# Internship Report

# Compositions of Extended Top-down Tree Transducers

Aurélie Lagoutte[⋆]
Supervised by Andreas Maletti

Internship carried out at
Universität Stuttgart, Institute for Natural Language Processing
Azenbergstraße 12, 70174 Stuttgart, Germany

# Introduction

## 1 Motivation

Many aspects of machine translation of natural languages can be formalized by employing weighted finite-state (string) transducers [22, 40]. Successful implementations based on this word- or phrase-based approach are, for example, the AT&T FSM toolkit [41], XEROX's finite-state calculus [24], the RWTH toolkit [23], CARMEL [19], and OPENFST [2]. However, the phrase-based approach is not expressive enough, for example, to easily handle the rotation needed in the translation of the English structure NP-V-NP (subject-verb-noun phrase) to the Arabic structure V-NP-NP. A finite-state transducer can only implement this rotation by storing the subject, which might be very long, in its finite memory.

Syntax-based (or tree-based) formalisms can remedy this shortage. An example of such formalisms is the top-down tree transducer [42, 43], of which a weighted version is implemented in the toolkit TIBURON [38], together with some standard operations. Those weighted top-down tree transducers [29, 14, 17] (also called 'tree series transducers') are a joint generalization of the unweighted top-down tree transducer (tdtt) [42, 43] and the weighted tree automaton [7, 10, 1, 27, 16, 9, 8].

During my internship, I investigated compositions of weighted and unweighted extended top-down tree transducers. An unweighted tree transducer computes a relation $\tau$ between input and output trees, and a weighted tree transducer computes a weighted relation between input and output trees (i.e., it assigns a weight to each pair of input and output trees). An unweighted tree transducer can be seen as a weighted tree transducer, where the weighted relation assigns *true* (resp. *false*) to a pair of trees if this pair is (resp. is not) in the relation $\tau$. Using the real numbers as weight structure, we compose two weighted relations $\tau_1\colon A \times B \to \mathbb{R}$ and $\tau_2\colon B \times C \to \mathbb{R}$ by requiring that

$$(\tau_1 \,;\, \tau_2)(a, c) = \sum_{b \in B} \tau_1(a, b) \cdot \tau_2(b, c) \tag{1}$$

for all $a \in A$ and $c \in C$. For the sake of simplicity, let us assume that $B$ is finite. Equation (1) can be imagined in an operational manner. The first process transforms the input $a$ into an intermediate product $b$ at a certain cost $\tau_1(a, b)$. This intermediate product is then fed into the second process, which transforms it into a final product at cost $\tau_2(b, c)$. Thus, the components are executed in a sequential manner. Traditionally, the multiplicative operation of the weight structure (typically, a semiring [21, 18]) is used to combine weights of processes that are executed in series (or in sequence). Consequently, we multiply the weights $\tau_1(a, b) \cdot \tau_2(b, c)$ to obtain the cost of producing $c$ from $a$ via the intermediate product $b$. Naturally, there might be a choice of intermediate products that are all suitable to some degree to produce the output $c$. Thus, we sum over all the possibilities of producing $c$ from $a$.

[⋆] École normale supérieure de Cachan, Département Informatique – France

**Table 1.** Composition results for unweighted devices.

| Case | tdtt $M$ | tdtt $N$ | Reference |
|---|---|---|---|
| (a) | | linear and nondeleting | [6, Theorem 1] |
| (b) | total | linear | [6, Theorem 1] |
| (c) | deterministic | nondeleting | [6, Theorem 1] |
| (d) | deterministic and total | | [6, Theorem 1] |
| Case | xtt $M$ in 1-symbol normal form | $N$ tdtt with $\varepsilon$-rules | Reference |
| (a) | | linear and nondeleting | [37, Theorem 17] |
| (b) | total | linear | [37, Theorem 17] |

**Table 2.** Composition results for weighted tdtt ('nondel.' abbreviates 'nondeleting' and 'det.' abbreviates 'deterministic').

| Case | $M$ | $N$ | Reference |
|---|---|---|---|
| (a) | linear and nondel. | linear and nondel. | [28, Theorem 2.4] |
| | deterministic | det., linear, nondel. | [14, Theorem 5.18] |
| | | linear and nondel. | [31, Theorem 26] |
| (d) | Boolean, det., total | deterministic | [14, Theorem 5.18] |
| | Boolean, det., total | linear | [31, Theorem 30] |

Compositions have been and are used in a number of application areas as diverse as machine translation [44] and functional program optimization [26]. The complexity of a given tree transformation problem can be tackled and broken down into smaller pieces with the help of (de-) composition in a *divide-and-conquer* approach. Once all the subproblems have been solved, the individual solutions can be recombined with the help of composition. This approach is used in [44], where a translation model is broken into 3 smaller pieces:

– a reordering component, which changes the order of subtrees but keeps the trees otherwise intact,
– an insertion component, which has the ability to spontaneously add subtrees to the output of the translation, and
– a translation component, which just translates the words (or phrases) occuring in the input tree.

