the information usually provided by $\overline{1_{f_1} 1_g}$. The right-hand side will thus be a pair, the second component of which is yet unknown:

$$\overline{f_1 g}\ (\beta\ u_1)\ y_{1,g}\ z_1 = (g\ (f_2\ u_1\ y_1\ (\beta\ \epsilon))\ z_1, ?)$$

The nesting of g and f_2 needs to be replaced by a call to $\overline{f_2 g}$, requiring translations of f_2's context parameters with g. Since $\overline{f_2 g}$ is assumed to return a triple additionally providing the information usually computed by $\overline{1_{f_2} 1_g}$ and $\overline{2_{f_2} 1_g}$, we can introduce a circular binding as follows:

$$\overline{f_1 g}\ (\beta\ u_1)\ y_{1,g}\ z_1 = \mathbf{let}\ (x, x_1, x_2) = \overline{f_2 g}\ u_1\ (g\ y_1\ x_1)\ (g\ (\beta\ \epsilon)\ x_2)\ z_1\ \mathbf{in}\ (x, ?)$$

Compare this step to the rule (14) above. In essence, we have merged the calls to $\overline{f_2 g}$, $\overline{1_{f_2} 1_g}$, and $\overline{2_{f_2} 1_g}$ into a single one (and, of course, used concrete replacements for u, y_1', y_2', and z_1). Now, we can rewrite the subexpression $g\ y_1\ x_1$ to $y_{1,g}$, referring to the g-translation of f_1's context parameter as passed to $\overline{f_1 g}$. But in contrast to the situation in Construction 3.1, where the appropriate *Pre*-rule would simply discard the context parameter of g, the x_1 here is exactly the value that $\overline{f_1 g}$ promises to return in its second result component, namely the context parameter of g on reaching the first context parameter of f_1. This insight leads to

$$\overline{f_1 g}\ (\beta\ u_1)\ y_{1,g}\ z_1 = \mathbf{let}\ (x, x_1, x_2) = \overline{f_2 g}\ u_1\ y_{1,g}\ (g\ (\beta\ \epsilon)\ x_2)\ z_1\ \mathbf{in}\ (x, x_1)$$

from which two reductions using rules of M_{pfx} to eliminate the intermediate symbols β and ϵ finally give the following:

$$\overline{f_1 g}\ (\beta\ u_1)\ y_{1,g}\ z_1 = \mathbf{let}\ (x, x_1, x_2) = \overline{f_2 g}\ u_1\ y_{1,g}\ (\beta\ (\epsilon\ x_2))\ z_1\ \mathbf{in}\ (x, x_1)$$

Note that even though x_1 was moved out of the binding when rewriting $g\ y_1\ x_1$ to $y_{1,g}$, there still is a circularity, namely in x_2. Nevertheless, the resulting program will be well-behaved under lazy evaluation, as argued in [Voi04b] by appealing to the correctness proof for Construction 3.1, where a potential circularity in the nesting of translations is safely broken at transformation time (through the introduction of *nil*-symbols in the definition of $nest_f$). In lazy composition the resolution of dependencies is instead deferred to runtime, relying on the lazy evaluation mechanism

to order computations appropriately. That this is possible is argued based on the syntactic restrictions to which M_1 and M_2 are subjected, guaranteeing that the context parameters of the consumer on reaching the context parameters of the producer are always uniquely determined and thus in particular cannot depend circularly on themselves. The syntactic restrictions also guarantee that during the construction of a rule like above there will never be two different candidates to replace the same question mark position in the result tuple. In particular, a weakly single-use mtt M_2 with exactly one state is an htt or is recursion-linear. Together with context-linearity of M_1 this suffices to guarantee that no y_k in any rule right-hand side of M_1 can be reached by two different calls of g.

Constructing all new rules for our running example in exactly the same way as shown for the rule for $\overline{f_1 g}$ at β above yields the following program:

$$
\begin{array}{lll}
\overline{f_1 g}\ (\alpha\ u_1)\ y_{1,g}\ z_1 & = \textbf{let}\ (x, x_1) = \overline{f_1 g}\ u_1\ y_{1,g}\ z_1\ \textbf{in}\ (\alpha\ x, x_1) \\
\overline{f_1 g}\ (\beta\ u_1)\ y_{1,g}\ z_1 & = \textbf{let}\ (x, x_1, x_2) = \overline{f_2 g}\ u_1\ y_{1,g}\ (\beta\ (\epsilon\ x_2))\ z_1\ \textbf{in}\ (x, x_1) \\
\overline{f_1 g}\ \epsilon\ \ y_{1,g}\ z_1 & = (y_{1,g}, z_1) \\
\overline{f_2 g}\ (\alpha\ u_1)\ y_{1,g}\ y_{2,g}\ z_1 & = \textbf{let}\ (x, x_1) = \overline{f_1 g}\ u_1\ (\delta\ y_{1,g})\ z_1\ \textbf{in}\ (\alpha\ x, y_{2,g}, x_1) \\
\overline{f_2 g}\ (\beta\ u_1)\ y_{1,g}\ y_{2,g}\ z_1 & = \textbf{let}\ (x, x_1, x_2) = \overline{f_2 g}\ u_1\ y_{1,g}\ (\beta\ (\beta\ y_{2,g}))\ z_1\ \textbf{in}\ (x, x_1, x_2) \\
\overline{f_2 g}\ \epsilon\ \ y_{1,g}\ y_{2,g}\ z_1 & = (\delta\ y_{1,g}, y_{2,g}, z_1)
\end{array}
$$

Analogous steps yield that the expression $e_{M_{pfx}}[v_1 \leftarrow e_{M_{spine}}] = g\ (f_1\ u_1\ \epsilon)\ \gamma$ should be replaced by $\textbf{let}\ (x, x_1) = \overline{f_1 g}\ u_1\ (\epsilon\ x_1)\ \gamma\ \textbf{in}\ x$. Figure 7.c contains a computation of the thus produced circular program, where tupled bindings are represented by explicit occurrences of selector functions ($pr_{m,n}$ denotes the projection on the nth element of an m-tuple). For the sake of clarity, the reduction shown in the figure deviates from the lazy evaluation strategy by first evaluating three calls to $\overline{f_1 g}$ and $\overline{f_2 g}$, leading to a circular data structure, and only in the end evaluating seven tuple selectors, recovering an ordinary tree. Under lazy evaluation these two phases would be intertwined, but the total number of steps performed would not be different.

Comparing Figure 7.c to the computations for different program versions on the same input in Figures 3.b, 4.a, and 7.b seems to suggest that lazy composition is not worth the effort because the circular program requires more reduction steps than any of the other programs. Indeed, while in contrast to the original program and to the program obtained by classical deforestation the complete intermediate tree is avoided, and while in contrast to the program obtained by tree transducer composition multiple traversals are avoided, there are additional costs imposed by the tuple nodes (which are also a kind of intermediate data structure) and the selector functions. However, in practice the circular program could nevertheless be quite efficient because evaluation of a tuple selector function is likely much cheaper than reduction using, e.g., a general mtt rule. In particular, several compiler optimizations for functions that yield multiple results in tuples have been proposed in the literature [Wad87, Spa93, Gro00]. Together with a post-processing presented in [Voi04b] that is tailored more specifically to the kind of programs produced by lazy composition, they may reduce the additional ballast caused by the introduction

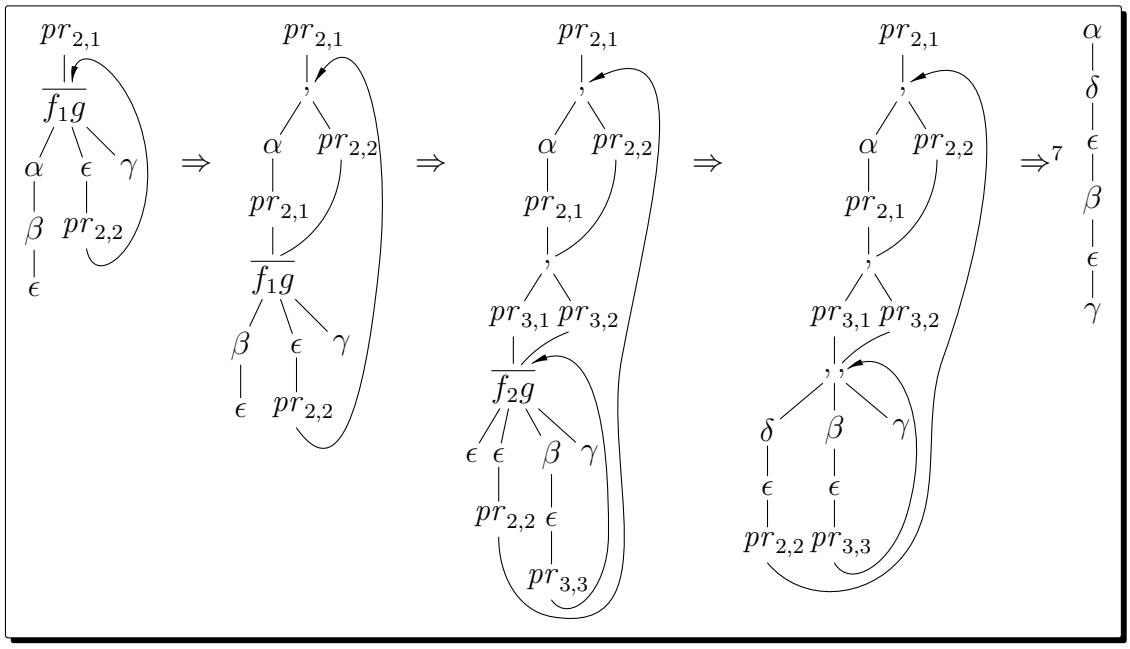

Figure 7.c: Reduction sequence using the program produced by lazy composition

of tuples considerably. On the other hand, the dependency of a circular program on lazy evaluation can run counter to benefits that the other program versions might have gained from strictness analysis. The mixed blessing of avoiding intermediate results and multiple traversals by circular tupling is also evident from the measurements performed in [Voi04b]. That the syntactic form and the basic computation steps of the circular programs are so fundamentally different from those of mtts also makes a formal efficiency analysis for lazy composition more complicated than for tree transducer composition or for classical deforestation.

It is worth pointing out that a transformation very similar to lazy composition in the use of tupling and circular bindings was independently developed in [Nis02] by combining shortcut deforestation (see next section) with a higher-order removal phase. The author informally compares his technique with our work on tree transducer and lazy composition ([Voi01], [Voi02c], and the technical report predecessor of [VK04a]) and observes that the transformational powers match quite closely. We agree with this assessment, even though the establishment of a formal correspondence is still open. The technique from [Nis02] was recently extended in [Nis04] by allowing a restricted use of stacks in the producer of an intermediate result and by allowing mutual recursion in both the producer and the consumer. The liberal use of mutual recursion comes at the price of tupling together all functions into a single one. In particular, for our running example the functions $\overline{f_1 g}$ and $\overline{f_2 g}$ would be combined into a single one, just as would be f_1 and f_2 to bring the original program into a form that can be handled by the technique from [Nis04]. This is a bit overkill because neither in the case of f_1 and f_2 nor in the case of $\overline{f_1 g}$ and $\overline{f_2 g}$ there are

multiple traversals to avoid[11], so the introduction of extra tuples does not pay off. On the other hand, our lazy composition technique also seems to require a tupling of all computation of the transformed program into a single function if we want to handle mutual recursion also in the consumer rather than only in the producer.

With respect to efficiency and formal efficiency analysis, the same reservations apply to the techniques in [Nis02, Nis04] as to lazy composition. In the presence of stacks it even seems that extra stack operations are introduced for the sake of proceeding with the transformation, without any guarantee that this overhead is revoked by later transformation steps. In conclusion, we do not consider lazy composition and related techniques as mature for integration into a compiler yet (in contrast to Construction 3.1, for which we have established formal efficiency guarantees and, not to forget, semantics-preservation for finite, partial, and infinite trees). However, they provide an interesting alternative account of the ideas behind elimination of intermediate results by tree transducer composition. In particular, this account seems to benefit an audience that feels comfortable about the introduction of circular dependencies in exchange for the intricate "walking upwards"- and "nesting translations"-schemes needed in Construction 3.1. This seems to be the case for many functional programmers because the "tying the knot"-method from [Bir84] is folklore in that community.

7.4 Free theorems-based program transformations

The transformations considered so far largely insist on quite strong restrictions on the syntactic form of the input program. A somewhat more liberal approach is inherent in transformations based on so called *free theorems* [Wad89b]. Such theorems can be derived from function types, which play no prominent role for the transformations considered in the previous sections. The most popular transformation based on free theorems is shortcut deforestation, achieving elimination of intermediate results by using certain combinators, compositions of which can be transformed by *cata*/*build*-rules or variants thereof [GLP93, Gil96, Chi99, Joh02]. The key ideas are to uniformly abstract from all output symbols in the producer of an intermediate result and to standardize the consumption of intermediate results using a fixed recursion pattern. We first illustrate the former on our running example.

Example 7.6 (uniform abstraction from output symbols in M_{spine})
Consider the following combinator (again using Haskell notation [Pey03]):

$$build_{\Delta_{tree}} :: (\forall \nu.\ (\nu \to \nu \to \nu) \to (\nu \to \nu) \to (\nu \to \nu) \to \nu \to \nu) \to T_{\Delta_{tree}}$$
$$build_{\Delta_{tree}}\ h = h\ \delta\ \alpha\ \beta\ \epsilon$$

It takes a function h and applies it to the symbols of Δ_{tree}, interpreted as functions over $T_{\Delta_{tree}}$ (e.g. $\delta :: T_{\Delta_{tree}} \to T_{\Delta_{tree}} \to T_{\Delta_{tree}}$). Thus, h can be seen as an abstract

[11] In both cases the two functions in question consume the input in an alternating fashion, but no input symbol is processed twice.

tree, from which a concrete tree is obtained through $build_{\Delta_{tree}}$. The essential bit guaranteeing that **all** output symbols are properly abstracted in h is the type of $build_{\Delta_{tree}}$. The "$\forall\nu$" there means that h has to be *polymorphic*, so it cannot use concrete symbols to build its result.[12] Using $build_{\Delta_{tree}}$, the producer from the running example can be expressed as follows:

$$\begin{aligned}\tau_{M_{spine}} = \lambda t \to build_{\Delta_{tree}}\ (\lambda d\ a\ b\ e \to \textbf{let}\ & f_1\ (\alpha\ u_1)\ y_1 = a\ (f_1\ u_1\ y_1)\\ & f_1\ (\beta\ u_1)\ y_1 = f_2\ u_1\ y_1\ (b\ e)\\ & f_1\ \epsilon\ y_1 = y_1\\ & f_2\ (\alpha\ u_1)\ y_1\ y_2 = a\ (f_1\ u_1\ (d\ y_1\ y_2))\\ & f_2\ (\beta\ u_1)\ y_1\ y_2 = f_2\ u_1\ y_1\ (b\ (b\ y_2))\\ & f_2\ \epsilon\ y_1\ y_2 = d\ y_1\ y_2\\ \textbf{in}\ & f_1\ t\ e)\end{aligned}$$

It should be straightforward to see that every mtt can be brought into such a form. □

While apart from its polymorphic type there are no further restrictions on the producer of an (abstracted) intermediate result in shortcut deforestation, the consumer needs to be expressed as a *catamorphism*. This captures function definition by structural recursion in a similar fashion as mtts do, just in combinator style rather than as explicit program schemes. And indeed, from every mtt one can synthesize a catamorphism representation in a systematic way, using tupling to deal with mutual recursion and lambda-abstraction to deal with context parameters.

Example 7.7 (M_{pfx} as a catamorphism)
Consider the following polymorphic combinator:

$$\begin{aligned}&cata_{\Delta_{tree}} :: \forall\nu.\ (\nu \to \nu \to \nu) \to (\nu \to \nu) \to (\nu \to \nu) \to \nu \to T_{\Delta_{tree}} \to \nu\\ &cata_{\Delta_{tree}}\ d\ a\ b\ e\ (\delta\ v_1\ v_2) = d\ (cata_{\Delta_{tree}}\ d\ a\ b\ e\ v_1)\ (cata_{\Delta_{tree}}\ d\ a\ b\ e\ v_2)\\ &cata_{\Delta_{tree}}\ d\ a\ b\ e\ (\alpha\ v_1) = a\ (cata_{\Delta_{tree}}\ d\ a\ b\ e\ v_1)\\ &cata_{\Delta_{tree}}\ d\ a\ b\ e\ (\beta\ v_1) = b\ (cata_{\Delta_{tree}}\ d\ a\ b\ e\ v_1)\\ &cata_{\Delta_{tree}}\ d\ a\ b\ e\ \epsilon = e\end{aligned}$$

It takes four values whose types correlate with the ranks of the symbols in Δ_{tree} and consumes a tree in $T_{\Delta_{tree}}$ by replacing every symbol in it with the associated value. Using $cata_{\Delta_{tree}}$, the consumer from the running example can be expressed as follows:

$$\begin{aligned}\tau_{M_{pfx}} = \lambda t \to cata_{\Delta_{tree}}\ & (\lambda g_1\ g_2\ z_1 \to \delta\ (g_1\ (g_2\ z_1)))\\ & (\lambda g_1\ z_1 \to \alpha\ (g_1\ z_1))\\ & (\lambda g_1\ z_1 \to \beta\ (g_1\ z_1))\\ & (\lambda\ z_1 \to \epsilon\ z_1)\\ & t\ \gamma\end{aligned}$$

[12] Note that $build_{\Delta_{tree}}$ itself is not polymorphic because there is no $\forall$ at the outermost level of its type. Rather, $build_{\Delta_{tree}}$ is a (monomorphic) function taking a polymorphic value as argument. This is called *rank-2 polymorphism* and is supported by most current Haskell implementations.

Note that the catamorphism used here is higher-order in the sense that the type variable ν is instantiated to a function type $(T_{\Omega_{mon}} \to T_{\Omega_{mon}})$. Tupling is not required in this example because M_{pfx} has only one state. □

The heart of shortcut deforestation is a rule, per data type, expressing how producer and consumer can be fused. For our example this rule looks as follows:

$$cata_{\Delta_{tree}}\ d\ a\ b\ e\ (build_{\Delta_{tree}}\ h) = h\ d\ a\ b\ e \tag{16}$$

It expresses that instead of applying an abstracted producer h to concrete symbols (via $build_{\Delta_{tree}}$) and afterwards replacing all occurrences of those symbols in the result with some appropriately typed values (via $cata_{\Delta_{tree}}$), one can directly pass those values to h, thus circumventing the construction of the intermediate result.

Example 7.8 (shortcut deforestation for the running example)
Inlining the definitions from Examples 7.6 and 7.7 in $\tau_{M_{spine}}\,;\tau_{M_{pfx}} = \lambda t \to \tau_{M_{pfx}}\ (\tau_{M_{spine}}\ t)$, applying rule (16), and performing a series of beta-reductions yields the following:

$$\begin{aligned}
\tau_{M_{spine}}\,;\tau_{M_{pfx}} = \lambda t \to (\textbf{let}\ & f_1\ (\alpha\ u_1)\ y_1 && = \lambda z_1 \to \alpha\ (f_1\ u_1\ y_1\ z_1)\\
& f_1\ (\beta\ u_1)\ y_1 && = f_2\ u_1\ y_1\ (\lambda z_1 \to \beta\ (\epsilon\ z_1))\\
& f_1\ \epsilon\ \ \ y_1 && = y_1\\
& f_2\ (\alpha\ u_1)\ y_1\ y_2 && = \lambda z_1 \to \alpha\ (f_1\ u_1\\
& && \qquad (\lambda z_1 \to \delta\ (y_1\ (y_2\ z_1)))\ z_1)\\
& f_2\ (\beta\ u_1)\ y_1\ y_2 && = f_2\ u_1\ y_1\ (\lambda z_1 \to \beta\ (\beta\ (y_2\ z_1)))\\
& f_2\ \epsilon\ \ \ y_1\ y_2 && = \lambda z_1 \to \delta\ (y_1\ (y_2\ z_1))\\
\textbf{in}\ & f_1\ t\ (\lambda z_1 \to \epsilon\ z_1))\ \gamma &&
\end{aligned}$$

Here the tree over Δ_{tree} has been eliminated, but in its stead sequences of suspended function calls are accumulated, which are also a form of intermediate results. □

The observed introduction of suspended function calls compromises the intended efficiency gain by shortcut deforestation dramatically, as can be seen from the comparative measurements in [VK04a] and [Voi04b]. For this reason, [Sve02] disputes the use of higher-order catamorphisms and instead proposes the *destroy/unfoldr*-rule (for the list case, but the generalization to other algebraic data types is straightforward) as a dual to *cata/build*. Then, consumers using accumulating parameters can be expressed without resorting to the introduction of extra lambda-abstractions as above. However, producers of intermediate results then need to be in the restricted form of *anamorphisms*, so that the more difficult problem of **producers** using accumulating parameters is not approached. In summary, the effect to eliminate intermediate results in accumulating parameters as accomplished by tree transducer composition (and variants like lazy composition) cannot be achieved by shortcut deforestation in general. For the special case of tdtts the two techniques correspond to each other [JV04c], but in the presence of context parameters shortcut deforestation fails to deliver good results.[13]

[13]Note that [Nis02] proposes a higher-order removal phase that turns the result of shortcut deforestation back into a first-order program, thus eliminating the offending overhead. However,

Of course, shortcut deforestation has other merits because it is applicable also to functions that form no mtts. For example, the *destroy/unfoldr*-rule successfully handles consumers recursing over several intermediate results simultaneously. Moreover, rules like (16) impose only very mild restrictions on the producer of the intermediate result, which does not have to follow a specific recursion pattern. This is possible because the correctness proof for (16) depends only on the type of h. Such free theorems derived from (polymorphic) types have first been proposed in [Wad89b], building on work on *parametricity* [Rey83]. They can also be applied beyond the limits of shortcut deforestation to transfer ideas from the theory of tree transducers to functional programs in general, as will be recollected next.

In [KGK01] a program transformation is presented that aims at efficiency improvement through the introduction of accumulating parameters. The input programs are particular *modular tree transducers* [EV91] consisting of essentially a tdtt that additionally may contain external calls—not necessarily adhering to the principle of structural descent—to an mtt defining tree substitution functions (see acc' and *sub* in Section 7.6 for an example). A specific such substitution function is list concatenation. By "freezing" external calls as special output symbols, the tdtt module can be turned into a proper tdtt, whose output is interpreted by a new function—essentially an htt—with external calls to the substitution mtt. Since the interpretation and substitution functions can be integrated into a single mtt (employing associativity of substitutions) and since that mtt can be composed with the frozen tdtt, one ends up with an mtt that is semantically equivalent to the original modular program but avoids repeated substitutions through the use of its context parameters.[14] In [Voi02a] we observed that the freezing and integration steps are restricted neither to the framework of tree transducers nor to substitution functions. By combining freezing with uniform abstraction from output symbols, expressing the result of integration as a catamorphism, and replacing tree transducer composition with shortcut deforestation, we arrived at so called *vanish*-combinators that capture optimization of repeated data manipulations in a very succinct way. In contrast to *cata/build*-rules they do not only eliminate intermediate results but also the costs attached, e.g., to repeated list concatenations when used in a left-associative way. In particular, the approach in [Voi02a] generalizes the classical accumulation technique of [Wad89a] but is also applicable to nested lists and to operations like list

this only works by yet again imposing strong syntactic restrictions (corresponding to those in Construction 3.1) on the original program and comes at the price of introducing tuples and circular bindings. See the previous section for a more detailed discussion.

[14] Very similar ideas are employed in [PS04] in the context of *forest transducers*. A *macro forest transducer* (for short *mft*) is an (in general nondeterministic) mtt that may additionally use concatenation on binary encodings of lists of trees. By analogs of freezing and integration every mft can be simulated by two consecutive mtts. Moreover, for the special case of *top-down forest transducers* (i.e. mfts without context parameters) [PS04] present a direct transformation into an mtt that corresponds to the fusion of the freezing, integration, and composition steps from [KGK01] in [Mal02]. The latter work also generalizes the technique from [KGK01] to context-linear mtt modules with external calls to substitution functions by using our Construction 3.1 instead of the more restricted composition from [Eng81].

reversal or polymorphic mapping and filtering over lists. Even though only lists were explicitly considered in [Voi02a], the methodology extends to other algebraic data types and then also to the tree substitution functions of [KGK01]. But in contrast to the latter work *vanish*-combinators are applicable to arbitrary functions—rather than just tree transducers—from which output symbols and data manipulating operations have been abstracted uniformly in an analogous manner as with *build* for shortcut deforestation. This strength again comes from free theorems.

A bit off-topic, but still noteworthy here, are the recent results of [JV04a, JV04b]. There, free theorems for lazy functional languages containing a polymorphic strict evaluation primitive are studied. Such a primitive is included in Haskell in the form of *seq*, which is widely used in fine-tuning performance by enforcing different evaluation orders than the default one. Even though it was recognized early on that *seq* has nontrivial impacts on Haskell's semantics and in particular compromises the validity of standard free theorems, the precise consequences of its inclusion into the language remained a blind spot for long. Extending on ideas from Appendix B of [Voi02a] and refuting some conventional wisdom, this gap between free theorems as a tool, in principle, to establish correctness of program transformations and the realities of mixed lazy and strict evaluation in Haskell was finally closed in [JV04a] (under the standard assumption of a parametric model for Haskell without *seq* as justified by the results of [Pit00]). By moving from equational statements to inequational ones (with respect to semantic approximation) and by crucial changes to the type-directed construction of free theorems to respect—and properly propagate—the change of extensionality at function types, namely that in the presence of *seq* an undefined function is not the same as a defined function that returns an undefined value for every argument supplied to it, free theorems were restored to their former glory. Applying the developed machinery to particular program transformations led to a number of startling insights about shortcut deforestation and related techniques [GLP93, Sve02, Voi02a], reported in detail in [JV04b].

It should be noted that even though shortcut deforestation is probably the most popular and well-known transformation technique for eliminating intermediate results, there are no formal efficiency guarantees whatsoever for it. The same is true for other transformations based on free theorems. This can be considered the downside of the generality achieved by depending only on the types of (some) functions rather than on the syntactic form of their definitions. An interesting direction for future investigation would be to see whether a similar approach as in [JV04a], using a transition from equational to inequational free theorems, but with respect to evaluation costs rather than with respect to definedness, could lead to efficiency statements about free theorems-based transformations. Our vision here is to combine the approach of [San96b] or [MS99] with that of [Pit00].

7.5 Formal efficiency analysis

7.5.1 Efficiency analysis for functional programs in general

The idea to externalize an intensional property of a computation is not new. For example, in [Ros89] for every first-order functional program a step-counting version is produced that—when called with the same arguments—returns the number of call-by-value reduction steps performed by the original program. Then, an abstract interpretation of this step-counting program is used to express the worst-case complexity (in dependence on the input size) by recurrence equations. The ultimate goal is to transform these equations into a closed expression, which unfortunately is feasible only for subclasses of programs. In [LG01] more detailed measures than just the number of reduction steps are used by counting different primitive operations separately, thus allowing to compare evaluation costs (for concrete inputs) along different dimensions.

The techniques for strict languages cannot easily be adapted to languages with nonstrict semantics because the number of call-by-name or call-by-need reduction steps performed for a certain program expression is not a compositional property. Since the computation time invested in evaluating a function argument depends on how much of the argument's value is actually "needed", the total cost of a function application cannot be understood solely from the cost properties of its ingredients in isolation. In [Wad88] step-counting versions computing upper bounds for the number of call-by-need steps are derived by formally describing this "need" using strictness analysis (see also the related approach in [BH89]). In [San90] this technique is extended by considering a higher-order language and also deriving lower bounds.

To avoid the problem of having to compute exact demand information, [San95] developed a more operational approach leading to a theory of call-by-name cost equivalence, an axiomatization of which is given by annotating functional programs with a special identity function that represents a single "tick" of computation time and by stating a set of laws that can be used to derive statements about the relative efficiency of program expressions. With a sharp eye, some of these laws can be recognized in the annotation schemes (for counting the call-by-name steps of the original and of the composed program) from Section 4.3. In [GS01] syntactic means similar to tick-symbols—so called *space gadgets*—are used in reasoning about asymptotic (i.e. only up to constant factors) **space** complexity.

While the papers mentioned in this section so far are concerned with deriving explicit cost bounds for a given program or with comparing the efficiency of two concrete programs that compute the same result but are otherwise unrelated (at least not related in a formal, systematic way), our aim was to analyze on a general level the relative efficiency for pairs of two closely connected programs, namely the input and output programs of Construction 3.1. In this respect our aim is similar to that of [San96b], where a call-by-name improvement relation (a nonsymmetric counterpart to the cost equivalence of [San95]) is used to prove the correctness—and, more as a side product, also call-by-name efficiency nondeterioration—of unfold/fold-

transformations. In [San96a] that machinery is applied to classical deforestation. A key difference separating our work from that of Sands is that, unlike unfold/fold-transformations, our construction cannot be understood as a sequence of locally equivalence-preserving steps, but rather must be considered as a whole in one go. As a consequence, also the "bookkeeping" of tick-symbols for the transformed program cannot be performed in a similarly modular fashion as in [San96b]. Instead, we had to very carefully analyze the composition construction to invent the annotation scheme in Section 4.4 so that eventually Lemma 4.12 could be proved. The same aspect prevents the application—to our construction—of the call-by-need variant of the approach from [San96b] developed in [MS99]. The latter succeeded to prove that the accumulation technique of [Wad89a] can never slow down programs by more than a constant factor (in contrast to the additive constant that we allow ourselves) with respect to the number of reduction steps of a lazy abstract machine, but in general the call-by-need variant seems to be less readily applicable than the original call-by-name improvement theory even to program transformations based on local equivalences. In particular, no successful application to classical deforestation is known (which, of course, would only make sense if leading to something more than efficiency nondeterioration for all linear input programs).

7.5.2 Efficiency analysis for tree transducer composition

Theorems 4.45 and 4.47 are sufficient to formally establish call-by-need efficiency nondeterioration for many typical examples of tree transducer composition for which previously efficiency nondeterioration was assured by ad-hoc reasoning or by experiments only. In particular, this is the case for the various such examples considered in [VK04a]. Our results also capture all the examples for tree transducer composition from [KV01], where the pure elimination of intermediate results was considered as success without regard to actual numbers of reduction steps. Moreover, those examples show that our proposed adjustments to the efficiency analysis in Sections 6.2 and 6.3 to take partial demand and tail calls into account work reasonably well in practice. Namely, for all examples in [KV01] and [VK04a] for which tree transducer composition de facto leads to an improvement with respect to the number of call-by-need steps to reach normal form, our analysis adjusted along the lines of Section 6.2 can establish a corresponding improvement even under partial demand. For all those examples except for one in [KV01] that corresponds to Example 6.7, the improvement holds also if one additionally takes the relative cheapness of tail calls into account. Of these, in nine cases our analysis adjusted for tail call-awareness as proposed in Section 6.3 is already sufficient, whereas in only four cases it would reject a transformation even though de facto an improvement holds also when taking tail calls into account.

Our results also improve on formal efficiency considerations performed in [Küh99] and [Höf99] for the special case that one of the mtts to be composed is a tdtt and without regard to partial demand or tail calls.[15] More precisely, the respective

[15]That these omissions can make a difference also in the special case of one of the mtts being

results of [Küh99], namely that the composed program is at least as efficient as the original program if M_1 is a linear tdtt and M_2 is linear (Corollary 21) or M_1 is linear, but not a tdtt, and M_2 is a linear htt (Theorem 23), are covered and generalized by Theorem 1 of [Voi02b], and thus by Theorem 4.45 of this thesis. The same is true for Conjectures 3.13[16] and 3.21 of [Höf99], together suggesting an efficiency improvement if M_1 is a tdtt and M_2 is linear. Theorem 4.34 of [Höf99] states an efficiency improvement for the case that M_1 is producing and context-nondeleting and M_2 is a nondeleting htt that is not linear. Theorem 2 of [Voi02b]—repeated as Theorem 4.47 in this thesis—gives (with $\lambda = 0$, using the fact that all $\uparrow$-rules are in $\mathcal{E}_{M_1,M_2}$ if M_1 is context-nondeleting and M_2 is nondeleting) a more general result in that M_2 is allowed to be any nondeleting tdtt. For the generalizations obtained through Theorem 4.45 we additionally get for free that the improvements hold even with regard to partial demand and in consideration of tail calls because by the observations in Sections 6.2 and 6.3 those aspects do not make a difference for the specialization of Theorem 4.45 to the case that one of the involved mtts is a tdtt. The only result regarding tree transducer composition in [Höf99] that goes beyond our theorems is Theorem 4.21 of that work, stating an efficiency improvement if M_1 is context-nondeleting and M_2 is a linear, nondeleting htt. We conjecture that context-nondeletingness of M_1 and nondeletingness of M_2 can be dropped here and that the improvement holds also when taking partial demand and tail calls into account.

Note that [Küh99] and [Höf99] also include classical deforestation in the efficiency comparison. For the case that M_1 is not a tdtt (i.e., when tree transducer composition and deforestation are really different) but M_2 is, this leads to results that are not in the scope of this thesis. However, the annotation scheme suggested in Section 7.2 provides the starting point for a comparative analysis of classical deforestation using the approach followed in this thesis, which would hopefully lead to more general results than obtained through ad-hoc reasoning (as was the case for the analysis of tree transducer composition alone).

7.6 Deaccumulation — Improving provability

So far, transformation of programs has been motivated by the desire to improve their efficiency. While this is indeed the prevailing motivation for most work on program transformation, there are also other aspects of a program that one might want to affect without changing its semantics. For example, in [Gie00] the transformation of programs to make them more suitable for induction proofs was pioneered.

a tdtt is evidenced by a counterexample that fulfills the conditions from Theorem 4.34 of [Höf99] but is not improving under every partial demand.

[16] Actually, this is Theorem 3.13 in [Höf99], but its proof is flawed by depending on a lemma which is wrong. Unfortunately, this affects also Theorem 3.16 of [Höf99], which moreover cannot be repaired using a different proof as there is a counterexample with a producing tdtt and a context-nondeleting mtt whose composition is less efficient than the original program. We also found counterexamples to two further conjectures regarding efficiency improvement by tree transducer composition in [Höf99].

An alternative technique with complementary strength was developed in [GKV03]. Since the latter is in interesting contrast to the other transformations considered in this thesis, both in its sacrifice of efficiency for the sake of provability and in its use of decomposition rather than composition of tree transducers, we use an example to illustrate it in this section.

Recall the mtts M_{rep} and M_{acc} from Example 6.1. There, we claimed that for every $t \in T_{\Sigma_{nat}}$: $\tau_{M_{acc}}(t) = \tau_{M_{rep}}(t)$, i.e., $nf(\Rightarrow_{R_{acc}}, acc\ t\ \epsilon) = nf(\Rightarrow_{R_{rep}}, rep\ t)$. Proofs of such claims are typically done by structural induction on the input tree. That is, one considers every possible choice for the root symbol and tries to prove appropriate instances of the claim by using the rewrite rules of the given program and appealing to the claim for proper subtrees as induction hypotheses. For the example such a proof attempt would look as follows, where the explicit mentioning of normal forms is dropped from now on so as not to cloud calculations and the discussion below in too much notation:

$\underline{t = \epsilon}$:
$acc\ \epsilon\ \epsilon = \epsilon = rep\ \epsilon$, by acc and rep

$\underline{t = \alpha\ t_1}$ for some $t_1 \in T_{\Sigma_{nat}}$:

$acc\ (\alpha\ t_1)\ \epsilon$
$=$ (by acc)
$acc\ t_1\ (\alpha\ \epsilon)$
$=$ (by ???)
$\alpha\ (acc\ t_1\ \epsilon)$
$=$ (by the induction hypothesis for t_1)
$\alpha\ (rep\ t_1)$
$=$ (by rep)
$rep\ (\alpha\ t_1)$

While in the base case simple exhaustive rewriting of both sides of the claim led to a proof, this is not so in the other case, not even if the induction hypothesis is additionally applied where possible. There remains a gap between $acc\ t_1\ (\alpha\ \epsilon)$ and $\alpha\ (acc\ t_1\ \epsilon)$ that cannot be closed by rewriting because the root symbol of t_1 is unknown. It is also not possible to apply the induction hypothesis except for turning $\alpha\ (acc\ t_1\ \epsilon)$ back into $\alpha\ (rep\ t_1)$, which clearly leads nowhere. The usual strategy (both of a human and of a mechanized verifier) would be to attempt a further induction proof for the open statement, i.e., of $acc\ t_1\ (\alpha\ \epsilon) = \alpha\ (acc\ t_1\ \epsilon)$ for every $t_1 \in T_{\Sigma_{nat}}$. While the base case is again simple, the inductive case leads to the necessity to prove $acc\ t_2\ (\alpha\ (\alpha\ \epsilon)) = \alpha\ (\alpha\ (acc\ t_2\ \epsilon))$ for every $t_2 \in T_{\Sigma_{nat}}$. Proceeding in this manner will not terminate because ever bigger claims are produced. One solution is to invent suitable *generalizations* of the statements to prove. For example, here one would succeed with proving $acc\ t\ (\alpha^n\ \epsilon) = \alpha^n\ (acc\ t\ \epsilon)$ or after generalizing the original claim to $acc\ t\ (\alpha^n\ \epsilon) = \alpha^n\ (rep\ t)$, with $t \in T_{\Sigma_{nat}}$ and $n \in \mathbb{N}$ in

both cases. However, such generalizations are difficult to find systematically or even automatically. Hence, it would be desirable to instead eliminate the cause for needing generalizations in the first place.

From the failed proof attempt above it is easy to identify that the problem lies with the accumulating parameter of *acc*. In the original claim to be proved—and hence also in the induction hypothesis—it is initialized with ϵ, but during the rewriting for the concrete instantiation $t = \alpha\ t_1$ it is altered to $\alpha\ \epsilon$. This does not anymore match with the induction hypothesis, preventing a successful application of the latter. Hence, the techniques in [Gie00] and [GKV03] take the approach of eliminating accumulating parameters from a given program while preserving its semantics. Since the transformed program has the same semantics as the original one, it serves as perfect substitute in proofs about the computed output, and since it contains no accumulating parameters, induction proofs for it can be considerably easier. The main advantages over generalization techniques are that the program transformations are fully automatic and that they do not depend on the statements to be proved. In contrast, the generalization approach typically requires new invention for every new statement to be proved about a given program.

In the framework of mtts, accumulating parameters can be eliminated by decomposing an mtt into a modular tree transducer consisting of essentially a tdtt with external calls to tree substitution functions. This *decomposition* step is based on results in [Eng80, EV85, EV91] and corresponds to the inverse construction in [KGK01] to the one aimed at introducing accumulating parameters (briefly discussed in Section 7.4). The key idea, instantiated to our example, is to simulate the binary function *acc* by a new unary function acc' plus a binary tree substitution function *sub* (not to be confused with the subtree function). A special nullary symbol is used to denote places where *acc* accesses its context parameter, and *sub* is used to later substitute the actual parameters into these places. While this eliminates accumulating style in the sense that acc' has no context parameters at all and *sub*'s context parameter remains unchanged in recursive calls, the problem remains that the (transformed) induction hypothesis contains the initial value ϵ. This is resolved by identifying the context parameter from the initial expression with the special symbol used to denote places where substitution has to take place, thus encoding the initial value into the program's rewrite rules. This *constructor replacement* step is only possible under certain syntactic restrictions that are met by M_{acc}. Without going into further detail, we present the program finally obtained:

$$
\begin{array}{lll}
acc'\ (\alpha\ u_1) & \to & sub\ (acc'\ u_1)\ (\alpha\ \epsilon) \\
acc'\ \epsilon & \to & \epsilon \\
\\
sub\ (\alpha\ u_1)\ y_1 & \to & \alpha\ (sub\ u_1\ y_1) \\
sub\ \epsilon\ y_1 & \to & y_1
\end{array}
$$

The correctness of the overall transformation (called *deaccumulation*) from [GKV03] now guarantees that $acc\ t\ \epsilon$ and $acc'\ t$ are equivalent for any input. Hence, our verification problem can be solved by proving $acc'\ t = rep\ t$ for every $t \in T_{\Sigma_{nat}}$.

Note that this makes for a better induction hypothesis than before because it is free of any initial values that might lead to mismatch. The base case is again proved by simple rewriting, whereas in the other case we again end up with the obligation to prove a statement that does not directly follow from the rewrite rules or the induction hypothesis:

$\underline{t = \epsilon}$:
$acc'\ \epsilon = \epsilon = rep\ \epsilon$, by acc' and rep

$\underline{t = \alpha\ t_1}$ for some $t_1 \in T_{\Sigma_{nat}}$:

$acc'\ (\alpha\ t_1)$
$=$ (by acc')
$sub\ (acc'\ t_1)\ (\alpha\ \epsilon)$
$=$ (by the induction hypothesis for t_1)
$sub\ (rep\ t_1)\ (\alpha\ \epsilon)$
$=$ (by ???)
$\alpha\ (rep\ t_1)$
$=$ (by rep)
$rep\ (\alpha\ t_1)$

However, this time an auxiliary induction proof succeeds without problems:

$\underline{t_1 = \epsilon}$:
$sub\ (rep\ \epsilon)\ (\alpha\ \epsilon) = sub\ \epsilon\ (\alpha\ \epsilon) = \alpha\ \epsilon = \alpha\ (rep\ \epsilon)$, by rep and sub

$\underline{t_1 = \alpha\ t_2}$ for some $t_2 \in T_{\Sigma_{nat}}$:

$sub\ (rep\ (\alpha\ t_2))\ (\alpha\ \epsilon)$
$=$ (by rep)
$sub\ (\alpha\ (rep\ t_2))\ (\alpha\ \epsilon)$
$=$ (by sub)
$\alpha\ (sub\ (rep\ t_2)\ (\alpha\ \epsilon))$
$=$ (by the auxiliary induction hypothesis for t_2)
$\alpha\ (\alpha\ (rep\ t_2))$
$=$ (by rep)
$\alpha\ (rep\ (\alpha\ t_2))$

Note that for the deaccumulated program no inventiveness at all was needed in the proof. In particular, the induction hypothesis for the auxiliary proof showed up automatically during the course of the main proof. Hence, such a proof could also be produced by existing induction theorem provers without human interaction. Even in cases where deaccumulation does not lead to fully automatic proofs for the transformed program, it can significantly improve provability. For example,

Appendix A of [GKV03] contains a verification problem where the deaccumulated program still requires a generalization inside the proof, but a much simpler one than required for the corresponding original program. Also the proofs of several (further) claims in this thesis profit from deaccumulation, as briefly outlined in the following.

In Subsection 7.1.2 we asserted that $\tau_{M_{paths}} ; \tau_{M_{prev}}$ is of quadratic size increase. The main proof obligation is that $size(\tau_{M_{paths}}(t)) \leq size(t)^2$ for every $t \in T_{\Sigma_{tree}}$.[17] To succeed with a direct proof requires a nontrivial generalization of the induction hypothesis, but if we first apply deaccumulation to M_{paths}, things become simpler. We only need to take two statements about the interaction of tree substitution functions with the *size*- and $|\,.\,|_\alpha$-functions for granted, then the proof is quite straightforward. The interesting bit is that these auxiliary statements are completely generic in the sense that they do not at all depend on the mtt M_{paths} but rather derive from the general scheme of tree substitution. Hence, they could be generated and proved once and for all as lemmata about tree substitution functions, as has been proposed for properties like associativity in [GKV03]. Then, they would be readily available in proofs like the one in question for M_{paths}, where they effectively render generalization efforts superfluous because the essence of the generalization that would be necessary in a direct proof is already captured in the generic statements about *sub*. Similarly, a generic statement about the interaction between tree substitution functions and *height* on monadic trees is useful for establishing two of the properties used in the proof of Theorem 7.3, namely that $height(\tau_{M_{exp}}(t)) = 1 + 2^{height(t)-1}$ for every $t \in T_{\Sigma_{nat}}$ and $height(\tau_{M_{count}}(t'')) = size(t'') + 1$ for every $t'' \in T_{\Sigma_{tree}}$. In both cases a direct proof requires to find an appropriate generalization of the induction hypothesis, whereas after deaccumulation the proof is automatic (modulo arithmetics and the mentioned generic statement).

Of course, the value of deaccumulation stands and falls with the flexibility it allows in input programs. As indicated above, the mtts M_{acc}, M_{paths}, M_{exp}, and M_{count} from Examples 6.1, 7.1, and 7.2 fulfill the necessary syntactic restrictions (as do M_{pfx} and M_{rev} from Examples 2.2 and 6.7), but many other mtts will not do so. An extension of deaccumulation to apply to richer classes of programs is work in progress.

[17] That $\tau_{M_{prev}} ; size = size$ if restricted to outputs of $\tau_{M_{paths}}$ seems easy enough to see.

Chapter 8

Conclusion

We have developed sufficient conditions for efficiency improvement by tree transducer composition in lazy functional languages. The key was to annotate programs thus that their outputs reflect the numbers of performed reduction steps. Together with the presented "decomposition" of the respective annotation of the composed program in terms of a suitable annotation of the original program, this allows to study the effect of the composition construction on call-by-name efficiency based solely on the outputs computed by two different annotated versions of the original program. The combination of annotations and subsequent manipulations were guided by the desire to focus the analysis on the initial expression and the rules of the first mtt in a potential composition, which makes sense intuitively because that is where the intermediate result shapes up from whose elimination one expects an efficiency gain.

The relatively simple approach to call-by-need analysis by essentially comparing call-by-name efficiency of the composed program to call-by-need efficiency of the original program in special cases comes with a certain dose of pessimism and imposes additional restrictions, but not more so than in comparable efficiency studies for other transformation techniques. In fact, it seems questionable whether a more faithful analysis of the call-by-need efficiency of the composed program is at all possible in general, at least using an "externalization" approach, given that the composition construction itself proceeds intrinsically call-by-name and loses a certain amount of sharing that cannot be recovered without leaving the framework of mtts. But often the mentioned pessimism does not even come to bear because in many cases there is actually no difference between the numbers of call-by-name and call-by-need steps for the composed program. We saw this for the running example; it is also true if M_1 is a tdtt and M_2 is context-linear or basic, as well as if M_1 is basic and M_2 is a tdtt (because in these cases $\overline{M_1;M_2}$ is context-linear or basic), and we conjecture that it is true if M_1 is context-linear and M_2 is a linear tdtt.[1]

More evidence for our approach being a reasonable one comes from the inspection of its strength on typical examples and from the comparison with previous formal results about the impact of tree transducer composition on efficiency. The aspiration

[1]Then, the "at least" in Footnote 7 on page 66 could be replaced by "exactly".

for practical relevance is also supported by our consideration of partial demand and tail calls. To move further forward into the direction of more and more realistic (but still machine-independent) criteria for when a transformed program should be considered more efficient than the original one, the primary candidates for extension are the use of weighted reduction steps and the incorporation of costs associated to memory allocation into the analysis. For the former, we think that much of the basic apparatus is already in place with this thesis. Experimentation to determine a proper abstraction of the different costs incurred by different reduction steps (e.g. based on the sizes of rule right-hand sides) and incorporation of the gained insights into our annotation schemes seem to be mainly engineering tasks. Of course, it is also possible that unexpected technical complications emerge. General techniques for the analysis of space efficiency are largely terra incognita, at least compared to the much more well-studied area of time cost analysis. It would be interesting to investigate what insights the inspiring work of [GS01] has to offer in this respect for our transformation.

Bibliography

[AGS03] D.F.R. van Arkel, J.H.G. van Groningen, and S. Smetsers. Fusion in practice. In *14th International Workshop on Implementation of Functional Languages — IFL'02, Revised Selected Papers*, volume 2670 of *LNCS*, pages 51–67. Springer-Verlag, 2003.

[Bau03] A. Bauer. Compilation of functional programming languages using GCC — Tail calls. Master's thesis, Technische Universität München, 2003.

[BD77] R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, 1977.

[BE00] R. Bloem and J. Engelfriet. 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, 2000.

[BH89] B. Bjerner and S. Holmström. A compositional approach to time analysis of first order lazy functional programs. In *4th International Conference on Functional Programming Languages and Computer Architecture — FPCA'89, Proceedings*, pages 157–165. ACM Press, 1989.

[Bir84] R.S. Bird. Using circular programs to eliminate multiple traversals of data. *Acta Informatica*, 21:239–250, 1984.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[Büc62] J.R. Büchi. On a decision method in restricted second-order arithmetic. In *1960 International Congress for Logic, Methodology and Philosophy of Science, Proceedings*, pages 1–11. Stanford University Press, 1962.

[BW88] R.S. Bird and P. Wadler. *Introduction to Functional Programming*. Prentice Hall, 1988.

[CDPR98] L. Correnson, E. Duris, D. Parigot, and G. Roussel. Symbolic composition. Technical Report 3348, Unité de recherche INRIA Rocquencourt, 1998.

[CDPR99] L. Correnson, E. Duris, D. Parigot, and G. Roussel. Declarative program transformation: A deforestation case-study. In *International Conference on Principles and Practice of Declarative Programming — PPDP'99, Proceedings*, volume 1702 of *LNCS*, pages 360–377. Springer-Verlag, 1999.

[CF82] B. Courcelle and P. Franchi-Zannettacci. Attribute grammars and recursive program schemes. *Theoretical Computer Science*, 17:163–191, 235–257, 1982.

[CGK99] W.N. Chin, A.H. Goh, and S.C. Khoo. Effective optimisation of multiple traversals in lazy languages. In *Workshop on Partial Evaluation and Semantics-Based Program Manipulation — PEPM'99, Proceedings,* Technical Report BRICS-NS-99-1, pages 119–130. University of Aarhus, 1999.

[Chi94] W.N. Chin. Safe fusion of functional expressions II: Further improvements. *Journal of Functional Programming*, 4(4):515–555, 1994.

[Chi99] O. Chitil. Type inference builds a short cut to deforestation. In *4th International Conference on Functional Programming — ICFP'99, Proceedings*, volume 34(9) of *SIGPLAN Notices*, pages 249–260. ACM Press, 1999.

[DE98] F. Drewes and J. Engelfriet. Decidability of the finiteness of ranges of tree transductions. *Information and Computation*, 145:1–50, 1998.

[DJ90] N. Dershowitz and J.P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier Science Publishers B.V., 1990.

[DJ04] N.A. Danielsson and P. Jansson. Chasing bottoms — A case study in program verification in the presence of partial and infinite values. In *7th International Conference on Mathematics of Program Construction — MPC'04, Proceedings*, volume 3125 of *LNCS*, pages 85–109. Springer-Verlag, 2004.

[EM99] J. Engelfriet and S. Maneth. Macro tree transducers, attribute grammars, and MSO definable tree translations. *Information and Computation*, 154(1):34–91, 1999.

[EM03a] J. Engelfriet and S. Maneth. A comparison of pebble tree transducers with macro tree transducers. *Acta Informatica*, 39(9):613–698, 2003.

[EM03b] J. Engelfriet and S. Maneth. Macro tree translations of linear size increase are MSO definable. *SIAM Journal on Computing*, 32(4):950–1006, 2003.

[Eng75] J. Engelfriet. Bottom-up and top-down tree transformations — A comparison. *Mathematical Systems Theory*, 9(3):198–231, 1975.

[Eng80] J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R.V. Book, editor, *Formal language theory; perspectives and open problems*, pages 241–286. Academic Press, 1980.

[Eng81] J. Engelfriet. Tree transducers and syntax directed semantics. Technical Report Memorandum 363, Technische Hogeschool Twente, 1981.

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31(1):71–146, 1985.

[EV88] J. Engelfriet and H. Vogler. High level tree transducers and iterated pushdown tree transducers. *Acta Informatica*, 26(1–2):131–192, 1988.

[EV91] J. Engelfriet and H. Vogler. Modular tree transducers. *Theoretical Computer Science*, 78(2):267–304, 1991.

[FFFK01] M. Felleisen, R.B. Findler, M. Flatt, and S. Krishnamurthi. *How to Design Programs — An Introduction to Programming and Computing*. MIT Press, 2001.

[Fra82] P. Franchi-Zannettacci. *Attributs sémantiques et schémas de programmes*. PhD thesis, Université de Bordeaux I, 1982.

[Fül81] Z. Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.

[FV98] Z. Fülöp and H. Vogler. *Syntax-Directed Semantics — Formal Models Based on Tree Transducers*. Springer-Verlag, 1998.

[Gan83] H. Ganzinger. Increasing modularity and language-independency in automatically generated compilers. *Science of Computer Programming*, 3(3):223–278, 1983.

[GG84] H. Ganzinger and R. Giegerich. Attribute coupled grammars. In *Symposium on Compiler Construction, Proceedings*, pages 157–170. ACM Press, 1984.

[Gie88] R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Informatica*, 25(4):355–423, 1988.

[Gie00] J. Giesl. Context-moving transformations for function verification. In *9th International Workshop on Logic-Based Program Synthesis and Transformation — LOPSTR'99, Selected Papers*, volume 1817 of *LNCS*, pages 293–312. Springer-Verlag, 2000.

[Gil96] A. Gill. *Cheap Deforestation for Non-strict Functional Languages*. PhD thesis, University of Glasgow, 1996.

[GKV03] J. Giesl, A. Kühnemann, and J. Voigtländer. Deaccumulation — Improving provability. In *8th Asian Computing Science Conference — ASIAN'03, Proceedings*, volume 2896 of *LNCS*, pages 146–160. Springer-Verlag, 2003.

[GLP93] A. Gill, J. Launchbury, and S.L. Peyton Jones. A short cut to deforestation. In *6th Conference on Functional Programming Languages and Computer Architecture — FPCA'93, Proceedings*, pages 223–232. ACM Press, 1993.

[Gro00] J.H.G. van Groningen. Optimising recursive functions yielding multiple results in tuples in a lazy functional language. In *11th International Workshop on Implementation of Functional Languages — IFL'99, Selected Papers*, volume 1868 of *LNCS*, pages 59–76. Springer-Verlag, 2000.

[GS01] J. Gustavsson and D. Sands. Possibilities and limitations of call-by-need space improvement. In *6th International Conference on Functional Programming — ICFP'01, Proceedings*, volume 36(10) of *SIGPLAN Notices*, pages 265–276. ACM Press, 2001.

[HJ92] G.W. Hamilton and S.B. Jones. Extending deforestation for first order functional programs. In *1991 Glasgow Workshop on Functional Programming, Proceedings*, Series of Workshops in Computing, pages 134–145. Springer-Verlag, 1992.

[Höf99] M. Höff. Vergleich von Verfahren zur Elimination von Zwischenergebnissen bei funktionalen Programmen. Master's thesis, Technische Universität Dresden, 1999.

[Hug86] R.J.M. Hughes. Strictness detection in non-flat domains. In *Workshop on Programs as Data Objects, Proceedings*, volume 217 of *LNCS*, pages 112–135. Springer-Verlag, 1986.

[Hug89] R.J.M. Hughes. Why functional programming matters. *Computer Journal*, 32(2):98–107, 1989.

[Joh87] T. Johnsson. Attribute grammars as a functional programming paradigm. In *3rd Conference on Functional Programming Languages and Computer Architecture — FPCA'87, Proceedings*, volume 274 of *LNCS*, pages 154–173. Springer-Verlag, 1987.

[Joh02] P. Johann. A generalization of short-cut fusion and its correctness proof. *Higher-Order and Symbolic Computation*, 15(4):273–300, 2002.

[Jür04] C. Jürgensen. *Categorical semantics and composition of tree transducers*. PhD thesis, Technische Universität Dresden, 2004.

[JV04a] P. Johann and J. Voigtländer. Free theorems in the presence of *seq*. In *31st Symposium on Principles of Programming Languages — POPL'04, Proceedings*, volume 39(1) of *SIGPLAN Notices*, pages 99–110. ACM Press, 2004.

[JV04b] P. Johann and J. Voigtländer. The impact of *seq* on free theorems-based program transformations. Submitted to *Fundamenta Informaticae*, 2004.

[JV04c] C. Jürgensen and H. Vogler. Syntactic composition of top-down tree transducers is short cut fusion. *Mathematical Structures in Computer Science*, 14(2):215–282, 2004.

[KGF02a] K. Kakehi, R. Glück, and Y. Futamura. An extension of shortcut deforestation for accumulative list folding. *IEICE Transactions on Information and Systems*, E85-D(9):1372–1383, 2002.

[KGF02b] K. Kakehi, R. Glück, and Y. Futamura. On deforesting parameters of accumulating maps. In *11th International Workshop on Logic Based Program Synthesis and Transformation — LOPSTR'01, Selected Papers*, volume 2372 of *LNCS*, pages 46–56. Springer-Verlag, 2002.

[KGK01] A. Kühnemann, R. Glück, and K. Kakehi. Relating accumulative and non-accumulative functional programs. In *12th International Conference on Rewriting Techniques and Applications — RTA'01, Proceedings*, volume 2051 of *LNCS*, pages 154–168. Springer-Verlag, 2001.

[Knu68] D.E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968. Corrections *Ibid.*, 5(1):95–96, 1971.

[KS87] M.F. Kuiper and S.D. Swierstra. Using attribute grammars to derive efficient functional programs. In *Computing Science in the Netherlands, Proceedings*. SION, 1987.

[Küh96] A. Kühnemann. A pumping lemma for output languages of macro tree transducers. In *21st International Colloquium on Trees in Algebra and Programming — CAAP'96, Proceedings*, volume 1059 of *LNCS*, pages 44–58. Springer-Verlag, 1996.

[Küh97] A. Kühnemann. *Berechnungsstärken von Teilklassen primitiv-rekursiver Programmschemata*. PhD thesis, Technische Universität Dresden, 1997.

[Küh98] A. Kühnemann. Benefits of tree transducers for optimizing functional programs. In *18th Conference on Foundations of Software Technology and Theoretical Computer Science — FSTTCS'98, Proceedings*, volume 1530 of *LNCS*, pages 146–157. Springer-Verlag, 1998.