These components can now be trained and optimized individually (even from different resources). However, since the evaluation of composition chains can be very inefficient [39], automatic procedures that "compose" finite representations of such weighted relations are desirable. Naturally, the obtained finite representation should compute the composition of the weighted relations computed by the input representations. As expected, the finite representation discussed in this report is the weighted and unweighted extended top-down tree transducer (xtt) together with its variants.

It is known that already in the unweighted setting, this cannot always be achieved [4]. Moreover, composition of top-down tree transducers has been investigated in the unweighted case by [12, 6] and these results were partially lifted to the weighted setting in [14, 31, 32]. Another important case are the compositions of selected xtt with top-down tree transducers that can additionally have $\varepsilon$-rules ($\varepsilon$tdtt), which have been investigated in [37]. Tables 1 and 2 summarize the relevant results.

## 2 General context of the internship

My internship was supervised by Andreas Maletti at the Institute for Natural Language Processing in Stuttgart. I was part of the project "Tree Transducers in Machine Translation" together with Andreas and three PhD students: Fabienne Braune, Nina Seemann and Daniel Quernheim. I would like to thank the whole Institute, and more specifically the team, for their warm welcome and their help during my stay in Stuttgart.

My contribution is split into two parts. The first part was in cooperation with the PhD students, and deals with unweighted extended top-down tree transducer compositions. The initial aim was to find a new composition construction that can compose two linear and nondeleting xtt (nl-xtt) $M$ and $N$ fulfilling some requirements into a single nl-xtt. Allowing $N$ to be an xtt rather than a tdtt or an $\varepsilon$tdtt is the first difference compared to the results existing so far (see Tables 1 and 2), and triggers additional problems. Further, as it is already known that the subclass of transformations computed by nl-xtt is not closed under composition [35, Example 3.1], the ideal target is a decision procedure that, given two nl-xtt, decides whether the composition is again computable by such a device. Such a decision procedure is desirable in the application domain of machine translation to check whether features that make composition impossible actually occur. Part I presents the results for unweighted xtt obtained during my internship, namely a partial decision procedure that takes two nl-xtt as input, tries to compose them, and either outputs the composed nl-xtt, or fails when it does not manage to compute a composition into a single nl-xtt.

Secondly, I co-authored with Andreas a survey about compositions of weighted extended top-down tree transducers, which is to appear as [30]. Part II presents my contribution, which was mostly adapting the composition results for unweighted $\varepsilon$tdtt from [37] to the weighted case. First I formalized a new model of xtt, which is equivalent to the usual one, as shown in Section 1. Then I generalized the existing results using the new notion of xtt in Section 2. Lastly, I built relevant examples to illustrate the different constructions. The full survey can be found in Appendix C.

## 3 Notation

The set of all nonnegative integers is $\mathbb{N}$, and we let $[n] = \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$ for every $n \in \mathbb{N}$. We fix the set $\mathrm{X} = \{\mathrm{x}_i \mid i \in \mathbb{N}\}$ of (formal) variables. The set of all finite words (or sequences) over a set $S$ is $S^*$, where $\varepsilon$ is the empty word. The concatenation of the words $v, w \in S^*$ is $v.w$ or simply $vw$. The length of a word $w \in S^*$ is denoted by $|w|$. An *alphabet* $\Sigma$ is a nonempty and finite set, of which the elements are called *symbols*. For every alphabet $Q$, we let $Q(S) = \{q(s) \mid q \in Q, s \in S\}$. The set $T_\Sigma(S)$ of *$\Sigma$-trees*[1] *with leaf labels* $S$ is the smallest set $U$ such that $S \subseteq U$ and $\sigma(u_1, \ldots, u_k) \in U$ for every $k \in \mathbb{N}$, $\sigma \in \Sigma$, and $u_1, \ldots, u_k \in U$. We denote by $C_\Sigma(V) \subseteq T_\Sigma(\{x_1\} \cup V)$ the set of all contexts over $\Sigma$ indexed by $V$, which are $\Sigma$-trees indexed by $\{x_1\} \cup V$ such that the variable $x_1$ occurs exactly once. We often omit qualifications like 'for all $k \in \mathbb{N}$' if it is obvious from the context that $k$ is a nonnegative integer. Moreover, we generally assume that $\Sigma \cap S = \emptyset$, and thus we write $\sigma()$ simply as $\sigma$ for every $\sigma \in \Sigma$. Given another alphabet $\Delta$, we treat elements of $T_\Delta(T_\Sigma(S))$ and $Q(T_\Sigma(S))$ as particular trees of $T_{Q \cup \Sigma \cup \Delta}(S)$.[2] Finally, we write $T_\Sigma$ (resp. $C_\Sigma$) for $T_\Sigma(\emptyset)$ (resp. $C_\Sigma(\emptyset)$).