[Küh99] A. Kühnemann. Comparison of deforestation techniques for functional programs and for tree transducers. In *4th International Symposium on Functional and Logic Programming — FLOPS'99, Proceedings*, volume 1722 of *LNCS*, pages 114–130. Springer-Verlag, 1999.

[KV94] A. Kühnemann and H. Vogler. Synthesized and inherited functions — A new computational model for syntax-directed semantics. *Acta Informatica*, 31(5):431–477, 1994.

[KV01] A. Kühnemann and J. Voigtländer. Tree transducer composition as deforestation method for functional programs. Technical Report TUD-FI01-07, Technische Universität Dresden, 2001.

[LG01] Y.A. Liu and G. Gómez. Automatic accurate cost-bound analysis for high-level languages. *IEEE Transactions on Computers*, 50(12):1295–1309, 2001.

[Mal01] A. Maletti. Efficiency analysis for the elimination of intermediate results in functional programs by compositions of attributed tree transducers. Bachelor's thesis, Technische Universität Dresden, 2001.

[Mal02] A. Maletti. Direct construction and efficiency analysis for the accumulation technique for 2-modular tree transducers. Master's thesis, Technische Universität Dresden, 2002.

[Man02] S. Maneth. The complexity of compositions of deterministic tree transducers. In *22nd Conference on Foundations of Software Technology and Theoretical Computer Science — FSTTCS'02, Proceedings*, volume 2556 of *LNCS*, pages 265–276. Springer Verlag, 2002.

[Man03] S. Maneth. The macro tree transducer hierarchy collapses for functions of linear size increase. In *23rd Conference on Foundations of Software Technology and Theoretical Computer Science — FSTTCS'03, Proceedings*, volume 2914 of *LNCS*, pages 326–337. Springer-Verlag, 2003.

[MBPS04] S. Maneth, A. Berlea, T. Perst, and H. Seidl. XML type checking with macro tree transducers. Technical Report TUM-I0407, Technische Universität München, 2004.

[MS99] A.K. Moran and D. Sands. Improvement in a lazy context: An operational theory for call-by-need. In *26th Symposium on Principles of Programming Languages — POPL'99, Proceedings*, pages 43–56. ACM Press, 1999.

[MW93] S. Marlow and P. Wadler. Deforestation for higher-order functions. In *1992 Glasgow Workshop on Functional Programming, Proceedings*, Series of Workshops in Computing, pages 154–165. Springer-Verlag, 1993.

[Nis02] S. Nishimura. Deforesting in accumulating parameters via type-directed transformations. In *3rd Asian Workshop on Programming Languages and Systems — APLAS'02, Informal Proceedings*, pages 145–159, 2002.

[Nis03] S. Nishimura. Correctness of a higher-order removal transformation through a relational reasoning. In *1st Asian Symposium on Programming Languages and Systems — APLAS'03, Proceedings*, volume 2895 of *LNCS*, pages 358–375. Springer-Verlag, 2003.

[Nis04] S. Nishimura. Fusion with stacks and accumulating parameters. In *Symposium on Partial Evaluation and Program Manipulation — PEPM'04, Proceedings*, pages 101–112. ACM Press, 2004.

[Pet77] A. Pettorossi. Transformation of programs and use of tupling strategy. In *Informatica, Proceedings*, pages 1–6, 1977.

[Pey03] S.L. Peyton Jones, editor. *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press, 2003.

[Pit00] A.M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10(3):321–359, 2000.

[PS04] T. Perst and H. Seidl. Macro forest transducers. *Information Processing Letters*, 89(3):141–149, 2004.

[Reu03] S. Reuther. Implementing tree transducer composition for the Glasgow Haskell Compiler. Master's thesis, Technische Universität Dresden, 2003.

[Rey83] J.C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing, Proceedings*, pages 513–523. Elsevier Science Publishers B.V., 1983.

[Ros89] M. Rosendahl. Automatic complexity analysis. In *4th International Conference on Functional Programming Languages and Computer Architecture — FPCA'89, Proceedings*, pages 144–156. ACM Press, 1989.

[Rou70] W.C. Rounds. Mappings and grammars on trees. *Mathematical Systems Theory*, 4(3):257–287, 1970.

[San90] D. Sands. Complexity analysis for a lazy higher-order language. In *3rd European Symposium on Programming — ESOP'90, Proceedings*, volume 432 of *LNCS*, pages 361–376. Springer-Verlag, 1990.

[San95] D. Sands. A naïve time analysis and its theory of cost equivalence. *Journal of Logic and Computation*, 5(4):495–541, 1995.

[San96a] D. Sands. Proving the correctness of recursion-based automatic program transformations. *Theoretical Computer Science*, 167:193–233, 1996.

[San96b] D. Sands. Total correctness by local improvement in the transformation of functional programs. *ACM Transactions on Programming Languages and Systems*, 18(2):175–234, 1996.

[Sch86] D.A. Schmidt. *Denotational Semantics: A Methodology for Language Development*. Allyn and Bacon, 1986.

[Spa93] J. Sparud. Fixing some space leaks without a garbage collector. In *6th Conference on Functional Programming Languages and Computer Architecture — FPCA'93, Proceedings*, pages 117–122. ACM Press, 1993.

[Sve02] J. Svenningsson. Shortcut fusion for accumulating parameters & zip-like functions. In *7th International Conference on Functional Programming — ICFP'02, Proceedings*, volume 37(9) of *SIGPLAN Notices*, pages 124–132. ACM Press, 2002.

[Tha70] J.W. Thatcher. Generalized^2 sequential machine maps. *Journal of Computer and System Sciences*, 4(4):339–367, 1970.

[VK04a] J. Voigtländer and A. Kühnemann. Composition of functions with accumulating parameters. *Journal of Functional Programming*, 14(3):317–363, 2004.

[VK04b] J. Voigtländer and A. Kühnemann. Proof appendix of [VK04a], 2004. Available from `http://wwwtcs.inf.tu-dresden.de/~voigt/JFP-appendix.pdf`.

[Vog87] H. Vogler. Basic tree transducers. *Journal of Computer and System Sciences*, 34(1):87–128, 1987.

[Vog91] H. Vogler. Functional description of the contextual analysis in block-structured programming languages: A case study of tree transducers. *Science of Computer Programming*, 16(3):251–275, 1991.

[Voi01] J. Voigtländer. Composition of restricted macro tree transducers. Master's thesis, Technische Universität Dresden, 2001.

[Voi02a] J. Voigtländer. Concatenate, reverse and map vanish for free. In *7th International Conference on Functional Programming — ICFP'02, Proceedings*, volume 37(9) of *SIGPLAN Notices*, pages 14–25. ACM Press, 2002.

[Voi02b] J. Voigtländer. Conditions for efficiency improvement by tree transducer composition. In *13th International Conference on Rewriting Techniques and Applications — RTA'02, Proceedings*, volume 2378 of *LNCS*, pages 222–236. Springer-Verlag, 2002.

[Voi02c] J. Voigtländer. Using circular programs to deforest in accumulating parameters. In *Asian Symposium on Partial Evaluation and Semantics-Based Program Manipulation — ASIA-PEPM'02, Proceedings*, pages 126–137. ACM Press, 2002.

[Voi04a] J. Voigtländer. Formal efficiency analysis for tree transducer composition. Technical Report TUD-FI04-08, Technische Universität Dresden, 2004. Abridged version submitted to *Theory of Computing Systems*.

[Voi04b] J. Voigtländer. Using circular programs to deforest in accumulating parameters. *Higher-Order and Symbolic Computation*, 17(1–2):129–163, 2004.

[Wad71] C.P. Wadsworth. *Semantics and Pragmatics of the Lambda Calculus*. PhD thesis, Oxford University, 1971.

[Wad87] P. Wadler. Fixing some space leaks with a garbage collector. *Software — Practice and Experience*, 17(9):595–608, 1987.

[Wad88] P. Wadler. Strictness analysis aids time analysis. In *15th Symposium on Principles of Programming Languages — POPL'88, Proceedings*, pages 119–132. ACM Press, 1988.

[Wad89a] P. Wadler. The concatenate vanishes. Note, University of Glasgow, 1987 (revised, 1989).

[Wad89b] P. Wadler. Theorems for free! In *4th International Conference on Functional Programming Languages and Computer Architecture — FPCA'89, Proceedings*, pages 347–359. ACM Press, 1989.

[Wad90] P. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, 73(2):231–248, 1990.

[Wad92] P. Wadler. The essence of functional programming (invited talk). In *19th Symposium on Principles of Programming Languages — POPL'92, Proceedings*, pages 1–14. ACM Press, 1992.

[WH87] P. Wadler and R.J.M. Hughes. Projections for strictness analysis. In *3rd Conference on Functional Programming Languages and Computer Architecture — FPCA'87, Proceedings*, volume 274 of *LNCS*, pages 385–407. Springer-Verlag, 1987.

Proof appendix

Proof of Lemma 2.8
We prove the following two statements:

(a) for every $t \in T_\Sigma$:
For every $f \in F^{(r'+1)}$, $r \in \mathbb{N}$, $k \in [r]$, and $\theta_1, \ldots, \theta_{r'} \in T_{F \cup \Sigma \cup \Delta}(Y_r)$:
$|nf(\Rightarrow_R, f\ t\ \theta_1 \cdots \theta_{r'})|_{y_k} = \sum_{k' \in [r']} |nf(\Rightarrow_R, f\ t\ y_1 \cdots y_{r'})|_{y_{k'}} * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}$.

(b) for every $p \in \mathbb{N}$ and $t_1, \ldots, t_p \in T_\Sigma$:
For every $r', r \in \mathbb{N}$, $k \in [r]$, $\theta_1, \ldots, \theta_{r'} \in T_{F \cup \Sigma \cup \Delta}(Y_r)$, $\phi \in RHS(F, \Delta, U_p, Y_{r'})$:
$|nf(\Rightarrow_R, \phi[u_1, \ldots, u_p, y_1, \ldots, y_{r'} \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_{r'}])|_{y_k}$
$= \sum_{k' \in [r']} |nf(\Rightarrow_R, \phi[u_i \leftarrow t_i \mid i \in [p]])|_{y_{k'}} * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}$.

by simultaneous induction. Lemma 2.8 coincides with statement (a).

(a) $\Leftarrow$ (b) : Assume $t = \sigma\ t_1 \cdots t_p$ for $\sigma \in \Sigma^{(p)}$ and $t_1, \ldots, t_p \in T_\Sigma$. We calculate:

$|nf(\Rightarrow_R, f\ (\sigma\ t_1 \cdots t_p)\ \theta_1 \cdots \theta_{r'})|_{y_k}$
$=$ (by $\Rightarrow_R$)
$|nf(\Rightarrow_R, rhs_{M,f,\sigma}[u_1, \ldots, u_p, y_1, \ldots, y_{r'} \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_{r'}])|_{y_k}$
$=$ (by induction hypothesis (b) with $\phi = rhs_{M,f,\sigma}$)
$\sum_{k' \in [r']} |nf(\Rightarrow_R, rhs_{M,f,\sigma}[u_i \leftarrow t_i \mid i \in [p]])|_{y_{k'}} * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}$
$=$ (by $\Rightarrow_R$)
$\sum_{k' \in [r']} |nf(\Rightarrow_R, f\ (\sigma\ t_1 \cdots t_p)\ y_1 \cdots y_{r'})|_{y_{k'}} * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}$

(b) $\Leftarrow$ (a) : for fixed $r', r \in \mathbb{N}$, $k \in [r]$, and $\theta_1, \ldots, \theta_{r'} \in T_{F \cup \Sigma \cup \Delta}(Y_r)$ by structural induction on $\phi \in RHS(F, \Delta, U_p, Y_{r'})$:

$\phi = y_{k''} \in Y_{r'}$:

$|nf(\Rightarrow_R, y_{k''}[u_1, \ldots, u_p, y_1, \ldots, y_{r'} \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_{r'}])|_{y_k}$
$=$ (by substitution)
$|nf(\Rightarrow_R, \theta_{k''})|_{y_k}$
$=$ (by $|y_{k''}|_{y_{k''}} = 1$ and $|y_{k''}|_{y_{k'}} = 0$ for $k' \neq k''$)
$\sum_{k' \in [r']} |y_{k''}|_{y_{k'}} * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}$

$= \quad$ (by substitution and normal form)

$\sum_{k' \in [r']} |nf(\Rightarrow_R, y_{k''}[u_i \leftarrow t_i \mid i \in [p]])|_{y_{k'}} * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}$

$\underline{\phi = \delta\ \phi_1 \cdots \phi_n}$ for some $\delta \in \Delta^{(n)}$ and $\phi_1, \ldots, \phi_n \in RHS(F, \Delta, U_p, Y_{r'})$:

$|nf(\Rightarrow_R, (\delta\ \phi_1 \cdots \phi_n)[u_1, \ldots, u_p, y_1, \ldots, y_{r'} \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_{r'}])|_{y_k}$

$= \quad$ (by substitution, normal form, and definition of $|\,.\,|_{y_k}$)

$\sum_{b \in [n]} |nf(\Rightarrow_R, \phi_b[u_1, \ldots, u_p, y_1, \ldots, y_{r'} \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_{r'}])|_{y_k}$

$= \quad$ (by the induction hypotheses for the $\phi_1, \ldots, \phi_n$)

$\sum_{b \in [n]} \sum_{k' \in [r']} |nf(\Rightarrow_R, \phi_b[u_i \leftarrow t_i \mid i \in [p]])|_{y_{k'}} * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}$

$= \quad$ (by sum manipulation)

$\sum_{k' \in [r']} \Big(\sum_{b \in [n]} |nf(\Rightarrow_R, \phi_b[u_i \leftarrow t_i \mid i \in [p]])|_{y_{k'}} \Big) * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}$

$= \quad$ (by substitution, normal form, and definition of $|\,.\,|_{y_{k'}}$)

$\sum_{k' \in [r']} |nf(\Rightarrow_R, (\delta\ \phi_1 \cdots \phi_n)[u_i \leftarrow t_i \mid i \in [p]])|_{y_{k'}} * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}$

$\underline{\phi = f\ u_{i'}\ \phi_1 \cdots \phi_{r''}}$ for some $f \in F^{(r''+1)}$, $u_{i'} \in U_p$, $\phi_1, \ldots, \phi_{r''} \in RHS(F, \Delta, U_p, Y_{r'})$:

$|nf(\Rightarrow_R, (f\ u_{i'}\ \phi_1 \cdots \phi_{r''})[u_1, \ldots, u_p, y_1, \ldots, y_{r'} \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_{r'}])|_{y_k}$

$= \quad$ (by substitution and induction hypothesis (a) for $t_{i'}$)

$\sum_{k'' \in [r'']} |nf(\Rightarrow_R, f\ t_{i'}\ y_1 \cdots y_{r''})|_{y_{k''}} * |nf(\Rightarrow_R, \phi_{k''}[u_i \leftarrow t_i \mid i \in [p],\ y_{k'} \leftarrow \theta_{k'} \mid k' \in [r']])|_{y_k}$

$= \quad$ (by the induction hypotheses for the $\phi_1, \ldots, \phi_{r''}$)

$\sum_{k'' \in [r'']} |nf(\Rightarrow_R, f\ t_{i'}\ y_1 \cdots y_{r''})|_{y_{k''}} * \sum_{k' \in [r']} |nf(\Rightarrow_R, \phi_{k''}[u_i \leftarrow t_i \mid i \in [p]])|_{y_{k'}} * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}$

$= \quad$ (by sum manipulation)

$\sum_{k' \in [r']} \Big(\sum_{k'' \in [r'']} |nf(\Rightarrow_R, f\ t_{i'}\ y_1 \cdots y_{r''})|_{y_{k''}} * |nf(\Rightarrow_R, \phi_{k''}[u_i \leftarrow t_i \mid i \in [p]])|_{y_{k'}} \Big) * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}$

$= \quad$ (by substitution and induction hypothesis (a) for $t_{i'}$)

$\sum_{k' \in [r']} |nf(\Rightarrow_R, (f\ u_{i'}\ \phi_1 \cdots \phi_{r''})[u_i \leftarrow t_i \mid i \in [p]])|_{y_{k'}} * |nf(\Rightarrow_R, \theta_{k'})|_{y_k}$ $\quad \square$

Proof of Lemma 2.9

For a context-linear mtt M we prove the following two statements:

(a) $\underline{\text{for every } t \in T_\Sigma\text{:}}$
For every $f \in F^{(r+1)}$ and $k \in [r]$: $|nf(\Rightarrow_R, f\ t\ y_1 \cdots y_r)|_{y_k} \leq 1$.

(b) $\underline{\text{for every } p \in \mathbb{N} \text{ and } t_1, \ldots, t_p \in T_\Sigma\text{:}}$
For every $r \in \mathbb{N}$, $k \in [r]$, and $\phi \in RHS(F, \Delta, U_p, Y_r)$:
$|nf(\Rightarrow_R, \phi[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} \leq |\phi|_{y_k}$.

by simultaneous induction. The first statement of Lemma 2.9 then coincides with statement (a). The second statement of Lemma 2.9, for context-nondeleting M, is obtained by replacing all occurrences of $\leq$ and "context-linear" by $\geq$ and "context-nondeleting", respectively, in the above statements and the following proof.

(a) $\Leftarrow$ (b) : Assume $t = \sigma\ t_1 \cdots t_p$ for $\sigma \in \Sigma^{(p)}$ and $t_1, \ldots, t_p \in T_\Sigma$. We calculate:

$$
\begin{aligned}
&|nf(\Rightarrow_R, f\ (\sigma\ t_1 \cdots t_p)\ y_1 \cdots y_r)|_{y_k} \\
&\quad = \quad \text{(by } \Rightarrow_R\text{)} \\
&|nf(\Rightarrow_R, rhs_{M,f,\sigma}[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} \\
&\quad \leq \quad \text{(by induction hypothesis (b) with } \phi = rhs_{M,f,\sigma}\text{)} \\
&|rhs_{M,f,\sigma}|_{y_k} \\
&\quad \leq \quad \text{(since } M \text{ is context-linear)} \\
&1
\end{aligned}
$$

(b) $\Leftarrow$ (a) : for fixed $r \in \mathbb{N}$, $k \in [r]$ by structural induction on $\phi \in RHS(F, \Delta, U_p, Y_r)$:

$\underline{\phi = y_{k'} \in Y_r}$:

$$
\begin{aligned}
&|nf(\Rightarrow_R, y_{k'}[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} \\
&\quad = \quad \text{(by substitution and normal form)} \\
&|y_{k'}|_{y_k}
\end{aligned}
$$

$\underline{\phi = \delta\ \phi_1 \cdots \phi_n}$ for some $\delta \in \Delta^{(n)}$ and $\phi_1, \ldots, \phi_n \in RHS(F, \Delta, U_p, Y_r)$:

$$
\begin{aligned}
&|nf(\Rightarrow_R, (\delta\ \phi_1 \cdots \phi_n)[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} \\
&\quad = \quad \text{(by substitution, normal form, and definition of } |\,\cdot\,|_{y_k}\text{)} \\
&\sum_{b \in [n]} |nf(\Rightarrow_R, \phi_b[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} \\
&\quad \leq \quad \text{(by the induction hypotheses for the } \phi_1, \ldots, \phi_n\text{)} \\
&\sum_{b \in [n]} |\phi_b|_{y_k} \\
&\quad = \quad \text{(by definition of } |\,\cdot\,|_{y_k}\text{)} \\
&|\delta\ \phi_1 \cdots \phi_n|_{y_k}
\end{aligned}
$$

$\underline{\phi = f\ u_{i'}\ \phi_1 \cdots \phi_{r'}}$ for some $f \in F^{(r'+1)}$, $u_{i'} \in U_p$, $\phi_1, \ldots, \phi_{r'} \in RHS(F, \Delta, U_p, Y_r)$:

$$
\begin{aligned}
&|nf(\Rightarrow_R, (f\ u_{i'}\ \phi_1 \cdots \phi_{r'})[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} \\
&\quad = \quad \text{(by substitution and Lemma 2.8)} \\
&\sum_{k' \in [r']} |nf(\Rightarrow_R, f\ t_{i'}\ y_1 \cdots y_{r'})|_{y_{k'}} * |nf(\Rightarrow_R, \phi_{k'}[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} \\
&\quad \leq \quad \text{(by induction hypothesis (a) for } t_{i'}\text{,} \\
&\qquad\quad \text{using that the factors } |nf(\Rightarrow_R, \phi_{k'}[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} \geq 0\text{)} \\
&\sum_{k' \in [r']} |nf(\Rightarrow_R, \phi_{k'}[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} \\
&\quad \leq \quad \text{(by the induction hypotheses for the } \phi_1, \ldots, \phi_{r'}\text{)} \\
&\sum_{k' \in [r']} |\phi_{k'}|_{y_k}
\end{aligned}
$$

$= \quad$ (by definition of $|\,\cdot\,|_{y_k}$)
$|f\ u_{i'}\ \phi_1 \cdots \phi_{r'}|_{y_k} \quad \square$

Proof of Lemma 4.10
For fixed $p, r \in \mathbb{N}$ we prove the following two statements:

(a) for every $\phi \in RHS(F, \Delta, U_p, Y_r)$:
For every $g \in G^{(s+1)}$ and $\eta_1, \ldots, \eta_s, \eta_1^\star, \ldots, \eta_s^\star \in T_{F \cup G \cup H \cup \Delta \cup \Omega \cup \{\diamond, \star, \circ, nil\}}(U_p \cup Y_r \cup Z \cup Z_G)$ with $nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, \eta_l^\star) = nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, \eta_l)$ for every $l \in [s]$:
$nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, g\ \phi[y_k \leftarrow \star\ y_k \mid k \in [r]]\ \eta_1^\star \cdots \eta_s^\star)$
$= nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, g\ \phi\ \eta_1 \cdots \eta_s)$.

(b) for every $r' \in \mathbb{N}$ and $\phi_1, \ldots, \phi_{r'} \in RHS(F, \Delta, U_p, Y_r)$:
For every $f' \in F^{(r'+1)}$, $u_i \in U_p$, and
$\eta_{g_1,1}, \ldots, \eta_{g_\mu,s_\mu}, \eta^\star_{g_1,1}, \ldots, \eta^\star_{g_\mu,s_\mu} \in T_{F \cup G \cup H \cup \Delta \cup \Omega \cup \{\diamond, \star, \circ, nil\}}(U_p \cup Y_r \cup Z \cup Z_G)$ with $nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, \eta^\star_{g',l'}) = nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, \eta_{g',l'})$ for every $g' \in G^{(s'+1)}$ and $l' \in [s']$, for every $\mathcal{C} \subseteq [r'] \times G$, $k' \in [r']$, and $g'' \in G$:

$$\begin{aligned} & nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, nest_{f'}(k', g'', \mathcal{C})[u \quad \leftarrow u_i, \\ & \qquad\qquad\qquad\qquad\qquad\qquad y'_b \quad \leftarrow \phi_b[y_k \leftarrow \star\ y_k \mid k \in [r]] \mid b \in [r'], \\ & \qquad\qquad\qquad\qquad\qquad\qquad z_{g',l'} \leftarrow \eta^\star_{g',l'} \mid g' \in G^{(s'+1)}, l' \in [s']]) \\ = & \ nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, nest_{f'}(k', g'', \mathcal{C})[u \quad \leftarrow u_i, \\ & \qquad\qquad\qquad\qquad\qquad\qquad y'_b \quad \leftarrow \phi_b \quad \mid b \in [r'], \\ & \qquad\qquad\qquad\qquad\qquad\qquad z_{g',l'} \leftarrow \eta_{g',l'} \mid g' \in G^{(s'+1)}, l' \in [s']]). \end{aligned}$$

by simultaneous induction. Lemma 4.10 then follows from statement (a) with $\phi = rhs_{M_1,f,\sigma}$ and $\eta_l = \eta_l^\star = z_l$ for every $l \in [s]$.

(a) $\Leftarrow$ (b) : by case analysis on $\phi \in RHS(F, \Delta, U_p, Y_r)$:

$\phi = y_{k'} \in Y_r$:

$nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, g\ y_{k'}[y_k \leftarrow \star\ y_k \mid k \in [r]]\ \eta_1^\star \cdots \eta_s^\star)$
$= \quad$ (by substitution)
$nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, g\ (\star\ y_{k'})\ \eta_1^\star \cdots \eta_s^\star)$
$= \quad$ (by $\Rightarrow_{R_2^{\diamond\star\to\circ}}$, using that $rhs_{M_2^{\diamond\star\to\circ},g,\star} = g\ v_1\ (\circ\ z_1)\ \cdots\ (\circ\ z_s)$ by Def. 4.9)
$nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, g\ y_{k'}\ (\circ\ \eta_1^\star)\ \cdots\ (\circ\ \eta_s^\star))$
$= \quad$ (by $\Rightarrow_{Pre}$ (cf. Construction 3.1) and normal form)
$y_{k',g}$
$= \quad$ (by $\Rightarrow_{Pre}$ (cf. Construction 3.1) and normal form)
$nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, g\ y_{k'}\ \eta_1 \cdots \eta_s)$

$\phi = \delta\ \phi_1 \cdots \phi_q$ for some $\delta \in \Delta^{(q)}$ and $\phi_1, \ldots, \phi_q \in RHS(F, \Delta, U_p, Y_r)$:

$nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, g\ (\delta\ \phi_1 \cdots \phi_q)[y_k \leftarrow \star\ y_k \mid k \in [r]]\ \eta_1^\star \cdots \eta_s^\star)$
$= \quad$ (by substitution and $\Rightarrow_{R_2^{\diamond\star\to\circ}}$,
using that $rhs_{M_2^{\diamond\star\to\circ},g,\delta} = rhs_{M_2,g,\delta}$ by Definition 4.9)

$$nf(\Rightarrow_{R_2^{\otimes\star\to\circ}\cup Pre\cup Pair}, rhs_{M_2,g,\delta}[v_j \leftarrow \phi_j[y_k \leftarrow \star\ y_k \mid k \in [r]] \mid j \in [q], \\ z_l \leftarrow \eta_l^\star \mid l \in [s]])$$

$=$ (see below)

$$nf(\Rightarrow_{R_2\cup Pre\cup Pair}, rhs_{M_2,g,\delta}[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \phi_1, \ldots, \phi_q, \eta_1, \ldots, \eta_s])$$

$=$ (by $\Rightarrow_{R_2}$)

$$nf(\Rightarrow_{R_2\cup Pre\cup Pair}, g\ (\delta\ \phi_1 \cdots \phi_q)\ \eta_1 \cdots \eta_s)$$

Above we used that for every $\psi \in RHS(G, \Omega, V_q, Z_s)$:

$$nf(\Rightarrow_{R_2^{\otimes\star\to\circ}\cup Pre\cup Pair}, \psi[v_j \leftarrow \phi_j[y_k \leftarrow \star\ y_k \mid k \in [r]] \mid j \in [q], \\ z_l \leftarrow \eta_l^\star \mid l \in [s]]) \\ = nf(\Rightarrow_{R_2\cup Pre\cup Pair}, \psi[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \phi_1, \ldots, \phi_q, \eta_1, \ldots, \eta_s]),$$

which can be proved by structural induction. The validity in the base case $\psi \in Z_s$ follows from the precondition that for every $l \in [s]$, $nf(\Rightarrow_{R_2^{\otimes\star\to\circ}\cup Pre\cup Pair}, \eta_l^\star) = nf(\Rightarrow_{R_2\cup Pre\cup Pair}, \eta_l)$. The one recursive case, $\psi = \omega\ \psi_1 \cdots \psi_n$, is straightforward and in the other one, $\psi = g'\ v_j\ \psi_1 \cdots \psi_{s'}$, the induction hypotheses for the $\psi_1, \ldots, \psi_{s'}$ establish the preconditions necessary to apply induction hypothesis (a) for ϕ_j, which suffices.