Next, we define a few operations on trees. The set $\mathrm{pos}(t) \subseteq \mathbb{N}^*$ of *positions* of a tree $t \in T_\Sigma(S)$ is inductively defined by $\mathrm{pos}(s) = \{\varepsilon\}$ for every $s \in S$ and

$$\mathrm{pos}(\sigma(t_1, \ldots, t_k)) = \{\varepsilon\} \cup \{i.w \mid i \in [k], w \in \mathrm{pos}(t_i)\}$$

for every $\sigma \in \Sigma$ and $t_1, \ldots, t_k \in T_\Sigma(S)$. The set $\mathrm{pos}(t)$ of positions is (totally) ordered by the lexicographic order on $\mathbb{N}^*$. Let $t, t' \in T_\Sigma(S)$ and $w \in \mathrm{pos}(t)$. The *label* of $t$ at position $w$ is $t(w)$, and the *$w$-rooted subtree* of $t$ is $t|_w$. Formally, these notions can be defined by $s(\varepsilon) = s|_\varepsilon = s$ for every $s \in S$ and

$$\sigma(t_1, \ldots, t_k)(\varepsilon) = \sigma \qquad \sigma(t_1, \ldots, t_k)(i.v) = t_i(v)$$
$$\sigma(t_1, \ldots, t_k)|_\varepsilon = \sigma(t_1, \ldots, t_k) \qquad \sigma(t_1, \ldots, t_k)|_{i.v} = t_i|_v$$

for every $\sigma \in \Sigma$, $t_1, \ldots, t_k \in T_\Sigma(S)$, $i \in [k]$, and $v \in \mathrm{pos}(t_i)$. For every subset $L \subseteq \Sigma \cup S$ of labels and $s \in S$, we let $\mathrm{pos}_L(t) = \{w \in \mathrm{pos}(t) \mid t(w) \in L\}$ and $\mathrm{pos}_s(t) = \mathrm{pos}_{\{s\}}(t)$. The expression $t[u_1, \ldots, u_n]_{w_1, \ldots, w_n}$ denotes the tree that is obtained from $t \in T_\Sigma(S)$ by replacing for all $i \in [n]$ the subtree $t|_{w_i}$ at position $w_i$ by $u_i \in T_\Sigma(S)$.

[1] These are actually unranked trees, but our operational tree transformation model will only have finitely many rules that prescribe (and limit) the ranks of symbols, so that we could have used a ranked alphabet as well.

[2] A benefit of our approach without explicit ranks for symbols is that we can always take the union of two alphabets.

The following operations implicitly always use the fixed set X of variables. We let $\mathrm{var}(t)$ be the set $\{x \in \mathrm{X} \mid \mathrm{pos}_x(t) \neq \emptyset\}$. The tree $t$ is *linear* if every $x \in \mathrm{X}$ occurs at most once in $t$. A *substitution* $\theta \colon \mathrm{X} \to T_\Sigma(S)$ can be applied to a tree $t \in T_\Sigma(S)$, and returns the tree $t\theta$ that is obtained by replacing (in parallel) all occurrences of each variable $x \in \mathrm{X}$ by $\theta(x)$. Formally, (i) $x\theta = \theta(x)$ for every $x \in \mathrm{X}$, (ii) $s\theta = s$ for every $s \in S \setminus \mathrm{X}$, and (iii) $\sigma(t_1, \ldots, t_k)\theta = \sigma(t_1\theta, \ldots, t_k\theta)$ for every $\sigma \in \Sigma$ and $t_1, \ldots, t_k \in T_\Sigma(S)$. If $C \in C_\Sigma$, then we also write $C[\theta(x_1)]$ instead of $C\theta$. A tree $t \in T_\Sigma$ is an *instance* of $C \in C_\Sigma(\mathrm{X})$ if there exists $\theta \colon \mathrm{X} \to T_\Sigma$ such that $t = C\theta$.

A *(commutative) semiring* [21, 18] is an algebraic structure $(A, +, \cdot, 0, 1)$ consisting of two commutative monoids $(A, +, 0)$ and $(A, \cdot, 1)$ such that $\cdot$ distributes over finite sums (including the empty sum, which yields $a \cdot 0 = 0$ for all $a \in A$). The semiring $A$ is *idempotent* if $1 + 1 = 1$ (by distributivity, this yields $a + a = a$ for all $a \in A$). Examples of semirings include

- the BOOLEAN semiring $(\{0, 1\}, \max, \min, 0, 1)$, which is idempotent,
- the powerset[3] semiring $(\mathcal{P}(S), \cup, \cap, \emptyset, S)$ for some set $S$, which is idempotent,
- the tropical semiring $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$, which is also idempotent,
- the nonnegative integers $(\mathbb{N}, +, \cdot, 0, 1)$, and
- the semiring of real numbers $(\mathbb{R}, +, \cdot, 0, 1)$.

Let $S$ and $T$ be sets, and let $(A, +, \cdot, 0, 1)$ be a semiring. A *weighted relation $r$ from $S$ to $T$* is a mapping $r \colon S \times T \to A$. Moreover, for every mapping $f \colon S \to A$, we let $\mathrm{supp}(f) = \{s \in S \mid f(s) \neq 0\}$. Thus, $\mathrm{supp}(r) \subseteq S \times T$.