$\underline{\phi = f'\ u_i\ \phi_1 \cdots \phi_{r'}}$ for some $f' \in F^{(r'+1)}$, $u_i \in U_p$, $\phi_1, \ldots, \phi_{r'} \in RHS(F, \Delta, U_p, Y_r)$:

$$nf(\Rightarrow_{R_2^{\otimes\star\to\circ}\cup Pre\cup Pair}, g\ (f'\ u_i\ \phi_1 \cdots \phi_{r'})[y_k \leftarrow \star\ y_k \mid k \in [r]]\ \eta_1^\star \cdots \eta_s^\star)$$

$=$ (by substitution and $\Rightarrow_{Pair}$ (cf. Construction 3.1))

$$nf(\Rightarrow_{R_2^{\otimes\star\to\circ}\cup Pre\cup Pair}, \overline{f'g}\ u_i \\ nest_{f'}(1, g_1, \emptyset)[u \leftarrow u_i, \\ y'_b \leftarrow \phi_b[y_k \leftarrow \star\ y_k \mid k \in [r]] \mid b \in [r'], \\ z_{g,l} \leftarrow \eta_l^\star \mid l \in [s], \\ z_{g',l'} \leftarrow nil \mid g' \in G^{(s'+1)} \setminus \{g\}, l' \in [s']] \\ \cdots \\ nest_{f'}(r', g_\mu, \emptyset)[u \leftarrow u_i, \\ y'_b \leftarrow \phi_b[y_k \leftarrow \star\ y_k \mid k \in [r]] \mid b \in [r'], \\ z_{g,l} \leftarrow \eta_l^\star \mid l \in [s], \\ z_{g',l'} \leftarrow nil \mid g' \in G^{(s'+1)} \setminus \{g\}, l' \in [s']] \\ \eta_1^\star \cdots \eta_s^\star)$$

$=$ (see below)

$$nf(\Rightarrow_{R_2\cup Pre\cup Pair}, \overline{f'g}\ u_i\ nest_{f'}(1, g_1, \emptyset)[u \leftarrow u_i, \\ y'_b \leftarrow \phi_b \mid b \in [r'], \\ z_{g,l} \leftarrow \eta_l \mid l \in [s], \\ z_{g',l'} \leftarrow nil \mid g' \in G^{(s'+1)} \setminus \{g\}, l' \in [s']] \\ \cdots \\ nest_{f'}(r', g_\mu, \emptyset)[u \leftarrow u_i, \\ y'_b \leftarrow \phi_b \mid b \in [r'], \\ z_{g,l} \leftarrow \eta_l \mid l \in [s], \\ z_{g',l'} \leftarrow nil \mid g' \in G^{(s'+1)} \setminus \{g\}, l' \in [s']] \\ \eta_1 \cdots \eta_s)$$

$=$ (by $\Rightarrow_{Pair}$ (cf. Construction 3.1))
$nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, g\ (f'\ u_i\ \phi_1 \cdots \phi_{r'})\ \eta_1 \cdots \eta_s)$

Given the precondition that $nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, \eta_l^\star) = nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, \eta_l)$ for every $l \in [s]$, the above gap can be closed (by normal forms) if we establish for every $k' \in [r']$ and $g'' \in G$:

$$\begin{aligned} nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, nest_{f'}(k', g'', \emptyset)[u &\leftarrow u_i, \\ y'_b &\leftarrow \phi_b[y_k \leftarrow \star\ y_k \mid k \in [r]] \mid b \in [r'], \\ z_{g,l} &\leftarrow \eta_l^\star \mid l \in [s], \\ z_{g',l'} &\leftarrow nil \mid g' \in G^{(s'+1)} \setminus \{g\}, l' \in [s']]) \\ = nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, nest_{f'}(k', g'', \emptyset)[u &\leftarrow u_i, \\ y'_b &\leftarrow \phi_b \mid b \in [r'], \\ z_{g,l} &\leftarrow \eta_l \mid l \in [s], \\ z_{g',l'} &\leftarrow nil \mid g' \in G^{(s'+1)} \setminus \{g\}, l' \in [s']]). \end{aligned}$$

But this is an instance of induction hypothesis (b), given that $nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, nil) = nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, nil)$ and the preconditions on the $\eta_1, \ldots, \eta_s, \eta_1^\star, \ldots, \eta_s^\star$.

(b) $\Leftarrow$ (a) : for fixed $f' \in F^{(r'+1)}$, $u_i \in U_p$, and $\eta_{g_1,1}, \ldots, \eta_{g_\mu,s_\mu}, \eta^\star_{g_1,1}, \ldots, \eta^\star_{g_\mu,s_\mu} \in T_{F \cup G \cup H \cup \Delta \cup \Omega \cup \{\diamond,\star,\circ,nil\}}(U_p \cup Y_r \cup Z \cup Z_G)$ with $nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, \eta^\star_{g',l'}) = nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, \eta_{g',l'})$ for every $g' \in G^{(s'+1)}$ and $l' \in [s']$, by induction over the reversed subset-order on $\mathcal{P}([r'] \times G)$:

$\underline{\mathcal{C} = [r'] \times G}$: In this base case we have for every $k' \in [r']$ and $g'' \in G$ that $(k', g'') \in \mathcal{C}$ and thus $nest_{f'}(k', g'', \mathcal{C}) = nil$, which makes the desired equation trivially true.

$\underline{\mathcal{C} \subset [r'] \times G}$: For every $k' \in [r']$ and $g'' \in G$ with $(k', g'') \notin \mathcal{C}$—the other case is again trivial—we have (setting $s'' = rank_G(g'') - 1$):

$$\begin{aligned} nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, nest_{f'}(k', g'', \mathcal{C})[u &\leftarrow u_i, \\ y'_b &\leftarrow \phi_b[y_k \leftarrow \star\ y_k \mid k \in [r]] \mid b \in [r'], \\ z_{g',l'} &\leftarrow \eta^\star_{g',l'} \mid g' \in G^{(s'+1)}, l' \in [s']]) \end{aligned}$$

$=$ (by definition of $nest_{f'}$ (cf. Construction 3.1) and substitution)

$$\begin{aligned} &nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, \\ &\quad g''\ \phi_{k'}[y_k \leftarrow \star\ y_k \mid k \in [r]] \\ &\qquad (\overline{k'_{f'} 1_{g''}}\ u\ nest_{f'}(1, g_1, \mathcal{C} \cup \{(k', g'')\})\ \cdots\ nest_{f'}(r', g_\mu, \mathcal{C} \cup \{(k', g'')\}) \\ &\qquad\quad z_{g_1,1} \cdots z_{g_\mu,s_\mu})[u \leftarrow u_i, \\ &\qquad\qquad y'_b \leftarrow \phi_b[y_k \leftarrow \star\ y_k \mid k \in [r]] \mid b \in [r'], \\ &\qquad\qquad z_{g',l'} \leftarrow \eta^\star_{g',l'} \mid g' \in G^{(s'+1)}, l' \in [s']] \\ &\qquad \cdots \\ &\qquad (\overline{k'_{f'} s''_{g''}}\ u\ nest_{f'}(1, g_1, \mathcal{C} \cup \{(k', g'')\})\ \cdots\ nest_{f'}(r', g_\mu, \mathcal{C} \cup \{(k', g'')\}) \\ &\qquad\quad z_{g_1,1} \cdots z_{g_\mu,s_\mu})[u \leftarrow u_i, \\ &\qquad\qquad y'_b \leftarrow \phi_b[y_k \leftarrow \star\ y_k \mid k \in [r]] \mid b \in [r'], \\ &\qquad\qquad z_{g',l'} \leftarrow \eta^\star_{g',l'} \mid g' \in G^{(s'+1)}, l' \in [s']]) \end{aligned}$$

$$
\begin{aligned}
&= \quad \text{(see below)}\\
&nf(\Rightarrow_{R_2 \cup Pre \cup Pair},\\
&\quad g''\ \phi_{k'}\\
&\qquad (\overline{k'_{f'} 1_{g''}}\ u\ nest_{f'}(1, g_1, \mathcal{C} \cup \{(k', g'')\})\ \cdots\ nest_{f'}(r', g_\mu, \mathcal{C} \cup \{(k', g'')\})\\
&\qquad\qquad z_{g_1,1} \cdots z_{g_\mu,s_\mu})[u \quad \leftarrow u_i,\\
&\qquad\qquad\qquad\qquad y'_b \quad \leftarrow \phi_b \quad \mid b \in [r'],\\
&\qquad\qquad\qquad\qquad z_{g',l'} \leftarrow \eta_{g',l'} \mid g' \in G^{(s'+1)}, l' \in [s']]\\
&\qquad \cdots\\
&\qquad (\overline{k'_{f'} s''_{g''}}\ u\ nest_{f'}(1, g_1, \mathcal{C} \cup \{(k', g'')\})\ \cdots\ nest_{f'}(r', g_\mu, \mathcal{C} \cup \{(k', g'')\})\\
&\qquad\qquad z_{g_1,1} \cdots z_{g_\mu,s_\mu})[u \quad \leftarrow u_i,\\
&\qquad\qquad\qquad\qquad y'_b \quad \leftarrow \phi_b \quad \mid b \in [r'],\\
&\qquad\qquad\qquad\qquad z_{g',l'} \leftarrow \eta_{g',l'} \mid g' \in G^{(s'+1)}, l' \in [s']])\\
&= \quad \text{(by definition of } nest_{f'} \text{ and substitution)}\\
&nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, nest_{f'}(k', g'', \mathcal{C})[u \quad \leftarrow u_i,\\
&\qquad\qquad\qquad\qquad\qquad\qquad y'_b \quad \leftarrow \phi_b \quad \mid b \in [r'],\\
&\qquad\qquad\qquad\qquad\qquad\qquad z_{g',l'} \leftarrow \eta_{g',l'} \mid g' \in G^{(s'+1)}, l' \in [s']])
\end{aligned}
$$

The above gap can be closed by using induction hypothesis (a) for $\phi_{k'}$ if we establish for every $l \in [s'']$:

$$
\begin{aligned}
&nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair},\\
&\quad (\overline{k'_{f'} l_{g''}}\ u\ nest_{f'}(1, g_1, \mathcal{C} \cup \{(k', g'')\})\ \cdots\ nest_{f'}(r', g_\mu, \mathcal{C} \cup \{(k', g'')\})\\
&\qquad z_{g_1,1} \cdots z_{g_\mu,s_\mu})[u \quad \leftarrow u_i,\\
&\qquad\qquad\qquad y'_b \quad \leftarrow \phi_b[y_k \leftarrow \star\ y_k \mid k \in [r]] \mid b \in [r'],\\
&\qquad\qquad\qquad z_{g',l'} \leftarrow \eta^\star_{g',l'} \mid g' \in G^{(s'+1)}, l' \in [s']])\\
&= nf(\Rightarrow_{R_2 \cup Pre \cup Pair},\\
&\quad (\overline{k'_{f'} l_{g''}}\ u\ nest_{f'}(1, g_1, \mathcal{C} \cup \{(k', g'')\})\ \cdots\ nest_{f'}(r', g_\mu, \mathcal{C} \cup \{(k', g'')\})\\
&\qquad z_{g_1,1} \cdots z_{g_\mu,s_\mu})[u \quad \leftarrow u_i,\\
&\qquad\qquad\qquad y'_b \quad \leftarrow \phi_b \quad \mid b \in [r'],\\
&\qquad\qquad\qquad z_{g',l'} \leftarrow \eta_{g',l'} \mid g' \in G^{(s'+1)}, l' \in [s']]).
\end{aligned}
$$

But this follows directly (by substitution and normal forms) from the preconditions on the $\eta_{g_1,1}, \ldots, \eta_{g_\mu,s_\mu}, \eta^\star_{g_1,1}, \ldots, \eta^\star_{g_\mu,s_\mu}$ and applications of the induction hypothesis for $\mathcal{C} \cup \{(k', g'')\}$. $\square$

Proof of Lemma 4.11
For fixed $\sigma \in \Sigma^{(p)}$ and $f \in F^{(r+1)}$ with $r \in \mathbb{N}_+$ we prove by induction over the prefix-order of paths $\pi \in paths(rhs_{M_1,f,\sigma})$ with $lab(rhs_{M_1,f,\sigma}, \pi) \notin U_p$ that for every $g \in G^{(s+1)}$ and $l \in [s]$:

$$
\begin{aligned}
&nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, par_{M_2^{\diamond\star\to\circ}, \diamond\ rhs_{M_1,f,\sigma}[y_k \leftarrow \star\ y_k \mid k \in [r]]}(1\pi, g, l))\\
&= nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, par_{M_2, rhs_{M_1,f,\sigma}}(\pi, g, l)).
\end{aligned}
$$

Lemma 4.11 then follows by $Y_r \cap U_p = \emptyset$ and $rhs_{M_1^{\to\diamond\star}, f, \sigma} = \diamond\ rhs_{M_1,f,\sigma}[y_k \leftarrow \star\ y_k \mid k \in [r]]$ (cf. Definition 4.9). In the base case ε we have:

$$
\begin{aligned}
&nf(\Rightarrow_{R_2^{\diamond\star\to\circ}\cup Pre\cup Pair}, par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1,g,l))\\
&\quad = \quad \text{(by definition of } par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]} \text{ (cf. Construction 3.1),}\\
&\qquad\quad \text{using that } rhs_{M_2^{\diamond\star\to\circ},g,\diamond} = \circ\ (g\ v_1\ z_1 \cdots z_s) \text{ by Definition 4.9)}\\
&nf(\Rightarrow_{R_2^{\diamond\star\to\circ}\cup Pre\cup Pair}, par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(\varepsilon,g,l))\\
&\quad = \quad \text{(by definition of } par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]} \text{ (cf. Construction 3.1))}\\
&nf(\Rightarrow_{R_2^{\diamond\star\to\circ}\cup Pre\cup Pair}, z_{g,l})\\
&\quad = \quad \text{(by normal forms)}\\
&nf(\Rightarrow_{R_2\cup Pre\cup Pair}, z_{g,l})\\
&\quad = \quad \text{(by definition of } par_{M_2,rhs_{M_1,f,\sigma}} \text{ (cf. Construction 3.1))}\\
&nf(\Rightarrow_{R_2\cup Pre\cup Pair}, par_{M_2,rhs_{M_1,f,\sigma}}(\varepsilon,g,l))
\end{aligned}
$$

The inductive case $\pi j \in paths(rhs_{M_1,f,\sigma})$ with $j \in \mathbb{N}_+$ and $lab(rhs_{M_1,f,\sigma}, \pi j) \notin U_p$ is shown by case analysis on $lab(rhs_{M_1,f,\sigma}, \pi) = lab(\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k \in [r]], 1\pi) \in \Delta \cup F$:

$\underline{lab(rhs_{M_1,f,\sigma}, \pi) = \delta}$ for some $\delta \in \Delta^{(q)}$, where $j \in [q]$:

If there is no $(g\ v_j\ \cdots)$-call in the δ-rules of M_2 and hence also no such call in the δ-rules of $M_2^{\diamond\star\to\circ}$ (cf. Definition 4.9), then both sides of the equation are equal to *nil* by the definitions of $par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}$ and $par_{M_2,rhs_{M_1,f,\sigma}}$ (cf. Construction 3.1).

Otherwise, if the unique such call—in both the δ-rules of M_2 and of $M_2^{\diamond\star\to\circ}$—looks, with $g' \in G^{(s'+1)}$ and $\psi_1, \ldots, \psi_s \in RHS(G, \Omega, V_q, Z_{s'})$, as follows:

$$g'\ (\delta\ v_1 \cdots v_q)\ z_1 \cdots z_{s'} \to \cdots (g\ v_j\ \psi_1 \cdots \psi_s) \cdots ,$$

then:

$$
\begin{aligned}
&nf(\Rightarrow_{R_2^{\diamond\star\to\circ}\cup Pre\cup Pair}, par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1\pi j,g,l))\\
&\quad = \quad \text{(by definition of } par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]} \text{ (cf. Construction 3.1))}\\
&nf(\Rightarrow_{R_2^{\diamond\star\to\circ}\cup Pre\cup Pair}, \psi_l[v_{j'} \leftarrow sub(\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]], 1\pi j') \mid j' \in [q],\\
&\qquad\qquad\qquad\qquad z_{l'} \leftarrow par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1\pi, g', l') \mid l' \in [s']])\\
&\quad = \quad \text{(by definition and properties of } sub,\\
&\qquad\quad \text{taking into account that } lab(rhs_{M_1,f,\sigma}, \pi) \notin Y_r)\\
&nf(\Rightarrow_{R_2^{\diamond\star\to\circ}\cup Pre\cup Pair}, \psi_l[v_{j'} \leftarrow sub(rhs_{M_1,f,\sigma}, \pi j')[y_k \leftarrow \star\, y_k \mid k\in[r]] \mid j' \in [q],\\
&\qquad\qquad\qquad\qquad z_{l'} \leftarrow par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1\pi, g', l') \mid l' \in [s']])\\
&\quad = \quad \text{(see below)}\\
&nf(\Rightarrow_{R_2\cup Pre\cup Pair}, \psi_l[v_{j'} \leftarrow sub(rhs_{M_1,f,\sigma}, \pi j') \mid j' \in [q],\\
&\qquad\qquad\qquad\qquad z_{l'} \leftarrow par_{M_2,rhs_{M_1,f,\sigma}}(\pi, g', l') \mid l' \in [s']])\\
&\quad = \quad \text{(by definition of } par_{M_2,rhs_{M_1,f,\sigma}} \text{ (cf. Construction 3.1))}\\
&nf(\Rightarrow_{R_2\cup Pre\cup Pair}, par_{M_2,rhs_{M_1,f,\sigma}}(\pi j, g, l))
\end{aligned}
$$

Above we used that for every $\psi \in RHS(G, \Omega, V_q, Z_{s'})$:

$$\begin{aligned}
&nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, \psi[v_{j'} \leftarrow sub(rhs_{M_1,f,\sigma}, \pi j')[y_k \leftarrow \star\, y_k \mid k \in [r]] \quad \mid j' \in [q],\\
&\qquad\qquad z_{l'} \leftarrow par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1\pi, g', l') \mid l' \in [s']])\\
=\ &nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, \psi[v_{j'} \leftarrow sub(rhs_{M_1,f,\sigma}, \pi j') \quad \mid j' \in [q],\\
&\qquad\qquad z_{l'} \leftarrow par_{M_2, rhs_{M_1,f,\sigma}}(\pi, g', l') \mid l' \in [s']]),
\end{aligned}$$

which can be proved by structural induction. The validity in the base case $\psi \in Z_{s'}$ follows from the induction hypothesis for π. One recursive case, $\psi = \omega\ \psi'_1 \cdots \psi'_n$, is straightforward. The validity in the remaining case is proved as follows.

$\underline{\psi = g''\ v_d\ \psi'_1 \cdots \psi'_{s''}}$ for some $g'' \in G^{(s''+1)}$, $v_d \in V_q$, and $\psi'_1, \ldots, \psi'_{s''} \in RHS(G, \Omega, V_q, Z_{s'})$:

$$\begin{aligned}
&nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair},\\
&\quad (g''\ v_d\ \psi'_1 \cdots \psi'_{s''})[v_{j'} \leftarrow sub(rhs_{M_1,f,\sigma}, \pi j')[y_k \leftarrow \star\, y_k \mid k \in [r]] \quad \mid j' \in [q],\\
&\qquad\qquad z_{l'} \leftarrow par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1\pi, g', l') \mid l' \in [s']])\\
=\ &\text{(by substitution)}\\
&nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair},\\
&\quad g''\ sub(rhs_{M_1,f,\sigma}, \pi d)[y_k \leftarrow \star\, y_k \mid k \in [r]]\\
&\qquad \psi'_1[v_{j'} \leftarrow sub(rhs_{M_1,f,\sigma}, \pi j')[y_k \leftarrow \star\, y_k \mid k \in [r]] \quad \mid j' \in [q],\\
&\qquad\quad z_{l'} \leftarrow par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1\pi, g', l') \mid l' \in [s']]\\
&\qquad \cdots\\
&\qquad \psi'_{s''}[v_{j'} \leftarrow sub(rhs_{M_1,f,\sigma}, \pi j')[y_k \leftarrow \star\, y_k \mid k \in [r]] \quad \mid j' \in [q],\\
&\qquad\quad z_{l'} \leftarrow par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1\pi, g', l') \mid l' \in [s']])\\
=\ &\text{(by statement (a) from the proof of Lemma 4.10 for}\\
&\text{}\ sub(rhs_{M_1,f,\sigma}, \pi d) \in RHS(F, \Delta, U_p, Y_r), \text{ using the induction hypotheses}\\
&\text{}\ \text{for the } \psi'_1, \ldots, \psi'_{s''} \text{ to establish the necessary preconditions)}\\
&nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, g''\ sub(rhs_{M_1,f,\sigma}, \pi d)\\
&\qquad\qquad \psi'_1[v_{j'} \leftarrow sub(rhs_{M_1,f,\sigma}, \pi j') \quad \mid j' \in [q],\\
&\qquad\qquad\quad z_{l'} \leftarrow par_{M_2, rhs_{M_1,f,\sigma}}(\pi, g', l') \mid l' \in [s']]\\
&\qquad\qquad \cdots\\
&\qquad\qquad \psi'_{s''}[v_{j'} \leftarrow sub(rhs_{M_1,f,\sigma}, \pi j') \quad \mid j' \in [q],\\
&\qquad\qquad\quad z_{l'} \leftarrow par_{M_2, rhs_{M_1,f,\sigma}}(\pi, g', l') \mid l' \in [s']])\\
=\ &\text{(by substitution)}\\
&nf(\Rightarrow_{R_2 \cup Pre \cup Pair}, (g''\ v_d\ \psi'_1 \cdots \psi'_{s''})[v_{j'} \leftarrow sub(rhs_{M_1,f,\sigma}, \pi j') \quad \mid j' \in [q],\\
&\qquad\qquad\qquad\qquad z_{l'} \leftarrow par_{M_2, rhs_{M_1,f,\sigma}}(\pi, g', l') \mid l' \in [s']])
\end{aligned}$$

$\underline{lab(rhs_{M_1,f,\sigma}, \pi) = f'}$ for some $f' \in F^{(r'+1)}$, where $j - 1 \in [r']$:
Note that $lab(\diamond\ rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k \in [r]], 1\pi 1) = lab(rhs_{M_1,f,\sigma}, \pi 1) \in U_p$ due to $rhs_{M_1,f,\sigma} \in RHS(F, \Delta, U_p, Y_r)$ and $lab(rhs_{M_1,f,\sigma}, \pi) \in F$. Assume the value is u_i with $i \in [p]$. Then, we calculate:

$$nf(\Rightarrow_{R_2^{\diamond\star\to\circ} \cup Pre \cup Pair}, par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1\pi j, g, l))$$

$$
\begin{array}{ll}
= & \text{(by definition of } par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]} \text{ (cf. Construction 3.1))} \\
nf(\Rightarrow_{R_2^{\diamond\star\to\circ}\cup Pre\cup Pair}, & \\
\quad \overline{((j-1)_{f'} l_g}\ u\ nest_{f'}(1,g_1,\{(j-1,g)\})\ \cdots\ nest_{f'}(r',g_\mu,\{(j-1,g)\}) & \\
\qquad\qquad z_{g_1,1}\cdots z_{g_\mu,s_\mu}) & \\
\quad [u \quad \leftarrow u_i, & \\
\quad\ y'_b \quad \leftarrow sub(\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]], 1\pi(b+1)) \mid b\in[r'], & \\
\quad\ z_{g',l'} \leftarrow par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1\pi,g',l') \mid g'\in G^{(s'+1)}, l'\in[s']]) & \\
= & \text{(by definition and properties of } sub, \\
 & \quad \text{taking into account that } lab(rhs_{M_1,f,\sigma},\pi)\notin Y_r) \\
nf(\Rightarrow_{R_2^{\diamond\star\to\circ}\cup Pre\cup Pair}, & \\
\quad \overline{((j-1)_{f'} l_g}\ u\ nest_{f'}(1,g_1,\{(j-1,g)\})\ \cdots\ nest_{f'}(r',g_\mu,\{(j-1,g)\}) & \\
\qquad\qquad z_{g_1,1}\cdots z_{g_\mu,s_\mu}) & \\
\quad [u \quad \leftarrow u_i, & \\
\quad\ y'_b \quad \leftarrow sub(rhs_{M_1,f,\sigma}, \pi(b+1))[y_k \leftarrow \star\, y_k \mid k\in[r]] \mid b\in[r'], & \\
\quad\ z_{g',l'} \leftarrow par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1\pi,g',l') \quad \mid g'\in G^{(s'+1)}, l'\in[s']]) & \\
= & \text{(see below)} \\
nf(\Rightarrow_{R_2\cup Pre\cup Pair}, & \\
\quad \overline{((j-1)_{f'} l_g}\ u\ nest_{f'}(1,g_1,\{(j-1,g)\})\ \cdots\ nest_{f'}(r',g_\mu,\{(j-1,g)\}) & \\
\qquad\qquad z_{g_1,1}\cdots z_{g_\mu,s_\mu}) & \\
\quad [u \quad \leftarrow u_i, & \\
\quad\ y'_b \quad \leftarrow sub(rhs_{M_1,f,\sigma}, \pi(b+1)) \mid b\in[r'], & \\
\quad\ z_{g',l'} \leftarrow par_{M_2, rhs_{M_1,f,\sigma}}(\pi,g',l') \quad \mid g'\in G^{(s'+1)}, l'\in[s']]) & \\
= & \text{(by definition of } par_{M_2,rhs_{M_1,f,\sigma}} \text{ (cf. Construction 3.1))} \\
nf(\Rightarrow_{R_2\cup Pre\cup Pair}, par_{M_2,rhs_{M_1,f,\sigma}}(\pi j,g,l)) &
\end{array}
$$

Since for every $g'\in G^{(s'+1)}$ and $l'\in[s']$,

$$
\begin{array}{rl}
 & nf(\Rightarrow_{R_2^{\diamond\star\to\circ}\cup Pre\cup Pair}, par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1\pi,g',l')) \\
= & nf(\Rightarrow_{R_2\cup Pre\cup Pair}, par_{M_2,rhs_{M_1,f,\sigma}}(\pi,g',l'))
\end{array}
$$

by the induction hypothesis for π, the above gap can be closed (by substitution and normal forms) if we establish for every $k'\in[r']$ and $g''\in G$:

$$
\begin{array}{ll}
 & nf(\Rightarrow_{R_2^{\diamond\star\to\circ}\cup Pre\cup Pair}, \\
 & \quad nest_{f'}(k',g'',\{(j-1,g)\}) \\
 & \quad\ [u \quad \leftarrow u_i, \\
 & \quad\ y'_b \quad \leftarrow sub(rhs_{M_1,f,\sigma}, \pi(b+1))[y_k \leftarrow \star\, y_k \mid k\in[r]] \mid b\in[r'], \\
 & \quad\ z_{g',l'} \leftarrow par_{M_2^{\diamond\star\to\circ},\diamond\, rhs_{M_1,f,\sigma}[y_k \leftarrow \star\, y_k \mid k\in[r]]}(1\pi,g',l') \quad \mid g'\in G^{(s'+1)}, l'\in[s']]) \\
= & nf(\Rightarrow_{R_2\cup Pre\cup Pair}, nest_{f'}(k',g'',\{(j-1,g)\}) \\
 & \qquad\qquad [u \quad \leftarrow u_i, \\
 & \qquad\qquad\ y'_b \quad \leftarrow sub(rhs_{M_1,f,\sigma}, \pi(b+1)) \mid b\in[r'], \\
 & \qquad\qquad\ z_{g',l'} \leftarrow par_{M_2,rhs_{M_1,f,\sigma}}(\pi,g',l') \quad \mid g'\in G^{(s'+1)}, l'\in[s']]).
\end{array}
$$

But this is an instance of statement (b) from the proof of Lemma 4.10 for $sub($ $rhs_{M_1,f,\sigma}, \pi 2), \ldots, sub(rhs_{M_1,f,\sigma}, \pi(r'+1)) \in RHS(F, \Delta, U_p, Y_r)$, where the statements obtained by the induction hypothesis for π establish the necessary preconditions on the values substituted for the $z_{g',l'}$. $\square$

Proof of Lemma 4.17
We prove the following two statements:

(a) for every $t' \in T_{\Delta \cup \{\diamond,\star\}}$:
For every $g \in G^{(s+1)}$ and $\eta_1^\circ, \ldots, \eta_s^\circ, \eta_1^\bullet, \ldots, \eta_s^\bullet, \eta_1^{\circ-\bullet}, \ldots, \eta_s^{\circ-\bullet} \in T_{G \cup \Delta \cup \Omega \cup \{\diamond,\star,\circ,\bullet\}}$ with $|nf(\Rightarrow_{R_2^{\diamond\star\to\circ}}, \eta_l^\circ)|_\circ - |nf(\Rightarrow_{R_2^{\diamond\to\bullet}}, \eta_l^\bullet)|_\bullet = |nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^{\circ-\bullet})|_{\circ-\bullet}$ for every $l \in [s]$:
$|nf(\Rightarrow_{R_2^{\diamond\star\to\circ}}, g\ t'\ \eta_1^\circ \cdots \eta_s^\circ)|_\circ - |nf(\Rightarrow_{R_2^{\diamond\to\bullet}}, g\ t'[\star \twoheadleftarrow id]\ \eta_1^\bullet \cdots \eta_s^\bullet)|_\bullet$
$= |nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ t'[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]\ \eta_1^{\circ-\bullet} \cdots \eta_s^{\circ-\bullet})|_{\circ-\bullet}$.