*From now on, let $(A, +, \cdot, 0, 1)$ be an arbitrary semiring such that $0 \neq 1$.*

Next, let us recall the weighted extended (top-down) tree transducer [11, 3, 25, 20]. We essentially follow the definitions of [33, 36], in which the corresponding unweighted device is discussed in detail. A *(weighted) extended (top-down) tree transducer* (xtt) is a tuple $(Q, \Sigma, \Delta, I, R)$, where

- $Q$ is a finite set of *states*,
- $\Sigma$ and $\Delta$ are alphabets of *input* and *output symbols* such that $Q \cap (\Sigma \cup \Delta) = \emptyset$,
- $I \subseteq Q$ is a set of *initial states*, and
- $R \colon Q(T_\Sigma(\mathrm{X})) \times T_\Delta(Q(\mathrm{X})) \to A$ assigns *rule weights* such that $\mathrm{supp}(R)$ is finite and for every $(l, r) \in \mathrm{supp}(R)$ we have that $\{l, r\} \not\subseteq Q(\mathrm{X})$, $l$ is linear, and $\mathrm{var}(r) \subseteq \mathrm{var}(l)$.[4]

For the following discussion, let $M = (Q, \Sigma, \Delta, I, R)$ be an xtt. The elements of $\mathrm{supp}(R)$ are called *rules* (of $M$), and we often write them as $l \to r$ instead of $(l, r)$. We call $l$ and $r$ of a rule $l \to r$ the *left-* and *right-hand side*, respectively. Moreover, we write $l \to r \in R$ instead of $(l, r) \in \mathrm{supp}(R)$, and we write $l \xrightarrow{a} r \in R$ instead of $R(l, r) = a$. A rule $l \to r \in R$ is (i) *linear* if $r$ is linear, (ii) *nondeleting* if $\mathrm{var}(r) = \mathrm{var}(l)$, and (iii) *simple* if $|\mathrm{pos}_\Sigma(l)| = 1$. In addition, the rule $l \to r$ is

- *consuming* if $|\mathrm{pos}_\Sigma(l)| \geq 1$, and an *$\varepsilon$-rule* otherwise, and
- *producing* if $|\mathrm{pos}_\Delta(r)| \geq 1$, and *erasing* otherwise.[5]

The xtt $M$ is (i) linear, (ii) nondeleting, and (iii) a top-down tree transducer [29, 14] (tdtt) if every rule $l \to r \in R$ is (i) linear, (ii) nondeleting, and (iii) simple, respectively. An xtt that is not linear may also be called *copying*. Moreover, the xtt $M$ is BOOLEAN if $R(l, r) = 1$ for every $l \to r \in R$.

The semantics of the xtt $M$ is given by term rewriting [13, 5, 36]. To simplify our composition constructions later on, we immediately present a semantics that can handle "foreign" symbols, which are symbols that are not in $Q \cup \Sigma \cup \Delta$.Let $\Sigma'$ and $\Delta'$ be such that $\Sigma \subseteq \Sigma'$ and $\Delta \subseteq \Delta'$. The elements of $T_{\Delta'}(Q(T_{\Sigma'}(\mathrm{X})))$ are called *sentential forms*. A position $w \in \mathrm{pos}_Q(\xi)$ in a sentential form $\xi$ is *reducible (for $M$)* if there exists a rule $l \to r \in R$ and a substitution $\theta \colon \mathrm{X} \to T_{\Sigma'}(\mathrm{X})$ such that $\xi|_w = l\theta$. Let $\xi, \zeta \in T_{\Delta'}(Q(T_{\Sigma'}(\mathrm{X})))$ be sentential forms and $l \to r \in R$ be a rule. We say that *$\xi$ rewrites to $\zeta$ using $l \to r$*, denoted by $\xi \Rightarrow_M^{(l,r)} \zeta$, if there exists a substitution $\theta \colon \mathrm{X} \to T_{\Sigma'}(\mathrm{X})$

[3] The powerset $\mathcal{P}(S)$ of a set $S$ is the set of its subsets; i.e., $\mathcal{P}(S) = \{U \mid U \subseteq S\}$.

[4] The restriction $\{l, r\} \not\subseteq Q(\mathrm{X})$, which is not present in [34], disallows rules of the form $(q(\mathrm{x}_i), p(\mathrm{x}_i))$. This additional restriction is necessary because we do require complete semirings [21, 18], which yields that we have to avoid infinite summations.

[5] The name 'erasing' is justified by the fact that each erasing rule is consuming.

such that $\xi|_w = l\theta$ and $\zeta = \xi[r\theta]_w$ where $w$ is the least reducible position in $\mathrm{pos}_Q(\xi)$ with respect to the lexicographic total ordering on $\mathbb{N}^*$.[6] As usual, we use ';' for relation composition, thus for example,

$$(\Rightarrow_M^{\rho_1} ; \Rightarrow_M^{\rho_2}) = \{(\xi, \zeta) \mid \exists \xi' \colon \xi \Rightarrow_M^{\rho_1} \xi' \Rightarrow_M^{\rho_2} \zeta\} \enspace .$$

The *(extended) weighted relation* $\tau'_M$ (or weighted tree transformation) *computed by* $M$ is given by

$$\tau'_M(\xi, \zeta) = \sum_{\substack{\rho_1, \dots, \rho_k \in \mathrm{supp}(R) \\ \xi \Rightarrow_M^{\rho_1} ; \cdots ; \Rightarrow_M^{\rho_k} \zeta}} \Bigl( \prod_{i=1}^{k} R(\rho_i) \Bigr)$$

for every $\xi, \zeta \in T_{\Delta'}(Q(T_{\Sigma'}(\mathrm{X})))$. Then, the *semantics* $\tau_M$ of the xtt $M$ is the weighted relation $\tau_M \colon T_\Sigma \times T_\Delta \to A$ such that $\tau_M(t, u) = \sum_{q \in I} \tau'_M(q(t), u)$ for every $t \in T_\Sigma$ and $u \in T_\Delta$.[7] Finally, the xtt $M$ is *deterministic* (respectively, *total*) if for all $q \in Q$ and $t \in T_\Sigma$ there exists at most (respectively, at least) one $u \in T_\Delta$ such that $(q(t), u) \in \mathrm{supp}(\tau'_M)$.

# Part I - Composition of Unweighted Linear and Nondeleting Xtt

This part is devoted to the composition of linear and nondeleting extended top-down tree transducers. Since it is well-known that this subclass is not closed under composition [35, Example 3.1], we would like to decide, given two nl-xtt, whether the composition of these nl-xtt can be computed by a single nl-xtt. This work provides a partial answer to this question. Indeed, the procedure described in the following, takes two nl-xtt as input, and tries to compose them into an nl-xtt. When the procedure succeeds, it outputs an nl-xtt that computes the composition. Otherwise, it fails, which means that the procedure was not able to compute the composition into a single nl-xtt, but unluckily, this does not mean that this cannot be achieved. However, it is believed that in most cases, the failure of the procedure matches compositions that are outside the subclass. More precisely, the procedure is divided into three parts:

- a pre-processing step, that transforms the first nl-xtt $M_1$ in order to fulfill some syntactic restrictions with respect to the second nl-xtt $M_2$. After this step, the linearity and nondeletion of $M_1$ may be lost but the transducers have an appropriate form for the composition.
- the main step that composes the two xtt. It depends on whether $M_1$ kept its linearity and nondeletion in the preprocessing step.
  - If yes, the procedure can handle recursive erasing: either it computes the nl-xtt that computes the composition, or it fails. The latter case happens when $M_1$ has recursive erasing at deadly or harmless positions.
  - If no, the procedure checks whether recursive erasing may happen in $M_1$. If no, the procedure computes the composition and outputs a xtt with regular look-ahead. If yes, the procedure fails.
- The last step occurs only if $M_1$ has lost its linearity and nondeletion in the preprocessing step. This loss may be purely artificial, so we try to recombine the rules of the xtt output from the main step, in order to get an nl-xtt. Either it succeeds and returns the composed nl-xtt, or it fails.

This approach is summarized in Figure 1. After a first section with some additional preliminaries, each section describes one step of the procedure. Appendix A runs the different steps of the procedure on a example.

[6] Given a sentential form $\xi$ and a rule $\rho \in R$, there exists at most one sentential form $\zeta$ such that $\xi \Rightarrow_M^\rho \zeta$.

[7] Since the xtt $M$ cannot consume symbols from $\Sigma' \setminus \Sigma$ and cannot produce symbols from $\Delta' \setminus \Delta$, the semantics $\tau_M$ does not depend on the particular choice of $\Sigma'$ and $\Delta'$.

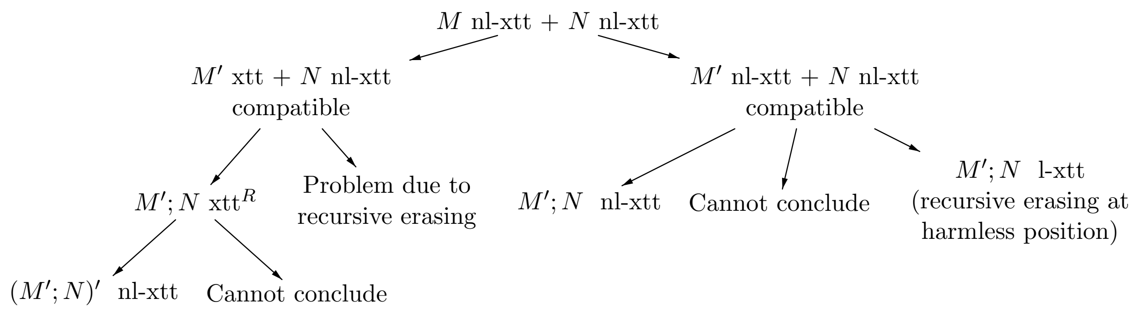

$M$ nl-xtt + $N$ nl-xtt
$M'$ xtt + $N$ nl-xtt compatible
$M'$ nl-xtt + $N$ nl-xtt compatible
$M';N$ xtt$^R$
Problem due to recursive erasing
$M';N$ nl-xtt
Cannot conclude
$M';N$ l-xtt (recursive erasing at harmless position)
$(M';N)'$ nl-xtt
Cannot conclude

**Fig. 1.** Description of the different steps

## 1 Preliminaries

In this part, we use unweighted extended top-down tree transducers with regular look-ahead. As it is slightly different from the notion described in Section 3, we define it in the following way: An *(unweighted) extended (top-down) tree transducer with regular look-ahead* (xtt$^R$) is a tuple $(Q, \Sigma, \Delta, I, R, c)$, where

– $Q$, $\Sigma$, $\Delta$, $I$ are *states*, *input* and *output alphabet*, and *initial states* just in the same way as in the previous device,
– $R \subseteq Q(T_\Sigma(\mathrm{X})) \times T_\Delta(Q(\mathrm{X}))$ is a finite set of *rules* such that $l$ is linear in X and $\mathrm{var}(r) \subseteq \mathrm{var}(l)$ for every $l \to r \in R$, and
– $c\colon R \to \mathrm{Rec}(\Sigma)$ (the *look-ahead*), where $\mathrm{Rec}(\Sigma)$ is the set of recognizable languages over $T_\Sigma$.