(b) for every $q \in \mathbb{N}$ and $t'_1, \ldots, t'_q \in T_{\Delta \cup \{\diamond,\star\}}$:
For every $s \in \mathbb{N}$ and $\eta_1^\circ, \ldots, \eta_s^\circ, \eta_1^\bullet, \ldots, \eta_s^\bullet, \eta_1^{\circ-\bullet}, \ldots, \eta_s^{\circ-\bullet} \in T_{G \cup \Delta \cup \Omega \cup \{\diamond,\star,\circ,\bullet\}}$ with $|nf(\Rightarrow_{R_2^{\diamond\star\to\circ}}, \eta_l^\circ)|_\circ - |nf(\Rightarrow_{R_2^{\diamond\to\bullet}}, \eta_l^\bullet)|_\bullet = |nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^{\circ-\bullet})|_{\circ-\bullet}$ for every $l \in [s]$, for every $\psi \in RHS(G, \Omega, V_q, Z_s)$:
$|nf(\Rightarrow_{R_2^{\diamond\star\to\circ}}, \psi[v_j \leftarrow t'_j \mid j \in [q],$
$\qquad z_l \leftarrow \eta_l^\circ \mid l \in [s]])|_\circ$
$- |nf(\Rightarrow_{R_2^{\diamond\to\bullet}}, \psi[v_j \leftarrow t'_j[\star \twoheadleftarrow id] \mid j \in [q],$
$\qquad z_l \leftarrow \eta_l^\bullet \mid l \in [s]])|_\bullet$
$= |nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \psi[v_j \leftarrow t'_j[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta] \mid j \in [q],$
$\qquad z_l \leftarrow \eta_l^{\circ-\bullet} \mid l \in [s]])|_{\circ-\bullet}$.

by simultaneous induction. Lemma 4.17 then follows from statement (b) with $q = 1$, $t'_1 = t'$, $s = 0$, and $\psi = e_{M_2}$.

(a) $\Leftarrow$ (b) : by case analysis on $t' \in T_{\Delta \cup \{\diamond,\star\}}$:

$t' = \diamond\ t'_1$ for some $t'_1 \in T_{\Delta \cup \{\diamond,\star\}}$:

$|nf(\Rightarrow_{R_2^{\diamond\star\to\circ}}, g\ (\diamond\ t'_1)\ \eta_1^\circ \cdots \eta_s^\circ)|_\circ - |nf(\Rightarrow_{R_2^{\diamond\to\bullet}}, g\ (\diamond\ t'_1)[\star \twoheadleftarrow id]\ \eta_1^\bullet \cdots \eta_s^\bullet)|_\bullet$
$=$ (by substitution, $\Rightarrow_{R_2^{\diamond\star\to\circ}}$ and $\Rightarrow_{R_2^{\diamond\to\bullet}}$, using that $rhs_{M_2^{\diamond\star\to\circ},g,\diamond} = \circ\ (g\ v_1\ z_1 \cdots z_s)$ and $rhs_{M_2^{\diamond\to\bullet},g,\diamond} = \bullet\ (g\ v_1\ z_1 \cdots z_s)$ by Definitions 4.9 and 4.4, by normal forms and definitions of $|\,.\,|_\circ$ and $|\,.\,|_\bullet$)
$|nf(\Rightarrow_{R_2^{\diamond\star\to\circ}}, g\ t'_1\ \eta_1^\circ \cdots \eta_s^\circ)|_\circ - |nf(\Rightarrow_{R_2^{\diamond\to\bullet}}, g\ t'_1[\star \twoheadleftarrow id]\ \eta_1^\bullet \cdots \eta_s^\bullet)|_\bullet$
$=$ (by induction hypothesis (a) for t'_1)
$|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ t'_1[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]\ \eta_1^{\circ-\bullet} \cdots \eta_s^{\circ-\bullet})|_{\circ-\bullet}$
$=$ (by substitution)
$|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ (\diamond\ t'_1)[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]\ \eta_1^{\circ-\bullet} \cdots \eta_s^{\circ-\bullet})|_{\circ-\bullet}$

$t' = \star\ t'_1$ for some $t'_1 \in T_{\Delta \cup \{\diamond,\star\}}$:

$|nf(\Rightarrow_{R_2^{\diamond\star\to\circ}}, g\ (\star\ t'_1)\ \eta_1^\circ \cdots \eta_s^\circ)|_\circ - |nf(\Rightarrow_{R_2^{\diamond\to\bullet}}, g\ (\star\ t'_1)[\star \twoheadleftarrow id]\ \eta_1^\bullet \cdots \eta_s^\bullet)|_\bullet$

$= \quad$ (by $\Rightarrow_{R_2^{\circ\star\to\circ}}$, using that $rhs_{M_2^{\circ\star\to\circ},g,\star} = g\ v_1\ (\circ\ z_1)\ \cdots\ (\circ\ z_s)$ by Definition 4.9, and by substitution)

$|nf(\Rightarrow_{R_2^{\circ\star\to\circ}}, g\ t'_1\ (\circ\ \eta_1^\circ)\ \cdots\ (\circ\ \eta_s^\circ))|_\circ - |nf(\Rightarrow_{R_2^{\circ\to\bullet}}, g\ t'_1[\star \twoheadleftarrow id]\ \eta_1^\bullet \cdots \eta_s^\bullet)|_\bullet$

$= \quad$ (by induction hypothesis (a) for t'_1, using that for every $l \in [s]$, $|nf(\Rightarrow_{R_2^{\circ\star\to\circ}}, \circ\ \eta_l^\circ)|_\circ - |nf(\Rightarrow_{R_2^{\circ\to\bullet}}, \eta_l^\bullet)|_\bullet = |nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \circ\ \eta_l^{\circ-\bullet})|_{\circ-\bullet}$, which follows from the preconditions on the $\eta_1^\circ, \ldots, \eta_s^\circ, \eta_1^\bullet, \ldots, \eta_s^\bullet, \eta_1^{\circ-\bullet}, \ldots, \eta_s^{\circ-\bullet}$ by the definitions of $|\,.\,|_\circ$ and $|\,.\,|_{\circ-\bullet}$)

$|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ t'_1[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]\ (\circ\ \eta_1^{\circ-\bullet})\ \cdots\ (\circ\ \eta_s^{\circ-\bullet}))|_{\circ-\bullet}$

$= \quad$ (by substitution and $\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}$, using that $rhs_{M_2^{\star\bullet\to\circ\bullet},g,\star} = g\ v_1\ (\circ\ z_1)\ \cdots\ (\circ\ z_s)$ by Definition 4.15)

$|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ (\star\ t'_1)[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]\ \eta_1^{\circ-\bullet} \cdots \eta_s^{\circ-\bullet})|_{\circ-\bullet}$

$\underline{t' = \delta'\ t'_1 \cdots t'_q}$ for some $\delta' \in \Delta^{(q)}$ and $t'_1, \ldots, t'_q \in T_{\Delta\cup\{\diamond,\star\}}$:

$|nf(\Rightarrow_{R_2^{\circ\star\to\circ}}, g\ (\delta'\ t'_1 \cdots t'_q)\ \eta_1^\circ \cdots \eta_s^\circ)|_\circ$
$-|nf(\Rightarrow_{R_2^{\circ\to\bullet}}, g\ (\delta'\ t'_1 \cdots t'_q)[\star \twoheadleftarrow id]\ \eta_1^\bullet \cdots \eta_s^\bullet)|_\bullet$

$= \quad$ (by substitution, $\Rightarrow_{R_2^{\circ\star\to\circ}}$ and $\Rightarrow_{R_2^{\circ\to\bullet}}$, using that $rhs_{M_2^{\circ\star\to\circ},g,\delta'} = rhs_{M_2,g,\delta'}$ and $rhs_{M_2^{\circ\to\bullet},g,\delta'} = \bullet\ rhs_{M_2,g,\delta'}$ by Definitions 4.9 and 4.4, by substitution, normal form, and definition of $|\,.\,|_\bullet$)

$-1 + |nf(\Rightarrow_{R_2^{\circ\star\to\circ}}, rhs_{M_2,g,\delta'}[v_j \leftarrow t'_j \mid j \in [q], z_l \leftarrow \eta_l^\circ \mid l \in [s]])|_\circ$
$\quad -|nf(\Rightarrow_{R_2^{\circ\to\bullet}}, rhs_{M_2,g,\delta'}[v_j \leftarrow t'_j[\star \twoheadleftarrow id] \mid j \in [q], z_l \leftarrow \eta_l^\bullet \mid l \in [s]])|_\bullet$

$= \quad$ (by induction hypothesis (b) with $\psi = rhs_{M_2,g,\delta'}$)

$-1 + |nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, rhs_{M_2,g,\delta'}[v_j \leftarrow t'_j[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta] \mid j \in [q], z_l \leftarrow \eta_l^{\circ-\bullet} \mid l \in [s]])|_{\circ-\bullet}$

$= \quad$ (by $\Rightarrow^2_{R_2^{\star\bullet\to\circ\bullet}}$, using that $rhs_{M_2^{\star\bullet\to\circ\bullet},g,\bullet} = \bullet\ (g\ v_1\ z_1 \cdots z_s)$ and $rhs_{M_2^{\star\bullet\to\circ\bullet},g,\delta'} = rhs_{M_2,g,\delta'}$ by Definition 4.15, by normal form and definition of $|\,.\,|_{\circ-\bullet}$)

$|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ (\bullet\ (\delta'\ t'_1[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]$
$\qquad \cdots$
$\qquad t'_q[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]))\ \eta_1^{\circ-\bullet} \cdots \eta_s^{\circ-\bullet})|_{\circ-\bullet}$

$= \quad$ (by substitution)

$|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ (\delta'\ t'_1 \cdots t'_q)[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet \cdot \delta \mid \delta \in \Delta]\ \eta_1^{\circ-\bullet} \cdots \eta_s^{\circ-\bullet})|_{\circ-\bullet}$

(b) $\Leftarrow$ (a) : for fixed $s \in \mathbb{N}$, $\eta_1^\circ, \ldots, \eta_s^\circ, \eta_1^\bullet, \ldots, \eta_s^\bullet, \eta_1^{\circ-\bullet}, \ldots, \eta_s^{\circ-\bullet} \in T_{G\cup\Delta\cup\Omega\cup\{\diamond,\star,\circ,\bullet\}}$ with $|nf(\Rightarrow_{R_2^{\circ\star\to\circ}}, \eta_l^\circ)|_\circ - |nf(\Rightarrow_{R_2^{\circ\to\bullet}}, \eta_l^\bullet)|_\bullet = |nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^{\circ-\bullet})|_{\circ-\bullet}$ for every $l \in [s]$, by structural induction on $\psi \in RHS(G, \Omega, V_q, Z_s)$. The validity in the base case $\psi \in Z_s$ follows from the preconditions on the $\eta_1^\circ, \ldots, \eta_s^\circ, \eta_1^\bullet, \ldots, \eta_s^\bullet, \eta_1^{\circ-\bullet}, \ldots, \eta_s^{\circ-\bullet}$. The one recursive case, $\psi = \omega\ \psi_1 \cdots \psi_n$, is straightforward and in the other one, $\psi = g\ v_j\ \psi_1 \cdots \psi_{s'}$, the induction hypotheses for the $\psi_1, \ldots, \psi_{s'}$ establish the preconditions necessary to apply induction hypothesis (a) for t'_j, which suffices. □

Proof of Lemma 4.18
For every $t \in T_\Sigma$:

$$
\begin{aligned}
&|\tau_{M_2^{\diamond\star}\to\circ}(\tau_{M_1^{\to\diamond\star}}(t))|_\circ - |\tau_{M_2^{\diamond}\to\bullet}(\tau_{M_1^{\to\diamond}}(t))|_\bullet \\
&\quad = \quad \text{(by Lemma 4.16(1))} \\
&|\tau_{M_2^{\diamond\star}\to\circ}(\tau_{M_1^{\to\diamond\star}}(t))|_\circ - |\tau_{M_2^{\diamond}\to\bullet}(\tau_{M_1^{\to\diamond\star}}(t)[\star \twoheadleftarrow id])|_\bullet \\
&\quad = \quad \text{(by Lemma 4.17 with } t' = \tau_{M_1^{\to\diamond\star}}(t)) \\
&|\tau_{M_2^{\star\bullet}\to\circ\bullet}(\tau_{M_1^{\to\diamond\star}}(t)[\diamond \twoheadleftarrow id][\delta \twoheadleftarrow \bullet\cdot\delta \mid \delta\in\Delta])|_{\circ-\bullet} \\
&\quad = \quad \text{(by Lemma 4.16(2))} \\
&|\tau_{M_2^{\star\bullet}\to\circ\bullet}(\tau_{M_1^{\to\star\bullet}}(t))|_{\circ-\bullet} \quad \square
\end{aligned}
$$

Proof of Lemma 4.23
For fixed $\kappa : \{(f,k) \mid f \in F^{(r+1)}, k \in [r]\} \longrightarrow \{0,\ldots,s_{max}\}$ we prove the following two statements:

(a) <u>for every $t \in T_\Sigma$:</u>
For every $f \in F^{(r+1)}$ and $\theta_1^\circ,\ldots,\theta_r^\circ,\theta_1^\star,\ldots,\theta_r^\star \in T_{F\cup\Sigma\cup\Delta\cup\{\bullet,\circ,\star\}}$ with $nf(\Rightarrow_{R_1^{\to\circ\bullet,\kappa}}, \theta_k^\circ) = nf(\Rightarrow_{R_1^{\to\star\bullet}}, \theta_k^\star)[\star \twoheadleftarrow \circ^{s_{max}}]$ for every $k \in [r]$:
$nf(\Rightarrow_{R_1^{\to\circ\bullet,\kappa}}, f\ t\ (\circ^{\kappa_{f,1}}\ \theta_1^\circ)\ \cdots\ (\circ^{\kappa_{f,r}}\ \theta_r^\circ)) = nf(\Rightarrow_{R_1^{\to\star\bullet}}, f\ t\ \theta_1^\star\cdots\theta_r^\star)[\star \twoheadleftarrow \circ^{s_{max}}]$.

(b) <u>for every $p \in \mathbb{N}$ and $t_1,\ldots,t_p \in T_\Sigma$:</u>
For every $r \in \mathbb{N}$ and $\theta_1^\circ,\ldots,\theta_r^\circ,\theta_1^\star,\ldots,\theta_r^\star \in T_{F\cup\Sigma\cup\Delta\cup\{\bullet,\circ,\star\}}$ with $nf(\Rightarrow_{R_1^{\to\circ\bullet,\kappa}}, \theta_k^\circ) = nf(\Rightarrow_{R_1^{\to\star\bullet}}, \theta_k^\star)[\star \twoheadleftarrow \circ^{s_{max}}]$ for every $k \in [r]$, for every $\phi \in RHS(F, \Delta\cup\{\bullet\}, U_p, Y_r)$:

$$
\begin{aligned}
&nf(\Rightarrow_{R_1^{\to\circ\bullet,\kappa}}, \phi[f' \twoheadleftarrow f'\cdot(id\times\circ^{\kappa_{f',1}}\cdots\times\circ^{\kappa_{f',r'}}) \mid f'\in F^{(r'+1)}] \\
&\qquad\qquad [u_1,\ldots,u_p,y_1,\ldots,y_r \leftarrow t_1,\ldots,t_p,\theta_1^\circ,\ldots,\theta_r^\circ]) \\
&= nf(\Rightarrow_{R_1^{\to\star\bullet}}, \phi[u_1,\ldots,u_p,y_1,\ldots,y_r \leftarrow t_1,\ldots,t_p,\theta_1^\star,\ldots,\theta_r^\star])[\star \twoheadleftarrow \circ^{s_{max}}].
\end{aligned}
$$

by simultaneous induction. Lemma 4.23 then follows from statement (b) with $p=1$, $t_1 = t$, $r = 0$, and $\phi = e_{M_1}[\delta \twoheadleftarrow \bullet\cdot\delta \mid \delta\in\Delta]$, taking into account the definitions of $e_{M_1^{\to\circ\bullet,\kappa}}$ and $e_{M_1^{\to\star\bullet}}$ (according to Definitions 4.21 and 4.15, respectively).

(a) $\Leftarrow$ (b) : Assume $t = \sigma\ t_1\cdots t_p$ for $\sigma\in\Sigma^{(p)}$ and $t_1,\ldots,t_p \in T_\Sigma$. We calculate:

$$
\begin{aligned}
&nf(\Rightarrow_{R_1^{\to\circ\bullet,\kappa}}, f\ (\sigma\ t_1\cdots t_p)\ (\circ^{\kappa_{f,1}}\ \theta_1^\circ)\ \cdots\ (\circ^{\kappa_{f,r}}\ \theta_r^\circ)) \\
&\quad = \quad \text{(by } \Rightarrow_{R_1^{\to\circ\bullet,\kappa}}\text{, using the definition of } rhs_{M_1^{\to\circ\bullet,\kappa},f,\sigma} \text{ according to Def. 4.21)} \\
&nf(\Rightarrow_{R_1^{\to\circ\bullet,\kappa}}, rhs_{M_1,f,\sigma}[\delta \twoheadleftarrow \bullet\cdot\delta \mid \delta\in\Delta] \\
&\qquad\qquad [f' \twoheadleftarrow f'\cdot(id\times\circ^{\kappa_{f',1}}\cdots\times\circ^{\kappa_{f',r'}}) \mid f'\in F^{(r'+1)}] \\
&\qquad\qquad [y_k \leftarrow \circ^{\overline{\kappa}_{f,k}}\ y_k \mid k\in[r]][u_i \leftarrow t_i \mid i\in[p], \\
&\qquad\qquad\qquad\qquad\qquad\qquad y_k \leftarrow \circ^{\kappa_{f,k}}\ \theta_k^\circ \mid k\in[r]]) \\
&\quad = \quad \text{(by composition of substitutions)} \\
&nf(\Rightarrow_{R_1^{\to\circ\bullet,\kappa}}, rhs_{M_1,f,\sigma}[\delta \twoheadleftarrow \bullet\cdot\delta \mid \delta\in\Delta] \\
&\qquad\qquad [f' \twoheadleftarrow f'\cdot(id\times\circ^{\kappa_{f',1}}\cdots\times\circ^{\kappa_{f',r'}}) \mid f'\in F^{(r'+1)}] \\
&\qquad\qquad [u_i \leftarrow t_i \mid i\in[p], \\
&\qquad\qquad\ y_k \leftarrow \circ^{\overline{\kappa}_{f,k}}\ (\circ^{\kappa_{f,k}}\ \theta_k^\circ) \mid k\in[r]])
\end{aligned}
$$

$$
\begin{aligned}
&= \quad \text{(see below)}\\
&nf(\Rightarrow_{R_1^{\rightarrow\star\bullet}}, rhs_{M_1,f,\sigma}[\delta \twoheadleftarrow \bullet\cdot\delta \mid \delta\in\Delta]\\
&\qquad\qquad [u_i \leftarrow t_i \mid i\in[p],\\
&\qquad\qquad\ y_k \leftarrow \star\ \theta_k^\star \mid k\in[r]])[\star \twoheadleftarrow \circ^{s_{max}}]\\
&= \quad \text{(by composition of substitutions)}\\
&nf(\Rightarrow_{R_1^{\rightarrow\star\bullet}}, rhs_{M_1,f,\sigma}[\delta \ \twoheadleftarrow \bullet\cdot\delta \mid \delta\in\Delta]\\
&\qquad\qquad [y_k \leftarrow \star\ y_k \mid k\in[r]][u_i \leftarrow t_i \ \mid i\in[p],\\
&\qquad\qquad\qquad\qquad\qquad\qquad y_k \leftarrow \theta_k^\star \mid k\in[r]])[\star \twoheadleftarrow \circ^{s_{max}}]\\
&= \quad \text{(by } \Rightarrow_{R_1^{\rightarrow\star\bullet}}\text{, using the definition of } rhs_{M_1^{\rightarrow\star\bullet},f,\sigma} \text{ according to Definition 4.15)}\\
&nf(\Rightarrow_{R_1^{\rightarrow\star\bullet}}, f\ (\sigma\ t_1\cdots t_p)\ \theta_1^\star\cdots\theta_r^\star)[\star \twoheadleftarrow \circ^{s_{max}}]
\end{aligned}
$$

The above gap can be closed by using induction hypothesis (b) with $\phi = rhs_{M_1,f,\sigma}[\delta \twoheadleftarrow \bullet\cdot\delta \mid \delta\in\Delta]$ if we establish for every $k\in[r]$:

$$nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet,\kappa}}, \circ^{\overline{\kappa}_{f,k}}\ (\circ^{\kappa_{f,k}}\ \theta_k^\circ)) = nf(\Rightarrow_{R_1^{\rightarrow\star\bullet}}, \star\ \theta_k^\star)[\star \twoheadleftarrow \circ^{s_{max}}].$$

But this follows directly (by normal forms and substitution) from the preconditions on the $\theta_1^\circ,\ldots,\theta_r^\circ,\theta_1^\star,\ldots,\theta_r^\star$ and the fact that $\overline{\kappa}_{f,k}+\kappa_{f,k}=s_{max}$ for every $k\in[r]$.

(b) $\Leftarrow$ (a) : for fixed $r\in\mathbb{N}$ and $\theta_1^\circ,\ldots,\theta_r^\circ,\theta_1^\star,\ldots,\theta_r^\star \in T_{F\cup\Sigma\cup\Delta\cup\{\bullet,\circ,\star\}}$ with $nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet,\kappa}}, \theta_k^\circ) = nf(\Rightarrow_{R_1^{\rightarrow\star\bullet}}, \theta_k^\star)[\star \twoheadleftarrow \circ^{s_{max}}]$ for every $k\in[r]$, by structural induction on $\phi\in RHS(F,\Delta\cup\{\bullet\},U_p,Y_r)$. The validity in the base case $\phi\in Y_r$ follows from the preconditions on the $\theta_1^\circ,\ldots,\theta_r^\circ,\theta_1^\star,\ldots,\theta_r^\star$. Two recursive cases, $\phi=\delta\ \phi_1\cdots\phi_n$ and $\phi=\bullet\ \phi_1$, are straightforward. The validity in the remaining case is proved as follows.

$\underline{\phi = f''\ u_i\ \phi_1\cdots\phi_{r''}}$ for some $f''\in F^{(r''+1)}$, $u_i\in U_p$, and $\phi_1,\ldots,\phi_{r''}\in RHS(F,\Delta\cup\{\bullet\},U_p,Y_r)$:

$$
\begin{aligned}
&nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet,\kappa}}, (f''\ u_i\ \phi_1\cdots\phi_{r''})[f' \twoheadleftarrow f'\cdot(id\times\circ^{\kappa_{f',1}}\cdots\times\circ^{\kappa_{f',r'}}) \mid f'\in F^{(r'+1)}]\\
&\qquad\qquad [u_1,\ldots,u_p,y_1,\ldots,y_r \leftarrow t_1,\ldots,t_p,\theta_1^\circ,\ldots,\theta_r^\circ])\\
&\quad = \quad \text{(by substitution)}\\
&nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet,\kappa}}, f''\ t_i\ (\circ^{\kappa_{f'',1}}\ \phi_1[f' \twoheadleftarrow f'\cdot(id\times\circ^{\kappa_{f',1}}\cdots\times\circ^{\kappa_{f',r'}}) \mid f'\in F^{(r'+1)}]\\
&\qquad\qquad\qquad [u_1,\ldots,u_p,y_1,\ldots,y_r \leftarrow t_1,\ldots,t_p,\theta_1^\circ,\ldots,\theta_r^\circ])\\
&\qquad\qquad \cdots\\
&\qquad\qquad (\circ^{\kappa_{f'',r''}}\ \phi_{r''}[f' \twoheadleftarrow f'\cdot(id\times\circ^{\kappa_{f',1}}\cdots\times\circ^{\kappa_{f',r'}}) \mid f'\in F^{(r'+1)}]\\
&\qquad\qquad\qquad [u_1,\ldots,u_p,y_1,\ldots,y_r \leftarrow t_1,\ldots,t_p,\theta_1^\circ,\ldots,\theta_r^\circ]))\\
&\quad = \quad \text{(by induction hypothesis (a) for } t_i\text{, using the induction hypotheses for}\\
&\qquad\quad \text{the } \phi_1,\ldots,\phi_{r''} \text{ to establish the necessary preconditions)}\\
&nf(\Rightarrow_{R_1^{\rightarrow\star\bullet}}, f''\ t_i\ \phi_1[u_1,\ldots,u_p,y_1,\ldots,y_r \leftarrow t_1,\ldots,t_p,\theta_1^\star,\ldots,\theta_r^\star]\\
&\qquad\qquad \cdots\\
&\qquad\qquad \phi_{r''}[u_1,\ldots,u_p,y_1,\ldots,y_r \leftarrow t_1,\ldots,t_p,\theta_1^\star,\ldots,\theta_r^\star])[\star \twoheadleftarrow \circ^{s_{max}}]\\
&\quad = \quad \text{(by substitution)}\\
&nf(\Rightarrow_{R_1^{\rightarrow\star\bullet}},\\
&\quad (f''\ u_i\ \phi_1\cdots\phi_{r''})[u_1,\ldots,u_p,y_1,\ldots,y_r \leftarrow t_1,\ldots,t_p,\theta_1^\star,\ldots,\theta_r^\star])[\star \twoheadleftarrow \circ^{s_{max}}] \quad \square
\end{aligned}
$$

Proof of Lemma 4.24
For a context-linear mtt M_2 we prove the following two statements:

(a) for every $t' \in T_{\Delta\cup\{\star,\bullet\}}$:
For every $g \in G^{(s+1)}$ and $\eta_1^\circ, \dots, \eta_s^\circ, \eta_1, \dots, \eta_s \in T_{G\cup\Delta\cup\Omega\cup\{\star,\bullet,\circ\}}$:
$|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ t'\ \eta_1^\circ \cdots \eta_s^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ t'[\star \twoheadleftarrow \circ^{s_{max}}]\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$
$\leq \sum_{l\in[s]} |nf(\Rightarrow_{R_2}, g\ t'[\star, \bullet \twoheadleftarrow id, id]\ z_1 \cdots z_s)|_{z_l}$
$* (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})$.
If additionally M_2 is context-nondeleting and $G = G^{(s_{max}+1)}$, then equality holds.

(b) for every $q \in \mathbb{N}$ and $t'_1, \dots, t'_q \in T_{\Delta\cup\{\star,\bullet\}}$:
For every $s \in \mathbb{N}$, $\eta_1^\circ, \dots, \eta_s^\circ, \eta_1, \dots, \eta_s \in T_{G\cup\Delta\cup\Omega\cup\{\star,\bullet,\circ\}}$, and $\psi \in RHS(G, \Omega, V_q, Z_s)$:
$|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \psi[v_j \leftarrow t'_j \mid j \in [q],$
$z_l \leftarrow \eta_l^\circ \mid l \in [s]])|_{\circ-\bullet}$
$- |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi[v_j \leftarrow t'_j[\star \twoheadleftarrow \circ^{s_{max}}] \mid j \in [q],$
$z_l \leftarrow \eta_l \mid l \in [s]])|_{\circ-\bullet}$
$\leq \sum_{l\in[s]} |nf(\Rightarrow_{R_2}, \psi[v_j \leftarrow t'_j[\star, \bullet \twoheadleftarrow id, id] \mid j \in [q]])|_{z_l}$
$* (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})$.
If additionally M_2 is context-nondeleting and $G = G^{(s_{max}+1)}$, then equality holds.

by simultaneous induction. Lemma 4.24 then follows from statement (b) with $q = 1$, $t'_1 = t'$, $s = 0$, and $\psi = e_{M_2}$, taking into account that an empty sum is 0.

(a) $\Leftarrow$ (b) : by case analysis on $t' \in T_{\Delta\cup\{\star,\bullet\}}$:

$t' = \star\ t'_1$ for some $t'_1 \in T_{\Delta\cup\{\star,\bullet\}}$:

$|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ (\star\ t'_1)\ \eta_1^\circ \cdots \eta_s^\circ)|_{\circ-\bullet}$
$-|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\star\ t'_1)[\star \twoheadleftarrow \circ^{s_{max}}]\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$
$=$ (by substitution, $\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}$ and $\Rightarrow^{s_{max}}_{R_2^{\circ\bullet\to\circ\bullet}}$, using that
$rhs_{M_2^{\star\bullet\to\circ\bullet},g,\star} = g\ v_1\ (\circ\ z_1)\ \cdots\ (\circ\ z_s)$ and $rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\circ} = \circ\ (g\ v_1\ z_1 \cdots z_s)$
by Definitions 4.15 and 4.21, by normal form and definition of $|\,.\,|_{\circ-\bullet}$)
$-s_{max} + |nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ t'_1\ (\circ\ \eta_1^\circ)\ \cdots\ (\circ\ \eta_s^\circ))|_{\circ-\bullet}$
$-|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ t'_1[\star \twoheadleftarrow \circ^{s_{max}}]\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$
$\leq$ (by induction hypothesis (a) for t'_1)
$-s_{max} + \sum_{l\in[s]} |nf(\Rightarrow_{R_2}, g\ t'_1[\star, \bullet \twoheadleftarrow id, id]\ z_1 \cdots z_s)|_{z_l}$
$* (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \circ\ \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})$
$=$ (by normal form, definition of $|\,.\,|_{\circ-\bullet}$, and sum manipulation)
$-s_{max} + \sum_{l\in[s]} |nf(\Rightarrow_{R_2}, g\ t'_1[\star, \bullet \twoheadleftarrow id, id]\ z_1 \cdots z_s)|_{z_l}$
$* (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})$
$+ \sum_{l\in[s]} |nf(\Rightarrow_{R_2}, g\ t'_1[\star, \bullet \twoheadleftarrow id, id]\ z_1 \cdots z_s)|_{z_l}$

$$
\begin{array}{ll}
\leq & \text{(by Lemma 2.9 for the context-linear mtt } M_2\text{)} \\
-s_{max} + \sum\limits_{l\in[s]} & |nf(\Rightarrow_{R_2}, g\ t_1'[\star, \bullet \twoheadleftarrow id, id]\ z_1 \cdots z_s)|_{z_l} \\
& * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet}) \\
& + \sum\limits_{l\in[s]} 1 \\
\leq & \text{(by } \sum\limits_{l\in[s]} 1 = s \leq s_{max}\text{)} \\
\sum\limits_{l\in[s]} & |nf(\Rightarrow_{R_2}, g\ t_1'[\star, \bullet \twoheadleftarrow id, id]\ z_1 \cdots z_s)|_{z_l} \\
& * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet}) \\
= & \text{(by substitution)} \\
\sum\limits_{l\in[s]} & |nf(\Rightarrow_{R_2}, g\ (\star\ t_1')[\star, \bullet \twoheadleftarrow id, id]\ z_1 \cdots z_s)|_{z_l} \\
& * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})
\end{array}
$$

If additionally M_2 is context-nondeleting and $G = G^{(s_{max}+1)}$, then induction hypothesis (a) and Lemma 2.9 give equalities instead of inequalities. Moreover, we then also have $s = s_{max}$; hence, all inequalities in the above calculation are turned into equalities then.

$\underline{t' = \bullet\ t_1'}$ for some $t_1' \in T_{\Delta\cup\{\star,\bullet\}}$:

$$
\begin{array}{ll}
& |nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ (\bullet\ t_1')\ \eta_1^\circ \cdots \eta_s^\circ)|_{\circ-\bullet} \\
& -|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\bullet\ t_1')[\star \twoheadleftarrow \circ^{s_{max}}]\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} \\
= & \text{(by substitution, } \Rightarrow_{R_2^{\star\bullet\to\circ\bullet}} \text{ and } \Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}\text{, using that} \\
& \quad rhs_{M_2^{\star\bullet\to\circ\bullet},g,\bullet} = rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\bullet} = \bullet\ (g\ v_1\ z_1 \cdots z_s) \text{ by Definitions 4.15} \\
& \quad \text{and 4.21, by normal forms and definition of } |\,.\,|_{\circ-\bullet}\text{)} \\
& |nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ t_1'\ \eta_1^\circ \cdots \eta_s^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ t_1'[\star \twoheadleftarrow \circ^{s_{max}}]\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} \\
\leq & \text{(by induction hypothesis (a) for } t_1'\text{)} \\
& \sum\limits_{l\in[s]} |nf(\Rightarrow_{R_2}, g\ t_1'[\star, \bullet \twoheadleftarrow id, id]\ z_1 \cdots z_s)|_{z_l} \\
& * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet}) \\
= & \text{(by substitution)} \\
& \sum\limits_{l\in[s]} |nf(\Rightarrow_{R_2}, g\ (\bullet\ t_1')[\star, \bullet \twoheadleftarrow id, id]\ z_1 \cdots z_s)|_{z_l} \\
& * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})
\end{array}
$$

If additionally M_2 is context-nondeleting and $G = G^{(s_{max}+1)}$, then induction hypothesis (a) gives equality instead of inequality in the above calculation.

$\underline{t' = \delta\ t_1' \cdots t_q'}$ for some $\delta \in \Delta^{(q)}$ and $t_1', \ldots, t_q' \in T_{\Delta\cup\{\star,\bullet\}}$:

$$
\begin{array}{ll}
& |nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, g\ (\delta\ t_1' \cdots t_q')\ \eta_1^\circ \cdots \eta_s^\circ)|_{\circ-\bullet} \\
& -|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\delta\ t_1' \cdots t_q')[\star \twoheadleftarrow \circ^{s_{max}}]\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} \\
= & \text{(by substitution, } \Rightarrow_{R_2^{\star\bullet\to\circ\bullet}} \text{ and } \Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}\text{, using that} \\
& \quad rhs_{M_2^{\star\bullet\to\circ\bullet},g,\delta} = rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\delta} = rhs_{M_2,g,\delta} \text{ by Definitions 4.15 and 4.21)} \\
& |nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, rhs_{M_2,g,\delta}[v_j \leftarrow t_j' \mid j \in [q], \\
& \qquad\qquad z_l \leftarrow \eta_l^\circ \mid l \in [s]])|_{\circ-\bullet} \\
& -|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, rhs_{M_2,g,\delta}[v_j \leftarrow t_j'[\star \twoheadleftarrow \circ^{s_{max}}] \mid j \in [q], \\
& \qquad\qquad z_l \leftarrow \eta_l \mid l \in [s]])|_{\circ-\bullet}
\end{array}
$$

$$
\begin{aligned}
&\leq \quad \text{(by induction hypothesis (b) with } \psi = rhs_{M_2,g,\delta}) \\
&\sum_{l\in[s]} |nf(\Rightarrow_{R_2}, rhs_{M_2,g,\delta}[v_j \leftarrow t'_j[\star, \bullet \nleftarrow id, id] \mid j \in [q]])|_{z_l} \\
&\quad * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet}) \\
&= \quad \text{(by substitution and } \Rightarrow_{R_2}) \\
&\sum_{l\in[s]} |nf(\Rightarrow_{R_2}, g\ (\delta\ t'_1 \cdots t'_q)[\star, \bullet \nleftarrow id, id]\ z_1 \cdots z_s)|_{z_l} \\
&\quad * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})
\end{aligned}
$$

If additionally M_2 is context-nondeleting and $G = G^{(s_{max}+1)}$, then induction hypothesis (b) gives equality instead of inequality in the above calculation.

(b) $\Leftarrow$ (a) : for fixed $s \in \mathbb{N}$ and $\eta_1^\circ, \ldots, \eta_s^\circ, \eta_1, \ldots, \eta_s \in T_{G\cup\Delta\cup\Omega\cup\{\star,\bullet,\circ\}}$ by structural induction on $\psi \in RHS(G, \Omega, V_q, Z_s)$:

$\underline{\psi = z_{l'} \in Z_s}$:

$$
\begin{aligned}
&|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, z_{l'}[v_j \leftarrow t'_j \mid j \in [q], \\
&\qquad\qquad z_l \leftarrow \eta_l^\circ \mid l \in [s]])|_{\circ-\bullet} \\
&-|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, z_{l'}[v_j \leftarrow t'_j[\star \nleftarrow \circ^{s_{max}}] \mid j \in [q], \\
&\qquad\qquad z_l \leftarrow \eta_l \mid l \in [s]])|_{\circ-\bullet} \\
&= \quad \text{(by substitution)} \\
&|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_{l'}^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_{l'})|_{\circ-\bullet} \\
&= \quad \text{(by } |z_{l'}|_{z_{l'}} = 1 \text{ and } |z_{l'}|_{z_l} = 0 \text{ for } l \neq l') \\
&\sum_{l\in[s]} |z_{l'}|_{z_l} * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet}) \\
&= \quad \text{(by substitution and normal form)} \\
&\sum_{l\in[s]} |nf(\Rightarrow_{R_2}, z_{l'}[v_j \leftarrow t'_j[\star, \bullet \nleftarrow id, id] \mid j \in [q]])|_{z_l} \\
&\quad * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})
\end{aligned}
$$

$\underline{\psi = \omega\ \psi_1 \cdots \psi_n}$ for some $\omega \in \Omega^{(n)}$ and $\psi_1, \ldots, \psi_n \in RHS(G, \Omega, V_q, Z_s)$:

$$
\begin{aligned}
&|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, (\omega\ \psi_1 \cdots \psi_n)[v_j \leftarrow t'_j \mid j \in [q], \\
&\qquad\qquad z_l \leftarrow \eta_l^\circ \mid l \in [s]])|_{\circ-\bullet} \\
&-|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, (\omega\ \psi_1 \cdots \psi_n)[v_j \leftarrow t'_j[\star \nleftarrow \circ^{s_{max}}] \mid j \in [q], \\
&\qquad\qquad z_l \leftarrow \eta_l \mid l \in [s]])|_{\circ-\bullet} \\
&= \quad \text{(by substitution, normal forms, and definition of } |\,.\,|_{\circ-\bullet}) \\
&\sum_{b\in[n]} (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \psi_b[v_j \leftarrow t'_j \mid j \in [q], \\
&\qquad\qquad z_l \leftarrow \eta_l^\circ \mid l \in [s]])|_{\circ-\bullet} \\
&\qquad -|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_b[v_j \leftarrow t'_j[\star \nleftarrow \circ^{s_{max}}] \mid j \in [q], \\
&\qquad\qquad z_l \leftarrow \eta_l \mid l \in [s]])|_{\circ-\bullet}) \\
&\leq \quad \text{(by the induction hypotheses for the } \psi_1, \ldots, \psi_n) \\
&\sum_{b\in[n]} \sum_{l\in[s]} |nf(\Rightarrow_{R_2}, \psi_b[v_j \leftarrow t'_j[\star, \bullet \nleftarrow id, id] \mid j \in [q]])|_{z_l} \\
&\quad * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet}) \\
&= \quad \text{(by sum manipulation)} \\
&\sum_{l\in[s]} \Big(\sum_{b\in[n]} |nf(\Rightarrow_{R_2}, \psi_b[v_j \leftarrow t'_j[\star, \bullet \nleftarrow id, id] \mid j \in [q]])|_{z_l} \Big) \\
&\quad * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})
\end{aligned}
$$

$= \quad$ (by substitution, normal form, and definition of $|\,\cdot\,|_{z_l}$)
$\sum_{l\in[s]} |nf(\Rightarrow_{R_2}, (\omega\ \psi_1\cdots\psi_n)[v_j \leftarrow t'_j[\star, \bullet \twoheadleftarrow id, id] \mid j\in[q]])|_{z_l}$
$\quad * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})$

If additionally M_2 is context-nondeleting and $G = G^{(s_{max}+1)}$, then the induction hypotheses for the $\psi_1, \ldots, \psi_n$ give equality instead of inequality in the above calculation.

$\underline{\psi = g\ v_{j'}\ \psi_1\cdots\psi_{s'}}$ for some $g \in G^{(s'+1)}$, $v_{j'} \in V_q$, and $\psi_1, \ldots, \psi_{s'} \in RHS(G, \Omega, V_q, Z_s)$:

$|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, (g\ v_{j'}\ \psi_1\cdots\psi_{s'})[v_j \leftarrow t'_j \mid j\in[q],$
$\qquad z_l \leftarrow \eta_l^\circ \mid l\in[s]])|_{\circ-\bullet}$
$-|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, (g\ v_{j'}\ \psi_1\cdots\psi_{s'})[v_j \leftarrow t'_j[\star \twoheadleftarrow \circ^{s_{max}}] \mid j\in[q],$
$\qquad z_l \leftarrow \eta_l \mid l\in[s]])|_{\circ-\bullet}$
$\leq \quad$ (by substitution and induction hypothesis (a) for $t'_{j'}$)
$\sum_{l'\in[s']} |nf(\Rightarrow_{R_2}, g\ t'_{j'}[\star, \bullet \twoheadleftarrow id, id]\ z_1\cdots z_{s'})|_{z_{l'}}$
$\quad * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \psi_{l'}[v_j \leftarrow t'_j \mid j\in[q],$
$\qquad z_l \leftarrow \eta_l^\circ \mid l\in[s]])|_{\circ-\bullet}$
$\qquad -|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_{l'}[v_j \leftarrow t'_j[\star \twoheadleftarrow \circ^{s_{max}}] \mid j\in[q],$
$\qquad z_l \leftarrow \eta_l \mid l\in[s]])|_{\circ-\bullet})$
$\leq \quad$ (by the induction hypotheses for the $\psi_1, \ldots, \psi_{s'}$,
$\quad$ using that the factors $|nf(\Rightarrow_{R_2}, g\ t'_{j'}[\star, \bullet \twoheadleftarrow id, id]\ z_1\cdots z_{s'})|_{z_{l'}} \geq 0$)
$\sum_{l'\in[s']} |nf(\Rightarrow_{R_2}, g\ t'_{j'}[\star, \bullet \twoheadleftarrow id, id]\ z_1\cdots z_{s'})|_{z_{l'}}$
$\quad * \sum_{l\in[s]} |nf(\Rightarrow_{R_2}, \psi_{l'}[v_j \leftarrow t'_j[\star, \bullet \twoheadleftarrow id, id] \mid j\in[q]])|_{z_l}$
$\qquad * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})$
$= \quad$ (by sum manipulation)
$\sum_{l\in[s]} \Big(\sum_{l'\in[s']} |nf(\Rightarrow_{R_2}, g\ t'_{j'}[\star, \bullet \twoheadleftarrow id, id]\ z_1\cdots z_{s'})|_{z_{l'}}$
$\qquad * |nf(\Rightarrow_{R_2}, \psi_{l'}[v_j \leftarrow t'_j[\star, \bullet \twoheadleftarrow id, id] \mid j\in[q]])|_{z_l} \Big)$
$\quad * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})$
$= \quad$ (by substitution and Lemma 2.8 for the mtt M_2)
$\sum_{l\in[s]} |nf(\Rightarrow_{R_2}, (g\ v_{j'}\ \psi_1\cdots\psi_{s'})[v_j \leftarrow t'_j[\star, \bullet \twoheadleftarrow id, id] \mid j\in[q]])|_{z_l}$
$\quad * (|nf(\Rightarrow_{R_2^{\star\bullet\to\circ\bullet}}, \eta_l^\circ)|_{\circ-\bullet} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet})$

If additionally M_2 is context-nondeleting and $G = G^{(s_{max}+1)}$, then induction hypothesis (a) and the induction hypotheses for the $\psi_1, \ldots, \psi_{s'}$ give equalities instead of inequalities in the above calculation. $\square$

Proof of Lemma 4.25
For every $t \in T_\Sigma$:

$|\tau_{M_2^{\star\bullet\to\circ\bullet}}(\tau_{M_1^{\to\star\bullet}}(t))|_{\circ-\bullet}$
$\leq \quad$ (by Lemma 4.24 with $t' = \tau_{M_1^{\to\star\bullet}}(t)$)
$|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\star\bullet}}(t)[\star \twoheadleftarrow \circ^{s_{max}}])|_{\circ-\bullet}$
$= \quad$ (by Lemma 4.23)

$|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\tau_{M_1^{\to\circ\bullet,\kappa}}(t))|_{\circ-\bullet}$

If additionally M_2 is context-nondeleting and $G = G^{(s_{max}+1)}$, then the above inequality can be strengthened to equality by Lemma 4.24. $\square$

Proof of Lemma 4.32

Let $|\,.\,|_{\bullet-\circ} = |\,.\,|_{\bullet} - |\,.\,|_{\circ}$. For an atmost mtt M_2 we prove the following two statements:

(a) for every $\theta \in T_{\Delta\cup\{\circ,\bullet\}}$:
For every $\theta' \in T_{\Delta\cup\{\circ,\bullet\}}$ with $\theta \Rightarrow^*_{\{\bullet\, v\to v\}} \theta'$, for every $g \in G^{(s+1)}$ and $\eta_1,\dots,\eta_s$, $\eta'_1,\dots,\eta'_s \in T_{G\cup\Omega\cup\Delta\cup\{\circ,\bullet\}}$:
$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta\ \eta_1\cdots\eta_s)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta'\ \eta'_1\cdots\eta'_s)|_{\bullet-\circ}$
$\le |\theta|_\bullet - |\theta'|_\bullet + \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta\ z_1\cdots z_s)|_{z_l}$
$\quad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})$.

(b) for every $q \in \mathbb{N}$ and $\theta_1,\dots,\theta_q \in T_{\Delta\cup\{\circ,\bullet\}}$:
For every $\theta'_1,\dots,\theta'_q \in T_{\Delta\cup\{\circ,\bullet\}}$ with $\theta_j \Rightarrow^*_{\{\bullet\, v\to v\}} \theta'_j$ for every $j\in[q]$, for every $s\in\mathbb{N}$, $\eta_1,\dots,\eta_s,\eta'_1,\dots,\eta'_s \in T_{G\cup\Omega\cup\Delta\cup\{\circ,\bullet\}}$, and $\psi \in RHS(G,\Omega,V_q,Z_s)$, if M_2 is context-linear or ψ contains no nested calls, then:
$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi[v_1,\dots,v_q,z_1,\dots,z_s \leftarrow \theta_1,\dots,\theta_q,\eta_1,\dots,\eta_s])|_{\bullet-\circ}$
$- |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi[v_1,\dots,v_q,z_1,\dots,z_s \leftarrow \theta'_1,\dots,\theta'_q,\eta'_1,\dots,\eta'_s])|_{\bullet-\circ}$
$\le \sum_{j\in[q]} |\psi|_{v_j} * (|\theta_j|_\bullet - |\theta'_j|_\bullet)$
$\quad + \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi[v_j \leftarrow \theta_j \mid j\in[q]])|_{z_l}$
$\quad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})$.

by simultaneous induction. The first statement of Lemma 4.32 then follows from statement (a) with $\eta'_l = \eta_l$ for every $l\in[s]$, taking into account that a sum of multiples of 0 is 0 and that $|\varrho|_{\bullet-\circ} = -|\varrho|_{\circ-\bullet}$ for every $\varrho \in T_{G\cup\Omega\cup\Delta\cup\{\circ,\bullet\}}$. The second statement of Lemma 4.32, for atleast M_2, is obtained correspondingly after replacing all occurrences of $\le$ and "-linear" by $\ge$ and "-nondeleting", respectively, in the above statements and the following proof.

(a) $\Leftarrow$ (b) : by case analysis on $\theta \in T_{\Delta\cup\{\circ,\bullet\}}$ and $\theta \Rightarrow^*_{\{\bullet\, v\to v\}} \theta'$, distinguishing four cases:

$\theta = \bullet\ \theta_1,\ \theta' = \theta'_1$ for some $\theta_1,\theta'_1 \in T_{\Delta\cup\{\circ,\bullet\}}$ with $\theta_1 \Rightarrow^*_{\{\bullet\, v\to v\}} \theta'_1$:

$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\bullet\ \theta_1)\ \eta_1\cdots\eta_s)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta'_1\ \eta'_1\cdots\eta'_s)|_{\bullet-\circ}$
$=$ (by $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$ (cf. Definition 4.21), normal form, and definition of $|\,.\,|_{\bullet-\circ}$)
$1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_1\ \eta_1\cdots\eta_s)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta'_1\ \eta'_1\cdots\eta'_s)|_{\bullet-\circ}$
$\le$ (by induction hypothesis (a) for $\theta_1 \Rightarrow^*_{\{\bullet\, v\to v\}} \theta'_1$)
$1 + |\theta_1|_\bullet - |\theta'_1|_\bullet + \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_1\ z_1\cdots z_s)|_{z_l}$
$\quad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})$
$=$ (by definition of $|\,.\,|_\bullet$, $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$ (cf. Definition 4.21), normal form, and definition of $|\,.\,|_{z_l}$)
$|\bullet\ \theta_1|_\bullet - |\theta'_1|_\bullet + \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\bullet\ \theta_1)\ z_1\cdots z_s)|_{z_l}$
$\quad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})$

$\underline{\theta = \circ\ \theta_1,\ \theta' = \circ\ \theta'_1}$ for some $\theta_1, \theta'_1 \in T_{\Delta\cup\{\circ,\bullet\}}$ with $\theta_1 \Rightarrow^*_{\{\bullet\ v \to v\}} \theta'_1$:

$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\circ\ \theta_1)\ \eta_1 \cdots \eta_s)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\circ\ \theta'_1)\ \eta'_1 \cdots \eta'_s)|_{\bullet-\circ}$
$=$ (by (twice) $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$, using that $rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\circ} = \circ\ (g\ v_1\ z_1 \cdots z_s)$ by Definition 4.21, by normal forms and definition of $|\,.\,|_{\bullet-\circ}$)
$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_1\ \eta_1 \cdots \eta_s)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta'_1\ \eta'_1 \cdots \eta'_s)|_{\bullet-\circ}$
$\leq$ (by induction hypothesis (a) for $\theta_1 \Rightarrow^*_{\{\bullet\ v\to v\}} \theta'_1$)
$|\theta_1|_\bullet - |\theta'_1|_\bullet + \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_1\ z_1 \cdots z_s)|_{z_l} * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})$
$=$ (by definition of $|\,.\,|_\bullet$, $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$, using that $rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\circ} = \circ\ (g\ v_1\ z_1 \cdots z_s)$ by Definition 4.21, by normal form and definition of $|\,.\,|_{z_l}$)
$|\circ\ \theta_1|_\bullet - |\circ\ \theta'_1|_\bullet + \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\circ\ \theta_1)\ z_1 \cdots z_s)|_{z_l} * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})$

$\underline{\theta = \bullet\ \theta_1,\ \theta' = \bullet\ \theta'_1}$ for some $\theta_1, \theta'_1 \in T_{\Delta\cup\{\circ,\bullet\}}$ with $\theta_1 \Rightarrow^*_{\{\bullet\ v\to v\}} \theta'_1$:
analogous to the previous case.

$\underline{\theta = \delta\ \theta_1 \cdots \theta_q,\ \theta' = \delta\ \theta'_1 \cdots \theta'_q}$ for some $\delta \in \Delta^{(q)}$ and $\theta_1, \ldots, \theta_q, \theta'_1, \ldots, \theta'_q \in T_{\Delta\cup\{\circ,\bullet\}}$ with $\theta_j \Rightarrow^*_{\{\bullet\ v\to v\}} \theta'_j$ for every $j \in [q]$:

$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\delta\ \theta_1 \cdots \theta_q)\ \eta_1 \cdots \eta_s)|_{\bullet-\circ}$
$-|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\delta\ \theta'_1 \cdots \theta'_q)\ \eta'_1 \cdots \eta'_s)|_{\bullet-\circ}$
$=$ (by (twice) $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$, using that $rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\delta} = rhs_{M_2,g,\delta}$ by Def. 4.21)
$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, rhs_{M_2,g,\delta}[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \theta_1, \ldots, \theta_q, \eta_1, \ldots, \eta_s])|_{\bullet-\circ}$
$-|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, rhs_{M_2,g,\delta}[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \theta'_1, \ldots, \theta'_q, \eta'_1, \ldots, \eta'_s])|_{\bullet-\circ}$
$\leq$ (by induction hypothesis (b) with $\psi = rhs_{M_2,g,\delta}$, using that M_2 is context-linear or M_2 is basic, in which case $rhs_{M_2,g,\delta}$ contains no nested calls)
$\sum_{j\in[q]} |rhs_{M_2,g,\delta}|_{v_j} * (|\theta_j|_\bullet - |\theta'_j|_\bullet)$
$+ \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, rhs_{M_2,g,\delta}[v_j \leftarrow \theta_j \mid j \in [q]])|_{z_l} * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})$
$\leq$ (by $|rhs_{M_2,g,\delta}|_{v_j} \leq 1$ for every $j \in [q]$ (since M_2 is recursion-linear), using that the factors $|\theta_j|_\bullet - |\theta'_j|_\bullet \geq 0$ due to $\theta_j \Rightarrow^*_{\{\bullet\ v\to v\}} \theta'_j$)
$\sum_{j\in[q]} (|\theta_j|_\bullet - |\theta'_j|_\bullet) + \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, rhs_{M_2,g,\delta}[v_j \leftarrow \theta_j \mid j \in [q]])|_{z_l} * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})$
$=$ (by definition of $|\,.\,|_\bullet$ and $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$, using that $rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\delta} = rhs_{M_2,g,\delta}$ by Definition 4.21)
$|\delta\ \theta_1 \cdots \theta_q|_\bullet - |\delta\ \theta'_1 \cdots \theta'_q|_\bullet + \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\delta\ \theta_1 \cdots \theta_q)\ z_1 \cdots z_s)|_{z_l} * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})$

(b) $\Leftarrow$ (a) : for fixed $\theta'_1, \ldots, \theta'_q \in T_{\Delta\cup\{\circ,\bullet\}}$ with $\theta_j \Rightarrow^*_{\{\bullet\ v\to v\}} \theta'_j$ for every $j \in [q]$, for fixed $s \in \mathbb{N}$ and $\eta_1, \ldots, \eta_s, \eta'_1, \ldots, \eta'_s \in T_{G\cup\Omega\cup\Delta\cup\{\circ,\bullet\}}$, by structural induction on $\psi \in RHS(G, \Omega, V_q, Z_s)$:

$\underline{\psi = z_{l'} \in Z_s}$:

$$
\begin{aligned}
&|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, z_{l'}[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \theta_1, \ldots, \theta_q, \eta_1, \ldots, \eta_s])|_{\bullet-\circ} \\
&-|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, z_{l'}[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \theta'_1, \ldots, \theta'_q, \eta'_1, \ldots, \eta'_s])|_{\bullet-\circ} \\
&= \quad \text{(by substitution)} \\
&|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_{l'})|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_{l'})|_{\bullet-\circ} \\
&= \quad \text{(by } |z_{l'}|_{z_{l'}} = 1 \text{ and } |z_{l'}|_{z_l} = 0 \text{ for } l \neq l') \\
&\sum_{l\in[s]} |z_{l'}|_{z_l} * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ}) \\
&= \quad \text{(by } |z_{l'}|_{v_j} = 0\text{, substitution, and normal form)} \\
&\sum_{j\in[q]} |z_{l'}|_{v_j} * (|\theta_j|_\bullet - |\theta'_j|_\bullet) + \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, z_{l'}[v_j \leftarrow \theta_j \mid j \in [q]])|_{z_l} \\
&\qquad\qquad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})
\end{aligned}
$$

$\underline{\psi = \omega\ \psi_1 \cdots \psi_n}$ for some $\omega \in \Omega^{(n)}$ and $\psi_1, \ldots, \psi_n \in RHS(G, \Omega, V_q, Z_s)$:

$$
\begin{aligned}
&|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, (\omega\ \psi_1 \cdots \psi_n)[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \theta_1, \ldots, \theta_q, \eta_1, \ldots, \eta_s])|_{\bullet-\circ} \\
&-|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, (\omega\ \psi_1 \cdots \psi_n)[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \theta'_1, \ldots, \theta'_q, \eta'_1, \ldots, \eta'_s])|_{\bullet-\circ} \\
&= \quad \text{(by substitution, normal forms, and definition of } |\,_\,|_{\bullet-\circ}) \\
&\sum_{b\in[n]} (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_b[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \theta_1, \ldots, \theta_q, \eta_1, \ldots, \eta_s])|_{\bullet-\circ} \\
&\quad -|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_b[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \theta'_1, \ldots, \theta'_q, \eta'_1, \ldots, \eta'_s])|_{\bullet-\circ}) \\
&\leq \quad \text{(by the induction hypotheses for the } \psi_1, \ldots, \psi_n\text{,} \\
&\qquad \text{using that if } \psi = \omega\ \psi_1 \cdots \psi_n \text{ contains no nested calls,} \\
&\qquad \text{then neither does any of the } \psi_1, \ldots, \psi_n) \\
&\sum_{b\in[n]} \Big(\sum_{j\in[q]} |\psi_b|_{v_j} * (|\theta_j|_\bullet - |\theta'_j|_\bullet) \\
&\qquad + \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_b[v_j \leftarrow \theta_j \mid j \in [q]])|_{z_l} \\
&\qquad\quad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ}) \Big) \\
&= \quad \text{(by sum manipulation)} \\
&\sum_{j\in[q]} \Big(\sum_{b\in[n]} |\psi_b|_{v_j} \Big) * (|\theta_j|_\bullet - |\theta'_j|_\bullet) \\
&+ \sum_{l\in[s]} \Big(\sum_{b\in[n]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_b[v_j \leftarrow \theta_j \mid j \in [q]])|_{z_l} \Big) \\
&\qquad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ}) \\
&= \quad \text{(by definition of } |\,_\,|_{v_j}\text{, substitution, normal form, and definition of } |\,_\,|_{z_l}) \\
&\sum_{j\in[q]} |\omega\ \psi_1 \cdots \psi_n|_{v_j} * (|\theta_j|_\bullet - |\theta'_j|_\bullet) \\
&+ \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, (\omega\ \psi_1 \cdots \psi_n)[v_j \leftarrow \theta_j \mid j \in [q]])|_{z_l} \\
&\quad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})
\end{aligned}
$$