The semantics is given by rewriting in almost the same way as before: for every $\xi, \zeta \in T_\Delta(Q(T_\Sigma))$, we say that $\xi$ *rewrites into* $\zeta$, denoted by $\xi \Rightarrow_M$ if there exists a position $w \in \mathrm{pos}(\xi)$, a rule $l \to r \in R$ and a substitution $\theta\colon \mathrm{X} \to T_\Sigma$ such that (i) $\xi|_w = l\theta$, (ii) $\xi|_{w1} \in c(l \to r)$ and (iii) $\zeta = \xi[r\theta]_w$. The *tree transformation computed by M*, denoted by $\tau_M$, is the relation

$$\tau_M = \{(t,u) \in T_\Sigma \times T_\Delta \mid \exists q \in I\colon q(t) \Rightarrow_M^* u\}$$

where $\Rightarrow_M^*$ denotes the reflexive, transitive closure of $\Rightarrow_M$. The properties of rules and xtt defined in Section 3 generalize straightforwardly to unweighted xtt$^R$.

In the following, only finite look-ahead will be needed, which means that for all $r \in R$, there exists $C_r \in C_\Sigma(X)$ such that $c(r)$ is the regular language composed of all instances of $C_r$. We will not distinguish between $c(r)$ and $C_r$.

From now on, let $M_1 = (Q_1, \Sigma, \Delta, I_1, R_1)$ and $M_2 = (Q_2, \Delta, \Gamma, I_2, R_2)$ be nl-xtt. Since the subclass of relations computable by nl-xtt is equal to the subclass of nl-xtt$^R$, we drop the look-aheads. Without loss of generality, assume two disjoint sets $X_1, X_2 \subseteq \mathrm{X}$ of variables such that $R_1 \subseteq Q_1(T_\Sigma(X_1)) \times T_\Delta(Q_1(X_1))$ and $R_2 \subseteq Q_2(T_\Delta(X_2)) \times T_\Gamma(Q_2(X_2))$, and assume that $Q_1 \cap Q_2 = \emptyset$.

## 2 The compatibility criterion

### 2.1 Definition

In the existing composition results on tdtt, $M_2$ has at most one symbol in its left-hand sides, which enables us to match this symbol with the positions where it occurs in the arbitrarily large right-hand sides of $M_1$. However, here $M_2$ is an extended top-down tree transducer, and can consequently have arbitrarily large left-hand sides. The first idea to solve this problem is to enforce $M_1$ to have only one symbol in its right-hand sides, so that we can match this symbol with the positions where it occurs in the left-hand sides of $M_2$. This so called *one-symbol normal form* is described in [37, Definition 6], as well as a procedure [37, Theorem 11] to put any nl-xtt into an equivalent xtt in one-symbol normal form. However, the procedure triggers in a lot of cases the loss of linearity and nondeletion. Moreover, when the same pattern occurs both in the right-hand side of $M_1$ and in the left-hand side of $M_2$, there is no need to separate the symbols occurring in the pattern. These two remarks lead us to a less restrictive criterion than the one-symbol normal form, that still enables us to do the composition properly.

**Definition 1.** *$M_1$ is* compatible *with $M_2$ if for all $p(l_2) \to s_2 \in R_2$, for all $w \in \mathrm{pos}_\Delta(l_2)$, for all $q(l_1) \to s_1 \in R_1$, one of the following conditions is fulfilled:*

– *$\exists w' \in \mathrm{pos}_\Delta(s_1) \cap \mathrm{pos}_\Delta(l_2|_w)$ such that $s_1(w') \neq l_2|_w(w')$*
– *$\exists t_1, \dots, t_k \in T_\Delta(X_2)$ such that $l_2|_w = s_1[t_1, \dots, t_k]_{w_1,\dots,w_k}$ with $\{w_1, \dots, w_k\} = \mathrm{pos}_{Q_1}(s_1)$.*

Intuitively, for every position $w$ of a $\Delta$ symbol $\sigma$ in a left-hand side $l_2$ of $M_2$, for any right-hand side $s_1$ of $M_1$ starting from the symbol $\sigma$, either $s_1$ and $l_2|_w$ differ from a $\Delta$ symbol, or the whole pattern $s_1$ occurs in $l_2|_w$.

*Remark 2.* An xtt $M_1$ in one-symbol normal form is compatible with any xtt $M_2$.

## 2.2 A procedure to get compatible xtt

This subsection shows a procedure that, given two xtt $M_1$ and $M_2$, modifies the rules and the states of $M_1$ to output an xtt that is compatible with $M_2$.

1. If $M_1$ is compatible with $M_2$, we are done.
2. Otherwise, there exists $l_1 \to s_1 \in R_1, l_2 \to s_2 \in R_2, t_1, \dots, t_k \in T_\Delta(X_2)$, $\{z_1, \dots, z_k\} = \mathrm{pos}_{Q_1}(s_1)$, $y_1, \dots, y_l \in X_2$, $w_1, \dots, w_l \in \mathrm{pos}_\Delta(s_1)$, a context $C \in C_{Q_2 \cup \Delta \cup X_2}$ such that
$$l_2 = C[(s_1[t_1, \dots, t_k]_{z_1,\dots,z_k})[y_1, \dots, y_l]_{w_1,\dots,w_l}]$$
Intuitively, we can obtain $l_2$ by plugging $s_1$ into a context, plugging any tree of $T_\Delta(X_2)$ instead of every $q(x)$, $Q \in Q_1$, $x \in X_1$, and then cut at positions $w_1, \dots, w_l$ and replace the new leaves by variables of $X_2$.
Let $w \in \{w_1, \dots, w_l\}$.
   – If $\mathrm{var}(s_1|_w) \neq \emptyset$, let $w'$ be the longest common prefix of $\mathrm{pos}_{\mathrm{var}(s_1|_w)}(l_1)$.
   – If $\mathrm{var}(s_1|_w) = \emptyset$, and there is a leaf in $l_1$ with a $\Delta$ symbol, let $w'$ be the position of this leaf.
   – Otherwise, $\mathrm{var}(s_1|_w) = \emptyset$, and there is no leaf in $l_1$ with a $\Delta$ symbol. Let $w' = \varepsilon$.
Let $A = \mathrm{var}(l_1|_{w'}) \setminus \mathrm{var}(s_1|_w)$, $q'$ a new state disjoint from $Q_1 \cup Q_2$, and $x'$ a fresh variable. We define the substitution $\varphi$ in the following way:
$$\varphi(q(x)) = \begin{cases} \overline{q''}(x') & \text{if } x \in A\text{, where } \overline{q''} \text{ is a new state} \\ q(x) & \text{otherwise} \end{cases}$$
Then, we add all new states to $Q_1$ and replace $l_1 \to s_1$ in $R_1$ by:
$$l_1[x']_{w'} \to (s_1[q'(x')]_w)\varphi \qquad \text{and} \qquad q'(l_1|_{w'}) \to s_1|_w$$
Moreover, for all $x \in A$ with $\varphi(q(x)) = \overline{q''}(x')$ then we add $\overline{q''}(l_1|_{w'}) \to q(x)$.
We note that, if $A = \emptyset$, then this construction preserves linearity and nondeletion.
3. Repeat this procedure.

This procedure is illustrated in Appendix A and in Example 3. Moreover, the latter shows that, in a significant number of cases, this procedure enables $M_1$ to keep its linearity and nondeletion, although this was not possible in the procedure that puts $M_1$ into one-symbol normal form.

*Example 3.* Let $M_1 = (\{q\}, \{\sigma, \alpha\}, \Delta, \{q\}, R_1)$ and $M_2 = (\{p\}, \Delta, \{\delta, \alpha\}, \{p\}, R_2)$ with the alphabet $\Delta = \{f, g_1, g_2, \alpha\}$, and $R_1 = \{\rho_1, \rho_2\}$, $R_2 = \{\mu_1, \mu_2\}$ are defined below and illustrated in Figure 2.

$$\rho_1 : q(\sigma(x_1, \sigma(x_2, \sigma(x_3, x_4)))) \to f(g_1(q(x_1), q(x_2)), g_2(q(x_3), q(x_4))) \quad \rho_2 : q(\alpha) \to \alpha$$
$$\mu_1 : p(f(g_1(y_1, y_2), y_3)) \to \delta(p(y_1), p(y_2), p(y_3)) \quad \mu_2 : p(g_2(\alpha, \alpha)) \to \delta(\alpha, \alpha, \alpha)$$

$M_1$ and $M_2$ are not compatible due to rules $\rho_1$ and $\mu_1$. We need to split $\rho_1$ at position $w = 2$ in the right-hand side, which leads to position $w' = 122$ in the left-hand side, $A = \emptyset$, and the following two rules (illustrated in Figure 3):

$$\rho_{1,1} : q(\sigma(x_1, \sigma(x_2, x'))) \to f(g_1(q(x_1), q(x_2)), q'(x')) \quad \rho_{1,2} : q'(\sigma(x_3, x_4)) \to g_2(q(x_3), q(x_4))$$

We observe that $M_1$ remains linear and nondeleting. It is easy to see that $M_1$ cannot be put into an nl-xtt in one-symbol normal form.