$\underline{\psi = g\ v_{j'}\ \psi_1 \cdots \psi_{s'}}$ for some $g \in G^{(s'+1)}$, $v_{j'} \in V_q$, and $\psi_1, \ldots, \psi_{s'} \in RHS(G, \Omega, V_q, Z_s)$:

$$
\begin{aligned}
&|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, (g\ v_{j'}\ \psi_1 \cdots \psi_{s'})[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \theta_1, \ldots, \theta_q, \eta_1, \ldots, \eta_s])|_{\bullet-\circ} \\
&-|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, (g\ v_{j'}\ \psi_1 \cdots \psi_{s'})[v_1, \ldots, v_q, z_1, \ldots, z_s \leftarrow \theta'_1, \ldots, \theta'_q, \eta'_1, \ldots, \eta'_s])|_{\bullet-\circ} \\
&\leq \quad \text{(by substitution and induction hypothesis (a) for } \theta_{j'} \Rightarrow^*_{\{\bullet\ v\to v\}} \theta'_{j'}) \\
&\qquad\qquad\qquad

$$
\begin{aligned}
&|\theta_{j'}|_\bullet - |\theta'_{j'}|_\bullet + \sum_{l' \in [s']} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_{j'}\ z_1 \cdots z_{s'})|_{z_{l'}} \\
&\qquad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_{l'}[v_j \leftarrow \theta_j \mid j \in [q],\ z_l \leftarrow \eta_l \mid l \in [s]])|_{\bullet-\circ} \\
&\qquad\quad - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_{l'}[v_j \leftarrow \theta'_j \mid j \in [q],\ z_l \leftarrow \eta'_l \mid l \in [s]])|_{\bullet-\circ}) \\
\leq\ & \text{(by the induction hypotheses for the } \psi_1, \ldots, \psi_{s'}, \\
& \text{using that if } \psi = g\ v_{j'}\ \psi_1 \cdots \psi_{s'} \text{ contains no nested calls,} \\
& \text{then neither does any of the } \psi_1, \ldots, \psi_{s'}, \\
& \text{and that the factors } |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_{j'}\ z_1 \cdots z_{s'})|_{z_{l'}} \geq 0) \\
&|\theta_{j'}|_\bullet - |\theta'_{j'}|_\bullet + \sum_{l' \in [s']} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_{j'}\ z_1 \cdots z_{s'})|_{z_{l'}} \\
&\qquad * \Big(\sum_{j \in [q]} |\psi_{l'}|_{v_j} * (|\theta_j|_\bullet - |\theta'_j|_\bullet) \\
&\qquad\quad + \sum_{l \in [s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_{l'}[v_j \leftarrow \theta_j \mid j \in [q]])|_{z_l} \\
&\qquad\quad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ}) \Big) \\
=\ & \text{(by sum manipulation)} \\
&|\theta_{j'}|_\bullet - |\theta'_{j'}|_\bullet + \sum_{l' \in [s']} \Big(\sum_{j \in [q]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_{j'}\ z_1 \cdots z_{s'})|_{z_{l'}} * |\psi_{l'}|_{v_j} * (|\theta_j|_\bullet - |\theta'_j|_\bullet) \\
&\qquad\quad + \sum_{l \in [s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_{j'}\ z_1 \cdots z_{s'})|_{z_{l'}} \\
&\qquad\quad * |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_{l'}[v_j \leftarrow \theta_j \mid j \in [q]])|_{z_l} \\
&\qquad\quad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ}) \Big) \\
\leq\ & \text{(by } |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_{j'}\ z_1 \cdots z_{s'})|_{z_{l'}} * |\psi_{l'}|_{v_j} \leq |\psi_{l'}|_{v_j} \\
& \text{for every } l' \in [s'] \text{ and } j \in [q] \text{ (see below),} \\
& \text{using that the factors } |\theta_j|_\bullet - |\theta'_j|_\bullet \geq 0 \text{ due to } \theta_j \Rightarrow^*_{\{\bullet\ v \to v\}} \theta'_j) \\
&|\theta_{j'}|_\bullet - |\theta'_{j'}|_\bullet + \sum_{l' \in [s']} \Big(\sum_{j \in [q]} |\psi_{l'}|_{v_j} * (|\theta_j|_\bullet - |\theta'_j|_\bullet) \\
&\qquad\quad + \sum_{l \in [s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_{j'}\ z_1 \cdots z_{s'})|_{z_{l'}} \\
&\qquad\quad * |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_{l'}[v_j \leftarrow \theta_j \mid j \in [q]])|_{z_l} \\
&\qquad\quad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ}) \Big) \\
=\ & \text{(by sum manipulation)} \\
&|\theta_{j'}|_\bullet - |\theta'_{j'}|_\bullet + \sum_{j \in [q]} \Big(\sum_{l' \in [s']} |\psi_{l'}|_{v_j} \Big) * (|\theta_j|_\bullet - |\theta'_j|_\bullet) \\
&+ \sum_{l \in [s]} \Big(\sum_{l' \in [s']} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_{j'}\ z_1 \cdots z_{s'})|_{z_{l'}} \\
&\qquad * |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_{l'}[v_j \leftarrow \theta_j \mid j \in [q]])|_{z_l} \Big) \\
&\qquad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ}) \\
=\ & \text{(by } |v_{j'}|_{v_{j'}} = 1 \text{ and } |v_{j'}|_{v_j} = 0 \text{ for } j \neq j') \\
&\sum_{j \in [q]} \Big(|v_{j'}|_{v_j} + \sum_{l' \in [s']} |\psi_{l'}|_{v_j} \Big) * (|\theta_j|_\bullet - |\theta'_j|_\bullet) \\
&+ \sum_{l \in [s]} \Big(\sum_{l' \in [s']} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_{j'}\ z_1 \cdots z_{s'})|_{z_{l'}} \\
&\qquad * |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi_{l'}[v_j \leftarrow \theta_j \mid j \in [q]])|_{z_l} \Big) \\
&\qquad * (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})
\end{aligned}
$$

$=$ (by definition of $|\,.\,|_{v_j}$, substitution, and Lemma 2.8 for the mtt $M_2^{\circ\bullet\to\circ\bullet}$)
$\sum_{j\in[q]} |g\ v_{j'}\ \psi_1\cdots\psi_{s'}|_{v_j} * (|\theta_j|_\bullet - |\theta'_j|_\bullet)$
$+ \sum_{l\in[s]} |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, (g\ v_{j'}\ \psi_1\cdots\psi_{s'})[v_j \leftarrow \theta_j \mid j\in[q]])|_{z_l}$
$* (|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\bullet-\circ} - |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\bullet-\circ})$

The statement, used above, that for every $l'\in[s']$ and $j\in[q]$,

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_{j'}\ z_1\cdots z_{s'})|_{z_{l'}} * |\psi_{l'}|_{v_j} \le |\psi_{l'}|_{v_j}\,,$$

follows from the precondition that M_2, and hence also $M_2^{\circ\bullet\to\circ\bullet}$, is context-linear or $\psi = g\ v_{j'}\ \psi_1\cdots\psi_{s'}$ contains no nested calls, since in the former case,

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_{j'}\ z_1\cdots z_{s'})|_{z_{l'}} \le 1$$

by Lemma 2.9 for $M_2^{\circ\bullet\to\circ\bullet}$ (and obviously $|\psi_{l'}|_{v_j}\ge 0$), while in the latter case $|\psi_{l'}|_{v_j} = 0$ holds. $\square$

Proof of Lemma 4.34
Let $M_1^{\to\circ\bullet} = (F,\Sigma,\Delta\cup\{\circ,\bullet\}, e_{M_1^{\to\circ\bullet}}, R_1^{\to\circ\bullet})$ and $M_1^{\to\circ\bullet'} = (F,\Sigma,\Delta\cup\{\circ,\bullet\}, e_{M_1^{\to\circ\bullet'}}, R_1^{\to\circ\bullet'})$, such that $M_1^{\to\circ\bullet} \rightsquigarrow_{\mathcal{E}_{M_1,M_2}} M_1^{\to\circ\bullet'}$. We prove the following two statements (a) and (b):

(a) for every $t\in T_\Sigma$:
For every $f\in F^{(r+1)}$ and $\theta_1,\ldots,\theta_r,\theta'_1,\ldots,\theta'_r\in T_{\Delta\cup\{\circ,\bullet\}}$, if for every $k\in[r]$, $g\in G^{(s+1)}$, and $\eta_1,\ldots,\eta_s,\eta'_1,\ldots,\eta'_s\in T_{G\cup\Omega\cup\Delta\cup\{\circ,\bullet\}}$ from $|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}},\eta_l)|_{\circ-\bullet} \le |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}},\eta'_l)|_{\circ-\bullet}$ for every $l\in[s]$ follows that

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_k\ \eta_1\cdots\eta_s)|_{\circ-\bullet} \le |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta'_k\ \eta'_1\cdots\eta'_s)|_{\circ-\bullet}\,,$$

then for every $g\in G^{(s+1)}$ and $\eta_1,\ldots,\eta_s,\eta'_1,\ldots,\eta'_s\in T_{G\cup\Omega\cup\Delta\cup\{\circ,\bullet\}}$ with $|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}},\eta_l)|_{\circ-\bullet} \le |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}},\eta'_l)|_{\circ-\bullet}$ for every $l\in[s]$:

$$\begin{aligned}&|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ \theta_1\cdots\theta_r)\ \eta_1\cdots\eta_s)|_{\circ-\bullet}\\ \le\ &|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, f\ t\ \theta'_1\cdots\theta'_r)\ \eta'_1\cdots\eta'_s)|_{\circ-\bullet}.\end{aligned}$$

(b) for every $p\in\mathbb{N}$ and $t_1,\ldots,t_p\in T_\Sigma$:
For every $r\in\mathbb{N}$ and $\theta_1,\ldots,\theta_r,\theta'_1,\ldots,\theta'_r\in T_{\Delta\cup\{\circ,\bullet\}}$, if for every $k\in[r]$, $g\in G^{(s+1)}$, and $\eta_1,\ldots,\eta_s,\eta'_1,\ldots,\eta'_s\in T_{G\cup\Omega\cup\Delta\cup\{\circ,\bullet\}}$ from $|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}},\eta_l)|_{\circ-\bullet} \le |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}},\eta'_l)|_{\circ-\bullet}$ for every $l\in[s]$ follows that

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_k\ \eta_1\cdots\eta_s)|_{\circ-\bullet} \le |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta'_k\ \eta'_1\cdots\eta'_s)|_{\circ-\bullet}\,,$$

then the following statements (i) and (ii) hold:

(i) for every $\phi \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$:
For every $\phi' \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$ with $\phi \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi'$, for every $g \in G^{(s+1)}$ and $\eta_1, \ldots, \eta_s, \eta'_1, \ldots, \eta'_s \in T_{G \cup \Omega \cup \Delta \cup \{\circ,\bullet\}}$ with $|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet} \leq |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\circ-\bullet}$ for every $l \in [s]$:

$$\begin{aligned} &|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi[u_i \leftarrow t_i \mid i \in [p], \\ &\qquad\qquad y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} \\ \leq\ &|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi'[u_i \leftarrow t_i \mid i \in [p], \\ &\qquad\qquad y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}. \end{aligned}$$

(ii) for every $q \in \mathbb{N}$ and $\phi_1, \ldots, \phi_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$:
For every $\phi'_1, \ldots, \phi'_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$ with $\phi_j \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi'_j$ for every $j \in [q]$, for every $s \in \mathbb{N}$ and $\eta_1, \ldots, \eta_s, \eta'_1, \ldots, \eta'_s \in T_{G \cup \Omega \cup \Delta \cup \{\circ,\bullet\}}$ with $|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet} \leq |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\circ-\bullet}$ for every $l \in [s]$, for every $\psi \in RHS(G, \Omega, V_q, Z_s)$:

$$\begin{aligned} &|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi[v_j \leftarrow nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi_j[u_i \leftarrow t_i \mid i \in [p], \\ &\qquad\qquad y_k \leftarrow \theta_k \mid k \in [r]]) \mid j \in [q], \\ &\qquad\qquad z_l \leftarrow \eta_l \mid l \in [s]])|_{\circ-\bullet} \\ \leq\ &|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \psi[v_j \leftarrow nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi'_j[u_i \leftarrow t_i \mid i \in [p], \\ &\qquad\qquad y_k \leftarrow \theta'_k \mid k \in [r]]) \mid j \in [q], \\ &\qquad\qquad z_l \leftarrow \eta'_l \mid l \in [s]])|_{\circ-\bullet}. \end{aligned}$$

by simultaneous induction, where the two statements (i) and (ii) nested inside (b) are also proved by simultaneous induction. The first statement of Lemma 4.34,

$$\tau_{M_1^{\to\circ\bullet}}\,;\tau_{M_2^{\circ\bullet\to\circ\bullet}}\,;|\,\cdot\,|_{\circ-\bullet} \leq \tau_{M_1^{\to\circ\bullet'}}\,;\tau_{M_2^{\circ\bullet\to\circ\bullet}}\,;|\,\cdot\,|_{\circ-\bullet}\,,$$

then follows from statement (b)(ii) with $p = 1$, $r = 0$, $q = 1$, $\phi_1 = e_{M_1^{\to\circ\bullet}}$, $\phi'_1 = e_{M_1^{\to\circ\bullet'}}$, $s = 0$, and $\psi = e_{M_2}$.

If for every $\downarrow$- or $\uparrow$-rule used for rewriting the initial expression or right-hand sides of rules of $M_1^{\to\circ\bullet}$, $\mathcal{E}_{M_1,M_2}$ also contains the symmetric rule, then there exists an $M_1^{\to\circ\bullet''} \in \mathcal{M}_1^{\to\circ\bullet}$ such that $M_1^{\to\circ\bullet'} \rightsquigarrow_{\mathcal{E}_{M_1,M_2}} M_1^{\to\circ\bullet''}$ and $M_1^{\to\circ\bullet''}$ can be obtained from $M_1^{\to\circ\bullet}$ by rewriting only with the rules $\bullet\ (\circ\ v) \to v$ and $\circ\ (\bullet\ v) \to v$.[1] Since under the restriction of $\mathcal{E}_{M_1,M_2}$ to $\{\bullet\ (\circ\ v) \to v, \circ\ (\bullet\ v) \to v\}$ all inequalities in the above statements and the following proof—except for the last four cases of **(i)** $\Leftarrow$ **(ii)**, which however need not be considered under the assumed restriction of $\mathcal{E}_{M_1,M_2}$—can be strengthened to equalities, the latter implies $\tau_{M_1^{\to\circ\bullet}}\,;\tau_{M_2^{\circ\bullet\to\circ\bullet}}\,;|\,\cdot\,|_{\circ-\bullet} = \tau_{M_1^{\to\circ\bullet''}}\,;\tau_{M_2^{\circ\bullet\to\circ\bullet}}\,;|\,\cdot\,|_{\circ-\bullet}$. Together with $\tau_{M_1^{\to\circ\bullet'}}\,;\tau_{M_2^{\circ\bullet\to\circ\bullet}}\,;|\,\cdot\,|_{\circ-\bullet} \leq \tau_{M_1^{\to\circ\bullet''}}\,;\tau_{M_2^{\circ\bullet\to\circ\bullet}}\,;|\,\cdot\,|_{\circ-\bullet}$ (which follows from $M_1^{\to\circ\bullet'} \rightsquigarrow_{\mathcal{E}_{M_1,M_2}} M_1^{\to\circ\bullet''}$) we then get equality instead of the above inequality.

[1] This $M_1^{\to\circ\bullet''}$ is obtained from $M_1^{\to\circ\bullet'}$ by "undoing" all the rewrites with $\downarrow$- or $\uparrow$-rules performed in $M_1^{\to\circ\bullet} \rightsquigarrow_{\mathcal{E}_{M_1,M_2}} M_1^{\to\circ\bullet'}$, while nothing is done about the rules $\bullet\ (\circ\ v) \to v$ and $\circ\ (\bullet\ v) \to v$.

(a) $\Leftarrow$ (b) : Assume $t = \sigma\ t_1 \cdots t_p$ for $\sigma \in \Sigma^{(p)}$ and $t_1, \ldots, t_p \in T_\Sigma$. We calculate:

$$
\begin{array}{ll}
& |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ (\sigma\ t_1 \cdots t_p)\ \theta_1 \cdots \theta_r)\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} \\
= & \quad (\text{by } \Rightarrow_{R_1^{\to\circ\bullet}}) \\
& |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, rhs_{M_1^{\to\circ\bullet},f,\sigma}[u_i \leftarrow t_i \mid i \in [p], \\
& \qquad\qquad\qquad\qquad\qquad\qquad y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} \\
\leq & \quad (\text{by induction hypothesis (b), statement (i),} \\
& \quad \text{with } \phi = rhs_{M_1^{\to\circ\bullet},f,\sigma} \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} rhs_{M_1^{\to\circ\bullet'},f,\sigma} = \phi') \\
& |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, rhs_{M_1^{\to\circ\bullet'},f,\sigma}[u_i \leftarrow t_i \mid i \in [p], \\
& \qquad\qquad\qquad\qquad\qquad\qquad y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet} \\
= & \quad (\text{by } \Rightarrow_{R_1^{\to\circ\bullet'}}) \\
& |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, f\ (\sigma\ t_1 \cdots t_p)\ \theta'_1 \cdots \theta'_r)\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}
\end{array}
$$

(b) $\Leftarrow$ (a) : for fixed r and $\theta_1, \ldots, \theta_r, \theta'_1, \ldots, \theta'_r \in T_{\Delta\cup\{\circ,\bullet\}}$ with the associated preconditions, by simultaneous induction of (i) and (ii):

(i) $\Leftarrow$ (ii) : by case analysis on $\phi \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$ and $\phi \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi'$, distinguishing eleven cases:

$\underline{\phi = \phi' = y_k \in Y_r}$:

$$
\begin{array}{ll}
& |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, y_k[u_i \leftarrow t_i \mid i \in [p], \\
& \qquad\qquad\qquad\qquad\qquad\qquad y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} \\
= & \quad (\text{by substitution and since } \theta_k \in T_{\Delta\cup\{\circ,\bullet\}} \text{ is a normal form w.r.t. } \Rightarrow_{R_1^{\to\circ\bullet}}) \\
& |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta_k\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} \\
\leq & \quad (\text{by the preconditions on } \theta_k,\ \theta'_k, \text{ and the } \eta_1, \ldots, \eta_s, \eta'_1, \ldots, \eta'_s) \\
& |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ \theta'_k\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet} \\
= & \quad (\text{by substitution and since } \theta'_k \in T_{\Delta\cup\{\circ,\bullet\}} \text{ is a normal form w.r.t. } \Rightarrow_{R_1^{\to\circ\bullet'}}) \\
& |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, y_k[u_i \leftarrow t_i \mid i \in [p], \\
& \qquad\qquad\qquad\qquad\qquad\qquad y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}
\end{array}
$$

$\underline{\phi = \circ\ \phi_1,\ \phi' = \circ\ \phi'_1}$ for some $\phi_1, \phi'_1 \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$ with $\phi_1 \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi'_1$:

$$
\begin{array}{ll}
& |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, (\circ\ \phi_1)[u_i \leftarrow t_i \mid i \in [p], \\
& \qquad\qquad\qquad\qquad\qquad\qquad y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} \\
= & \quad (\text{by substitution, normal form, } \Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}} \text{ (cf. Definition 4.21),} \\
& \quad \text{normal form, and definition of } |\,\cdot\,|_{\circ-\bullet}) \\
& 1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi_1[u_i \leftarrow t_i \mid i \in [p], \\
& \qquad\qquad\qquad\qquad\qquad\qquad y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet} \\
\leq & \quad (\text{by induction hypothesis (i) for } \phi_1 \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi'_1) \\
& 1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi'_1[u_i \leftarrow t_i \mid i \in [p], \\
& \qquad\qquad\qquad\qquad\qquad\qquad y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet} \\
= & \quad (\text{by substitution, normal form, } \Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}} \text{ (cf. Definition 4.21),} \\
& \quad \text{normal form, and definition of } |\,\cdot\,|_{\circ-\bullet})
\end{array}
$$

$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, (\circ\ \phi_1')[u_i \leftarrow t_i \mid i \in [p],$
$y_k \leftarrow \theta_k' \mid k \in [r]])\ \eta_1' \cdots \eta_s')|_{\circ-\bullet}$

$\underline{\phi = \bullet\ \phi_1,\ \phi' = \bullet\ \phi_1'}$ for some $\phi_1, \phi_1' \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$ with $\phi_1 \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi_1'$:

analogous to the previous case, but with "$-1+$" instead of "$1+$".

$\underline{\phi = \delta\ \phi_1 \cdots \phi_q,\ \phi' = \delta\ \phi_1' \cdots \phi_q'}$ for some $\delta \in \Delta^{(q)}$ and $\phi_1, \ldots, \phi_q, \phi_1', \ldots, \phi_q' \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$ with $\phi_j \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi_j'$ for every $j \in [q]$:

$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, (\delta\ \phi_1 \cdots \phi_q)[u_i \leftarrow t_i \mid i \in [p],$
$y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$

$=$ (by substitution, normal form, and $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$, using that $rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\delta} = rhs_{M_2,g,\delta}$ by Definition 4.21)

$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, rhs_{M_2,g,\delta}[v_j \leftarrow nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi_j[u_i \leftarrow t_i \mid i \in [p],$
$y_k \leftarrow \theta_k \mid k \in [r]]) \mid j \in [q],$
$z_l \leftarrow \eta_l \mid l \in [s]])|_{\circ-\bullet}$

$\leq$ (by induction hypothesis (ii) with $\psi = rhs_{M_2,g,\delta}$)

$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, rhs_{M_2,g,\delta}[v_j \leftarrow nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_j'[u_i \leftarrow t_i \mid i \in [p],$
$y_k \leftarrow \theta_k' \mid k \in [r]]) \mid j \in [q],$
$z_l \leftarrow \eta_l' \mid l \in [s]])|_{\circ-\bullet}$

$=$ (by substitution, normal form, and $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$, using that $rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\delta} = rhs_{M_2,g,\delta}$ by Definition 4.21)

$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, (\delta\ \phi_1' \cdots \phi_q')[u_i \leftarrow t_i \mid i \in [p],$
$y_k \leftarrow \theta_k' \mid k \in [r]])\ \eta_1' \cdots \eta_s')|_{\circ-\bullet}$

$\underline{\phi = f\ u_{i'}\ \phi_1 \cdots \phi_{r'},\ \phi' = f\ u_{i'}\ \phi_1' \cdots \phi_{r'}'}$ for some $f \in F^{(r'+1)}$, $u_{i'} \in U_p$, and $\phi_1, \ldots, \phi_{r'}, \phi_1', \ldots, \phi_{r'}' \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$ with $\phi_k \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi_k'$ for every $k \in [r']$:

$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, (f\ u_{i'}\ \phi_1 \cdots \phi_{r'})[u_i \leftarrow t_i \mid i \in [p],$
$y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$

$=$ (by substitution and normal form)

$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t_{i'}$
$nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi_1[u_i \leftarrow t_i \mid i \in [p],$
$y_k \leftarrow \theta_k \mid k \in [r]])$
$\ldots$
$nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi_{r'}[u_i \leftarrow t_i \mid i \in [p],$
$y_k \leftarrow \theta_k \mid k \in [r]]))\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$

$\leq$ (by induction hypothesis (a) for $t_{i'}$,[2]
using the induction hypotheses (i) for $\phi_1 \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi_1'$, $\ldots$,
$\phi_{r'} \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi_{r'}'$ to establish the necessary precondition)

[2]To make sure that the "typing" agrees here, we have to establish that for every $k' \in [r']$, $nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi_{k'}[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_r])$ and $nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_{k'}'[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \theta_1', \ldots, \theta_r'])$ are in $T_{\Delta\cup\{\circ,\bullet\}}$. But this follows easily from $\phi_{k'}, \phi_{k'}' \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$, $t_1, \ldots, t_p \in T_\Sigma$, $\theta_1, \ldots, \theta_r, \theta_1', \ldots, \theta_r' \in T_{\Delta\cup\{\circ,\bullet\}}$, and the state, input, and output

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, f\ t_{i'}\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi'_1[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]])\ \cdots\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi'_{r'}[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]]))\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

$=$ (by substitution and normal form)

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, (f\ u_{i'}\ \phi'_1 \cdots \phi'_{r'})[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

$\underline{\phi = \bullet\ (\circ\ \phi_1),\ \phi' = \phi_1}$ for some $\phi_1 \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$:

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, (\bullet\ (\circ\ \phi_1))[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$$

$=$ (by substitution, normal form, $\Rightarrow^2_{R_2^{\circ\bullet\to\circ\bullet}}$, using that $rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\bullet} = \bullet\ (g\ v_1\ z_1 \cdots z_s)$ and $rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\circ} = \circ\ (g\ v_1\ z_1 \cdots z_s)$ by Definition 4.21, by normal form and definition of $|\,\blacksquare\,|_{\circ-\bullet}$)

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi_1[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$$

$\leq$ (by induction hypothesis (i) for $\phi_1 \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi_1$)

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_1[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

$\underline{\phi = \circ\ (\bullet\ \phi_1),\ \phi' = \phi_1}$ for some $\phi_1 \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$:

analogous to the previous case.

$\underline{\phi = \bullet\ (\delta\ \phi_1 \cdots \phi_q),\ \phi' = \delta\ \phi_1 \cdots (\bullet\ \phi_j) \cdots \phi_q}$ for some $\delta \in \Delta^{(q)}$, $\phi_1, \ldots, \phi_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$, and $j \in [q]$ with $\downarrow_{\delta,j} \in \mathcal{E}_{M_1,M_2}$:

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, (\bullet\ (\delta\ \phi_1 \cdots \phi_q))[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$$

$=$ (by substitution, normal form, $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$, using that $rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\bullet} = \bullet\ (g\ v_1\ z_1 \cdots z_s)$ by Definition 4.21, by normal form and definition of $|\,\blacksquare\,|_{\circ-\bullet}$)

$$-1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, (\delta\ \phi_1 \cdots \phi_q)[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$$

$\leq$ (by induction hypothesis (i) for $\delta\ \phi_1 \cdots \phi_q \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \delta\ \phi_1 \cdots \phi_q$)

$$-1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, (\delta\ \phi_1 \cdots \phi_q)[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

$=$ (by substitution and normal form)

$$-1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\delta\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_1[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]])\ \cdots\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_q[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]]))\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

ranked alphabets of $M_1^{\to\circ\bullet}$ and $M_1^{\to\circ\bullet'}$ being F, Σ, and $\Delta \cup \{\circ, \bullet\}$, respectively. Similar issues arise when applying Lemma 4.33 in other cases below, and are resolved likewise.

$\leq$ (by Lemma 4.33(1a))

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\delta\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_1[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]]) \ldots (\bullet\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_j[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]])) \ldots nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_q[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]]))\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

$=$ (by substitution and normal form)

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, (\delta\ \phi_1 \cdots (\bullet\ \phi_j) \cdots \phi_q)[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

$\underline{\phi = \delta\ \phi_1 \cdots (\bullet\ \phi_j) \cdots \phi_q,\ \phi' = \bullet\ (\delta\ \phi_1 \cdots \phi_q)}$ for some $\delta \in \Delta^{(q)}$, $\phi_1, \ldots, \phi_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$, and $j \in [q]$ with $\uparrow_{\delta,j} \in \mathcal{E}_{M_1,M_2}$:

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, (\delta\ \phi_1 \cdots (\bullet\ \phi_j) \cdots \phi_q)[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$$

$\leq$ (analogous to the case $\phi = \delta\ \phi_1 \cdots \phi_q$ above, using that $\phi_1 \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi_1, \ldots, \bullet\ \phi_j \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \bullet\ \phi_j, \ldots, \phi_q \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi_q$)

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, (\delta\ \phi_1 \cdots (\bullet\ \phi_j) \cdots \phi_q)[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

$=$ (by substitution and normal form)

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\delta\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_1[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]]) \ldots (\bullet\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_j[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]])) \ldots nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_q[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]]))\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

$\leq$ (by Lemma 4.33(1b))

$$-1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ (\delta\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_1[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]]) \ldots nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_q[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta'_k \mid k \in [r]]))\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

$=$ (by substitution, normal form, $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$, using that $rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\bullet} = \bullet\ (g\ v_1\ z_1 \cdots z_s)$ by Definition 4.21, by normal form and definition of $|\,\cdot\,|_{\circ-\bullet}$)

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, (\bullet\ (\delta\ \phi_1 \cdots \phi_q))[u_i \leftarrow t_i \mid i \in [p],\ y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

$\phi = \bullet\ (f\ u_{i'}\ \phi_1 \cdots \phi_{r'})$, $\phi' = f\ u_{i'}\ \phi_1 \cdots (\bullet\ \phi_{k'}) \cdots \phi_{r'}$ for some $f \in F^{(r'+1)}$, $u_{i'} \in U_p$, $\phi_1, \ldots, \phi_{r'} \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$, and $k' \in [r']$ with $\downarrow_{f,k'} \in \mathcal{E}_{M_1,M_2}$:

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, (\bullet\ (f\ u_{i'}\ \phi_1 \cdots \phi_{r'}))[u_i \leftarrow t_i \mid i \in [p],\ y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$$

$=$ (by substitution, normal form, $\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}$, using that $rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\bullet} = \bullet\ (g\ v_1\ z_1 \cdots z_s)$ by Definition 4.21, by normal form and definition of $|\,.\,|_{\circ-\bullet}$)

$$-1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, (f\ u_{i'}\ \phi_1 \cdots \phi_{r'})[u_i \leftarrow t_i \mid i \in [p],\ y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$$

$\leq$ (by induction hypothesis (i) for $f\ u_{i'}\ \phi_1 \cdots \phi_{r'} \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} f\ u_{i'}\ \phi_1 \cdots \phi_{r'}$)

$$-1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, (f\ u_{i'}\ \phi_1 \cdots \phi_{r'})[u_i \leftarrow t_i \mid i \in [p],\ y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

$=$ (by substitution and normal form)

$$\begin{aligned}-1 + |nf(&\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, f\ t_{i'}\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_1[u_i \leftarrow t_i \mid i \in [p],\ y_k \leftarrow \theta'_k \mid k \in [r]]) \\ &\quad \cdots \\ &\quad nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_{r'}[u_i \leftarrow t_i \mid i \in [p],\ y_k \leftarrow \theta'_k \mid k \in [r]]))\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}\end{aligned}$$

$\leq$ (by Lemma 4.33(2a) for the mtt $M_1^{\to\circ\bullet'}$)

$$\begin{aligned}|nf(&\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, f\ t_{i'}\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_1[u_i \leftarrow t_i \mid i \in [p],\ y_k \leftarrow \theta'_k \mid k \in [r]]) \\ &\quad \cdots \\ &\quad (\bullet\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_{k'}[u_i \leftarrow t_i \mid i \in [p],\ y_k \leftarrow \theta'_k \mid k \in [r]])) \\ &\quad \cdots \\ &\quad nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_{r'}[u_i \leftarrow t_i \mid i \in [p],\ y_k \leftarrow \theta'_k \mid k \in [r]]))\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}\end{aligned}$$

$=$ (by substitution and normal form)

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, (f\ u_{i'}\ \phi_1 \cdots (\bullet\ \phi_{k'}) \cdots \phi_{r'})[u_i \leftarrow t_i \mid i \in [p],\ y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}$$

$\phi = f\ u_{i'}\ \phi_1 \cdots (\bullet\ \phi_{k'}) \cdots \phi_{r'}$, $\phi' = \bullet\ (f\ u_{i'}\ \phi_1 \cdots \phi_{r'})$ for some $f \in F^{(r'+1)}$, $u_{i'} \in U_p$, $\phi_1, \ldots, \phi_{r'} \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$, and $k' \in [r']$ with $\uparrow_{f,k'} \in \mathcal{E}_{M_1,M_2}$:

$$|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, g\ nf(\Rightarrow_{R_1^{\to\circ\bullet}}, (f\ u_{i'}\ \phi_1 \cdots (\bullet\ \phi_{k'}) \cdots \phi_{r'})[u_i \leftarrow t_i \mid i \in [p],\ y_k \leftarrow \theta_k \mid k \in [r]])\ \eta_1 \cdots \eta_s)|_{\circ-\bullet}$$

$$
\begin{aligned}
&\leq \quad \text{(analogous to the case } \phi = f\ u_{i'}\ \phi_1 \cdots \phi_{r'} \text{ above,}\\
&\qquad \text{using that } \phi_1 \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi_1, \ldots, \bullet\ \phi_{k'} \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \bullet\ \phi_{k'}, \ldots, \phi_{r'} \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi_{r'})\\
&|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}},\\
&\quad g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, (f\ u_{i'}\ \phi_1 \cdots (\bullet\ \phi_{k'}) \cdots \phi_{r'})[u_i \leftarrow t_i \mid i \in [p],\\
&\qquad\qquad y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}\\
&= \quad \text{(by substitution and normal form)}\\
&|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}},\\
&\quad g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, f\ t_{i'}\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_1[u_i \leftarrow t_i \mid i \in [p],\\
&\qquad\qquad y_k \leftarrow \theta'_k \mid k \in [r]])\\
&\qquad \cdots\\
&\qquad (\bullet\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_{k'}[u_i \leftarrow t_i \mid i \in [p],\\
&\qquad\qquad y_k \leftarrow \theta'_k \mid k \in [r]]))\\
&\qquad \cdots\\
&\qquad nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_{r'}[u_i \leftarrow t_i \mid i \in [p],\\
&\qquad\qquad y_k \leftarrow \theta'_k \mid k \in [r]]))\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}\\
&\leq \quad \text{(by Lemma 4.33(2b) for the mtt } M_1^{\to\circ\bullet'})\\
&-1 + |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}},\\
&\qquad g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, f\ t_{i'}\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_1[u_i \leftarrow t_i \mid i \in [p],\\
&\qquad\qquad y_k \leftarrow \theta'_k \mid k \in [r]])\\
&\qquad\qquad \cdots\\
&\qquad\qquad nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, \phi_{r'}[u_i \leftarrow t_i \mid i \in [p],\\
&\qquad\qquad y_k \leftarrow \theta'_k \mid k \in [r]]))\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}\\
&= \quad \text{(by substitution, normal form, } \Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}} \text{, using that}\\
&\qquad rhs_{M_2^{\circ\bullet\to\circ\bullet},g,\bullet} = \bullet\ (g\ v_1\ z_1 \cdots z_s) \text{ by Definition 4.21,}\\
&\qquad \text{by normal form and definition of } |\blacksquare|_{\circ-\bullet})\\
&|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}},\\
&\quad g\ nf(\Rightarrow_{R_1^{\to\circ\bullet'}}, (\bullet\ (f\ u_{i'}\ \phi_1 \cdots \phi_{r'}))[u_i \leftarrow t_i \mid i \in [p],\\
&\qquad\qquad y_k \leftarrow \theta'_k \mid k \in [r]])\ \eta'_1 \cdots \eta'_s)|_{\circ-\bullet}
\end{aligned}
$$

(ii) $\Leftarrow$ (i) : for fixed $\phi'_1, \ldots, \phi'_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$ with $\phi_j \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi'_j$ for every $j \in [q]$, for fixed $s \in \mathbb{N}$ and $\eta_1, \ldots, \eta_s, \eta'_1, \ldots, \eta'_s \in T_{G\cup\Omega\cup\Delta\cup\{\circ,\bullet\}}$ with $|nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta_l)|_{\circ-\bullet} \leq |nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, \eta'_l)|_{\circ-\bullet}$ for every $l \in [s]$, by structural induction on $\psi \in RHS(G, \Omega, V_q, Z_s)$. The validity in the base case $\psi \in Z_s$ follows from the preconditions on the $\eta_1, \ldots, \eta_s, \eta'_1, \ldots, \eta'_s$. The one recursive case, $\psi = \omega\ \psi_1 \cdots \psi_n$, is straightforward and in the other one, $\psi = g\ v_j\ \psi_1 \cdots \psi_{s'}$, the induction hypotheses for the $\psi_1, \ldots, \psi_{s'}$ establish the preconditions necessary to apply induction hypothesis (i) for $\phi_j \Rightarrow^?_{\mathcal{E}_{M_1,M_2}} \phi'_j$, which suffices. □

Proof of Lemma 4.36
Let $M_1^{\to\circ\bullet} = (F, \Sigma, \Delta\cup\{\circ,\bullet\}, e_{M_1^{\to\circ\bullet}}, R_1^{\to\circ\bullet}) \in \mathcal{M}_1^{\to\circ\bullet}$ be such that the right-hand sides of the rules in $R_1^{\to\circ\bullet}$ contain no $\circ$-symbols. We prove the following two statements:

(a) for every $t \in T_\Sigma$:
For every $f \in F^{(r+1)}$ and $\theta_1, \ldots, \theta_r \in T_{F\cup\Sigma\cup\Delta\cup\{\circ,\bullet\}}$:

$$|nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ \theta_1 \cdots \theta_r)|_\circ = \sum_{k\in[r]} |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t\ y_1 \cdots y_r)|_{y_k} * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \theta_k)|_\circ.$$

(b) <u>for every $p \in \mathbb{N}$ and $t_1, \ldots, t_p \in T_\Sigma$:</u>
For every $r \in \mathbb{N}$, $\theta_1, \ldots, \theta_r \in T_{F\cup\Sigma\cup\Delta\cup\{\circ,\bullet\}}$, and $\phi \in RHS(F, \Delta\cup\{\circ,\bullet\}, U_p, Y_r)$, if $|\phi|_\circ = 0$ or M_1 is context-linear, then:
$|nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_r])|_\circ$
$\leq |\phi|_\circ + \sum_{k\in[r]} |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \theta_k)|_\circ.$
If $|\phi|_\circ = 0$ or additionally M_1 is context-nondeleting, then equality holds.

by simultaneous induction. Lemma 4.36 then follows from statement (b) with $p = 1$, $t_1 = t$, $r = 0$, and $\phi = e_{M_1^{\to\circ\bullet}}$, taking into account that an empty sum is 0.

(a) $\Leftarrow$ (b) : Assume $t = \sigma\ t_1 \cdots t_p$ for $\sigma \in \Sigma^{(p)}$ and $t_1, \ldots, t_p \in T_\Sigma$. We calculate:

$|nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ (\sigma\ t_1 \cdots t_p)\ \theta_1 \cdots \theta_r)|_\circ$
$=$ (by $\Rightarrow_{R_1^{\to\circ\bullet}}$)
$|nf(\Rightarrow_{R_1^{\to\circ\bullet}}, rhs_{M_1^{\to\circ\bullet},f,\sigma}[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_r])|_\circ$
$=$ (by induction hypothesis (b) with $\phi = rhs_{M_1^{\to\circ\bullet},f,\sigma}$,
using that $|rhs_{M_1^{\to\circ\bullet},f,\sigma}|_\circ = 0$ by the precondition of the lemma)
$\sum_{k\in[r]} |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, rhs_{M_1^{\to\circ\bullet},f,\sigma}[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \theta_k)|_\circ$
$=$ (by $\Rightarrow_{R_1^{\to\circ\bullet}}$)
$\sum_{k\in[r]} |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ (\sigma\ t_1 \cdots t_p)\ y_1 \cdots y_r)|_{y_k} * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \theta_k)|_\circ$

(b) $\Leftarrow$ (a) : for fixed $r \in \mathbb{N}$ and $\theta_1, \ldots, \theta_r \in T_{F\cup\Sigma\cup\Delta\cup\{\circ,\bullet\}}$ by structural induction on $\phi \in RHS(F, \Delta\cup\{\circ,\bullet\}, U_p, Y_r)$:

<u>$\phi = y_{k'} \in Y_r$</u> :

$|nf(\Rightarrow_{R_1^{\to\circ\bullet}}, y_{k'}[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_r])|_\circ$
$=$ (by substitution)
$|nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \theta_{k'})|_\circ$
$=$ (by $|y_{k'}|_{y_{k'}} = 1$ and $|y_{k'}|_{y_k} = 0$ for $k \neq k'$)
$\sum_{k\in[r]} |y_{k'}|_{y_k} * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \theta_k)|_\circ$
$=$ (by definition of $|\,.\,|_\circ$, substitution, and normal form)
$|y_{k'}|_\circ + \sum_{k\in[r]} |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, y_{k'}[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \theta_k)|_\circ$

<u>$\phi = \bullet\ \phi_1$</u> for some $\phi_1 \in RHS(F, \Delta\cup\{\circ,\bullet\}, U_p, Y_r)$:

$|nf(\Rightarrow_{R_1^{\to\circ\bullet}}, (\bullet\ \phi_1)[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_r])|_\circ$
$=$ (by substitution, normal form, and definition of $|\,.\,|_\circ$)
$0 + |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi_1[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_r])|_\circ$

$\leq$ (by the induction hypothesis for ϕ_1,
using that if $|\phi|_\circ = |\bullet\ \phi_1|_\circ = 0$, then also $|\phi_1|_\circ = 0$)
$0 + |\phi_1|_\circ + \sum_{k\in[r]} |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, \phi_1[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} * |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, \theta_k)|_\circ$
$=$ (by definition of $|\,.\,|_\circ$, substitution, normal form, and definition of $|\,.\,|_{y_k}$)
$|\bullet\ \phi_1|_\circ + \sum_{k\in[r]} |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, (\bullet\ \phi_1)[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} * |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, \theta_k)|_\circ$

If $|\phi|_\circ = |\bullet\ \phi_1|_\circ = 0$ (and thus $|\phi_1|_\circ = 0$) or additionally M_1 is context-nondeleting, then the induction hypothesis for ϕ_1 gives equality instead of inequality in the above calculation.

$\underline{\phi = \circ\ \phi_1}$ for some $\phi_1 \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$:
analogous to the previous case, but with "1+" instead of "0+".

$\underline{\phi = \delta\ \phi_1 \cdots \phi_q}$ for some $\delta \in \Delta^{(q)}$ and $\phi_1, \ldots, \phi_q \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$:

$|nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, (\delta\ \phi_1 \cdots \phi_q)[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_r])|_\circ$
$=$ (by substitution, normal form, and definition of $|\,.\,|_\circ$)
$\sum_{j\in[q]} |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, \phi_j[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_r])|_\circ$
$\leq$ (by the induction hypotheses for the $\phi_1, \ldots, \phi_q$, using
that if $|\phi|_\circ = |\delta\ \phi_1 \cdots \phi_q|_\circ = 0$, then also $|\phi_j|_\circ = 0$ for every $j \in [q]$)
$\sum_{j\in[q]} \Big(|\phi_j|_\circ + \sum_{k\in[r]} |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, \phi_j[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} * |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, \theta_k)|_\circ\Big)$
$=$ (by sum manipulation)
$\sum_{j\in[q]} |\phi_j|_\circ + \sum_{k\in[r]} \Big(\sum_{j\in[q]} |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, \phi_j[u_i \leftarrow t_i \mid i \in [p]])|_{y_k}\Big) * |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, \theta_k)|_\circ$
$=$ (by definition of $|\,.\,|_\circ$, substitution, normal form, and definition of $|\,.\,|_{y_k}$)
$|\delta\ \phi_1 \cdots \phi_q|_\circ + \sum_{k\in[r]} |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, (\delta\ \phi_1 \cdots \phi_q)[u_i \leftarrow t_i \mid i \in [p]])|_{y_k} * |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, \theta_k)|_\circ$

If $|\phi|_\circ = |\delta\ \phi_1 \cdots \phi_q|_\circ = 0$ (and thus $|\phi_j|_\circ = 0$ for every $j \in [q]$) or additionally M_1 is context-nondeleting, then the induction hypotheses for the $\phi_1, \ldots, \phi_q$ give equalities instead of inequalities in the above calculation.

$\underline{\phi = f\ u_{i'}\ \phi_1 \cdots \phi_{r'}}$ for some $f \in F^{(r'+1)}$, $u_{i'} \in U_p$, and $\phi_1, \ldots, \phi_{r'} \in RHS(F, \Delta \cup \{\circ, \bullet\}, U_p, Y_r)$:

$|nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, (f\ u_{i'}\ \phi_1 \cdots \phi_{r'})[u_1, \ldots, u_p, y_1, \ldots, y_r \leftarrow t_1, \ldots, t_p, \theta_1, \ldots, \theta_r])|_\circ$
$=$ (by substitution and induction hypothesis (a) for $t_{i'}$)
$\sum_{k'\in[r']} |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, f\ t_{i'}\ y_1 \cdots y_{r'})|_{y_{k'}} * |nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, \phi_{k'}[u_i \leftarrow t_i \mid i \in [p], y_k \leftarrow \theta_k \mid k \in [r]])|_\circ$
$\leq$ (by the induction hypotheses for the $\phi_1, \ldots, \phi_{r'}$,
using that if $|\phi|_\circ = |f\ u_{i'}\ \phi_1 \cdots \phi_{r'}|_\circ = 0$, then also $|\phi_{k'}|_\circ = 0$ for
every $k' \in [r']$, and that the factors $|nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet}}, f\ t_{i'}\ y_1 \cdots y_{r'})|_{y_{k'}} \geq 0$)

$$\sum_{k'\in[r']} |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t_{i'}\ y_1\cdots y_{r'})|_{y_{k'}} * \Big(|\phi_{k'}|_\circ + \sum_{k\in[r]} |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi_{k'}[u_i \leftarrow t_i \mid i\in[p]])|_{y_k} * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \theta_k)|_\circ\Big)$$

$=$ (by sum manipulation)

$$\sum_{k'\in[r']} |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t_{i'}\ y_1\cdots y_{r'})|_{y_{k'}} * |\phi_{k'}|_\circ + \sum_{k\in[r]} \Big(\sum_{k'\in[r']} |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t_{i'}\ y_1\cdots y_{r'})|_{y_{k'}} * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi_{k'}[u_i \leftarrow t_i \mid i\in[p]])|_{y_k}\Big) * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \theta_k)|_\circ$$

$\leq$ (by $|nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t_{i'}\ y_1\cdots y_{r'})|_{y_{k'}} * |\phi_{k'}|_\circ \leq |\phi_{k'}|_\circ$ for every $k'\in[r']$, see below)

$$\sum_{k'\in[r']} |\phi_{k'}|_\circ + \sum_{k\in[r]} \Big(\sum_{k'\in[r']} |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t_{i'}\ y_1\cdots y_{r'})|_{y_{k'}} * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \phi_{k'}[u_i \leftarrow t_i \mid i\in[p]])|_{y_k}\Big) * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \theta_k)|_\circ$$

$=$ (by definition of $|\,.\,|_\circ$, substitution, and Lemma 2.8 for the mtt $M_1^{\to\circ\bullet}$)

$$|f\ u_{i'}\ \phi_1\cdots\phi_{r'}|_\circ + \sum_{k\in[r]} |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, (f\ u_{i'}\ \phi_1\cdots\phi_{r'})[u_i \leftarrow t_i \mid i\in[p]])|_{y_k} * |nf(\Rightarrow_{R_1^{\to\circ\bullet}}, \theta_k)|_\circ$$

The statement, used above, that for every $k'\in[r']$,

$$|nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t_{i'}\ y_1\cdots y_{r'})|_{y_{k'}} * |\phi_{k'}|_\circ \leq |\phi_{k'}|_\circ\,,$$

follows from the precondition that $|\phi|_\circ = |f\ u_{i'}\ \phi_1\cdots\phi_{r'}|_\circ = 0$ or M_1, and hence also $M_1^{\to\circ\bullet}$, is context-linear, since in the former case $|\phi_{k'}|_\circ = 0$, while in the latter case,

$$|nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t_{i'}\ y_1\cdots y_{r'})|_{y_{k'}} \leq 1$$

by Lemma 2.9(1) for $M_1^{\to\circ\bullet}$ (and obviously $|\phi_{k'}|_\circ \geq 0$).

If $|\phi|_\circ = |f\ u_{i'}\ \phi_1\cdots\phi_{r'}|_\circ = 0$ (and thus $|\phi_{k'}|_\circ = 0$ for every $k'\in[r']$) or additionally M_1 (and hence also $M_1^{\to\circ\bullet}$) is context-nondeleting, then the induction hypotheses for the $\phi_1,\ldots,\phi_{r'}$ give equalities instead of inequalities, and also the other occurrences of $\leq$ above can be replaced by $=$ because for context-nondeleting $M_1^{\to\circ\bullet}$,

$$|nf(\Rightarrow_{R_1^{\to\circ\bullet}}, f\ t_{i'}\ y_1\cdots y_{r'})|_{y_{k'}} \geq 1$$

holds by Lemma 2.9(2). □

Proof of Lemma 4.38

For atmost M_2, we take the statements (a) and (b) and the simultaneous induction establishing their validity in the proof of Lemma 4.32. We replace all occurrences of $\Rightarrow^*_{\{\bullet\ v\to v\}}$, $|\,.\,|_\bullet$, and $|\,.\,|_{\bullet\multimap\circ}$ by $\Rightarrow^*_{\{\circ\ v\to v\}}$, $|\,.\,|_\circ$, and $|\,.\,|_\circ$, respectively, and all occurrences of $\bullet\ \theta_1$ by $\circ\ \theta_1$ in the first case of **(a)** $\Leftarrow$ **(b)**. This yields again a valid proof. We obtain

$$|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\theta)|_\circ - |\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\theta[\circ \twoheadleftarrow id])|_\circ \leq |e_{M_2}|_{v_1} * |\theta|_\circ$$

from the thus adjusted statement (b) with $q = 1$, $\theta_1 = \theta$, $\theta'_1 = \theta[\circ \twoheadleftarrow id]$, $s = 0$, and $\psi = e_{M_2}$, taking into account that M_2 is context-linear or M_2 is basic, in which case e_{M_2} contains no nested calls, and that an empty sum is 0, and so is $|\theta[\circ \twoheadleftarrow id]|_\circ$. The first statement of Lemma 4.38 then follows because $|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\theta[\circ \twoheadleftarrow id])|_\circ = 0$ by Lemma 4.37. For atleast M_2, we can replace all occurrences of $\leq$ and "-linear" by $\geq$ and "-nondeleting", respectively, in the adjusted statements and proof, again without compromising validity, and correspondingly (using that an atleast mtt is context-nondeleting or basic) obtain $|\tau_{M_2^{\circ\bullet\to\circ\bullet}}(\theta)|_\circ \geq |e_{M_2}|_{v_1} * |\theta|_\circ$. □

Proof of Lemma 5.4
Let $\phi \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$ and $d \in \mathbb{Z}$.

1. $clean_{\mathcal{E}}(\bullet\ \phi, d)$
   iff (by Definition 5.1)
   $\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).$ $\begin{cases} \bullet\ \phi \Rightarrow^*_{\mathcal{E}} \bullet^d\ \phi' & \text{if } d > 0 \\ \bullet^{-d+1}\ \phi \Rightarrow^*_{\mathcal{E}} \phi' & \text{otherwise.} \end{cases}$
   iff (by Lemma 5.2)
   $\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).$ $\begin{cases} \phi \Rightarrow^*_{\mathcal{E}} \bullet^{d-1}\ \phi' & \text{if } d > 0 \\ \bullet^{-d+1}\ \phi \Rightarrow^*_{\mathcal{E}} \phi' & \text{otherwise.} \end{cases}$
   iff (since $\phi \Rightarrow^*_{\mathcal{E}} \bullet^{1-1}\ \phi'$ iff $\bullet^{-1+1}\ \phi \Rightarrow^*_{\mathcal{E}} \phi'$)
   $\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).$ $\begin{cases} \phi \Rightarrow^*_{\mathcal{E}} \bullet^{d-1}\ \phi' & \text{if } d > 1 \\ \bullet^{-d+1}\ \phi \Rightarrow^*_{\mathcal{E}} \phi' & \text{otherwise.} \end{cases}$
   iff (by Definition 5.1)
   $clean_{\mathcal{E}}(\phi, d-1)$

2. $clean_{\mathcal{E}}(\circ\ \phi, d)$
   iff (by Definition 5.1)
   $\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).$ $\begin{cases} \circ\ \phi \Rightarrow^*_{\mathcal{E}} \bullet^d\ \phi' & \text{if } d > 0 \\ \bullet^{-d}\ (\circ\ \phi) \Rightarrow^*_{\mathcal{E}} \phi' & \text{otherwise.} \end{cases}$
   iff (by Lemma 5.3(1))
   $\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).$ $\begin{cases} \phi \Rightarrow^*_{\mathcal{E}} \bullet^{d+1}\ \phi' & \text{if } d > 0 \\ \bullet^{-d}\ (\circ\ \phi) \Rightarrow^*_{\mathcal{E}} \phi' & \text{otherwise.} \end{cases}$
   iff (since by Lemma 5.3(1) for every $\phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y)$:
   $\bullet^{-0}\ (\circ\ \phi) \Rightarrow^*_{\mathcal{E}} \phi'$ iff $\phi \Rightarrow^*_{\mathcal{E}} \bullet^{0+1}\ \phi'$)
   $\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).$ $\begin{cases} \phi \Rightarrow^*_{\mathcal{E}} \bullet^{d+1}\ \phi' & \text{if } d > -1 \\ \bullet^{-d}\ (\circ\ \phi) \Rightarrow^*_{\mathcal{E}} \phi' & \text{otherwise.} \end{cases}$
   iff (by Lemma 5.3(2))
   $\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).$ $\begin{cases} \phi \Rightarrow^*_{\mathcal{E}} \bullet^{d+1}\ \phi' & \text{if } d > -1 \\ \bullet^{-d-1}\ \phi \Rightarrow^*_{\mathcal{E}} \phi' & \text{otherwise.} \end{cases}$
   iff (by Definition 5.1)
   $clean_{\mathcal{E}}(\phi, d+1)$ □

Index

$[n]$ (for some natural number), 9
$\blacksquare[\ldots \leftarrow \ldots]$, 11
$\blacksquare[\ldots \twoheadleftarrow \ldots]$, 11
$|\blacksquare|_s$ (for some symbol), 10–11
$|\blacksquare|_{\circ-\bullet}$, 47
$|\blacksquare|_{\bullet-\circ}$, 56
$\Rightarrow^?$ and $\Rightarrow^*$ (for some binary relation), 9
$\Rightarrow_R$ (for some rewrite system), 12
$\overset{cbn}{\Rightarrow}_{R_1 \cup R_2}$ (for rule sets of two mtts), 31
$\overset{cbn}{\Rightarrow}_{\overline{R_1;R_2}}$ (for rule sets of two mtts), 30
$\downarrow_{\delta,j}$, $\uparrow_{\delta,j}$, $\downarrow_{f,k}$, and $\uparrow_{f,k}$, 53
$\rightsquigarrow_{\mathcal{E}_{M_1,M_2}}$ (for two mtts), 55

accumulating parameter, 3, 111, 117
atleast, 15
atmost, 15
att, *see* attributed tree transducer
attribute grammar, 4, 92
attributed tree transducer, 4, 92–93

basic, 14

call, 12
call-by-name, 30–33
call-by-need, 64–67
cata/*build*, 108–110
catamorphism, 109–111
$cbn_{M_1;M_2}$ (for two mtts), 33
$cbn_{\overline{M_1;M_2}}$ (for two mtts), 33
classical deforestation, *see* deforestation
$clean_{\mathcal{E}}$, 70
context parameter, 4, 12
context variable, 12
context-linear, 15, 63
context-nondeleting, 15

deaccumulation, 117–119
decomposition, 91, 117
deforestation, 3
  classical, 98–103
  shortcut, 108–112
destroy/*unfoldr*, 110–111

ε, 9
$\mathcal{E}_{M_1,M_2}$ (for two mtts), 54

finite copying, 63, 93
Fold, 98
free theorem, 3, 108, 111–112

Haskell, 77–89, 105, 108–110, 112
height, 10–11
high-level tree transducer, 91–92, 96
homomorphism tree transducer, 14
htt, *see* homomorphism tree transducer

id, 9
induction, 15–16, 115–119
infinite tree, 77–83
initial expression, 12

$\kappa_{f,k}$ and $\overline{\kappa}_{f,k}$, 50

lab, 10
lazy composition, 79, 93, 104–108
linear, 15

μ, 21
$\overline{M_1;M_2}$ (for two mtts), 22
$\overline{M_1;M_2}^{\rightarrow\circ}$, 38
$\overline{M_1;M_2}^{\perp}$, 81
$M_1^{\rightarrow\diamond}$, 35
$M_1^{\rightarrow\diamond\star}$, 40
$M_1^{\rightarrow\star\bullet}$, 46
$M_1^{\rightarrow\circ\bullet,\kappa}$, 50

$\mathcal{M}_1^{\to\circ\bullet}$, 55
$M_1^{\perp}$, 81
$M_2^{\diamond\to\bullet}$, 35–36
$M_2^{\diamond\star\to\circ}$, 40
$M_2^{\star\bullet\to\circ\bullet}$, 46
$M_2^{\circ\bullet\to\circ\bullet}$, 50
$M_2^{\perp}$, 81
macro tree transducer, 4, 12
max, 9
modular tree transducer, 111, 117
mtt, *see* macro tree transducer

$\mathbb{N}$ and $\mathbb{N}_+$, 9
$nest_f$ (for some state), 23
$nf(\Rightarrow, \ldots)$, 9
nondeleting, 15

$\mathcal{P}$, 9
Pair, 22
$par_{M_2,\phi}$ (for some mtt, expression), 23–24
partial demand, 84–88, 90, 102, 114–115
partial tree, 79–83
paths, 10
polymorphism, 109–112
  rank-2, 109
Pre, 22
producing, 15
proper subtree, 16

$\overline{R_1;R_2}$ (for rule sets of two mtts), 22
$\overline{R_1;R_2}^D$ (for rule sets of two mtts), 98
rank, 10
ranked alphabet, 10
recursion argument, 12
recursion variable, 12
recursion-linear, 15
recursion-nondeleting, 15
reduction relation, 12
rewrite system, 11
$rhs_{M,f,\sigma}$ (for some mtt, state, symbol), 12
$rhs_{\overline{M_1;M_2}^D,\overline{fg},\sigma}$, 98
$RHS(\ldots)$, 12
rule,
  $\uparrow$- or $\downarrow$-, 53
  rewrite, 11
  σ- (for some symbol), 12

s_{max}, 50
shortcut deforestation, *see* deforestation
size, 10
state, 12
sub, 10
$sup_{\mathcal{E}}$, 74
superfluous context parameter, 26
symbol,
  input or output, 12
  nullary, unary, or binary, 10
  rank of a, 10
  root, 11

T_Σ and $T_\Sigma(X)$ (for a ranked alphabet and a set of variables), 10
τ_M (for some mtt), 13
tail call, 88–90, 102–103, 114–115
tdtt, *see* top-down tree transducer
top-down tree transducer, 4, 14
tree over a ranked alphabet, 10
tree transduction,
  induced, 13
  macro, 13
tupling, 103–110

unfold/fold, 3, 98, 114

vanish, 111–112

weakly single-use, 14, 92
well-formed term
  for an mtt, 13
  for the original program, 31
  without duplicable calls, 64
WFT_M (for some mtt), 13
$WFT_{M_1;M_2}$ (for two mtts), 31
WFT'_{M_1} (for some mtt), 64
$WFT'_{M_1;M_2}$ (for two mtts), 64

$\mathbb{Z}$, 9