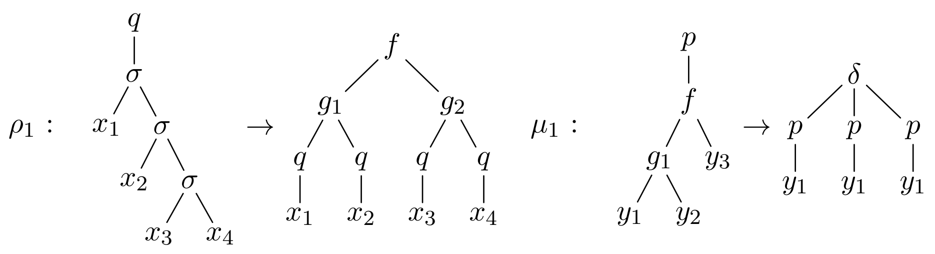


**Fig. 2.** Illustration of rules $\rho_1$ and $\mu_1$

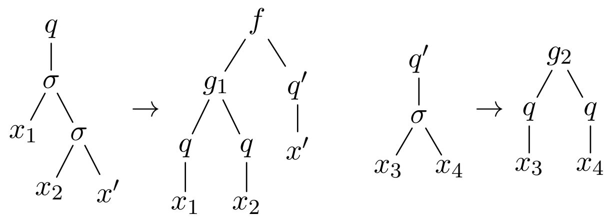


**Fig. 3.** Illustration of the rules created by the compatibility procedure

## 3 Composition step

This section presents the main step of the procedure. This composition step is slighty different whether $M_1$ has lost its linearity and nondeletion during the preprocessing step, or not, and whether $M_1$ has recursive erasing or not. The different cases are handled as follows:

- $M_1$ is not linear anymore, but $M_1$ has no recursive erasing: then the procedure applies the composition step described in Subsection 3.2, using the unification operation described in Subsection 3.1. This case is illustrated in Appendix A.
- $M_1$ is not linear anymore, and has recursive erasing rules: then the procedure fails, giving no conclusion.
- $M_1$ is still linear and nondeleting, then we apply an adapted composition step, described in Subsection 3.3 that can handle recursive erasing.

### 3.1 Unification operation

In the following, we need a *unification* operation on finite sets of trees of $T_{\Sigma\cup\{\star\}}(\mathrm{X})$, for any alphabet $\Sigma$, symbol $\star \notin \Sigma$, and set X of variables. Unification is a mapping $\psi = (\psi_1, \psi_2)$ that matches a set $S$ of linear trees of $T_{\Sigma\cup\{\star\}}(\mathrm{X})$ with disjoint variables into $\psi_1(S) \in T_\Sigma(\mathrm{X}) \cup \{\bot\}$ and a mapping $\psi_2(S)$ from all variables occurring in $S$ into $\mathrm{pos}(\psi_1(S))$. $\psi_1(S)$ is either $\bot$, which means that the unification failed, or a unifier tree, which is the most general tree that matches all the trees in $S$; $\psi_2(S)$ assigns to each variable of the input set, the position in $\psi_1(S)$ matching its position in the input. We define $\psi_1$ and $\psi_2$ by induction:

$$
\psi_1 : \left|\begin{array}{lll}
\{t_1,\ldots,t_l\} & \mapsto \bot & \text{if } \exists i,j \quad t_i(\varepsilon), t_j(\varepsilon) \in \Sigma \text{ and } t_i(\varepsilon) \neq t_j(\varepsilon) \\
\{x_1,\ldots,x_n\} & \mapsto x & \text{where } x \text{ is fresh and } x_1,\ldots,x_n \in \mathrm{X} \\
\{\star(s_1,\ldots,s_r)\} \cup A & \mapsto \psi_1(\{s_1,\ldots,s_r\} \cup A) & \text{where } s_1,\ldots,s_r \in T_{\Sigma\cup\{\star\}}(\mathrm{X}) \text{ and } A \text{ a set} \\
\{\sigma^{(0)}, x_1,\ldots,x_n\} & \mapsto \sigma & \text{where } \sigma \in \Sigma^{(0)}, x_1,\ldots,x_n \in \mathrm{X} \\
\{t_1,\ldots,t_l,x_1,\ldots,x_n\} & \mapsto \delta(t'_1,\ldots,t'_k) & \text{where } t_1(\varepsilon) = \ldots = t_l(\varepsilon) = \delta \in \Sigma, \\
 & & \forall 1 \leq i \leq k \quad \psi_1(\{t_1|_i,\ldots,t_l|_i\}) = t'_i, \\
 & & \text{and } x_1,\ldots,x_n \in \mathrm{X},
\end{array}\right.
$$

If $\psi_1$ succeeds, we build $\psi_2$ in the following way:

$$
\psi_2 : \left|\begin{array}{lll}
\{\star(s_1,\ldots,s_r)\} \cup A & \mapsto \psi_2(\{s_1,\ldots,s_r\} \cup A) & \text{where } s_1,\ldots,s_r \in T_{\Sigma\cup\{\star\}}(\mathrm{X}) \text{ and } A \text{ a set} \\
\{t_1,\ldots,t_n\} & \mapsto \cup_{i=1}^n \psi_2^i & \text{where } t_1,\ldots,t_n \in T_\Sigma(\mathrm{X}), \\
 & & \text{and } \forall 1 \leq i \leq n \quad \psi_2^i\colon x \in \mathrm{var}(t_i) \mapsto \mathrm{pos}_x(t_i)
\end{array}\right.
$$

The unification operation is illustrated in Example 4.

*Example 4.* Let $A$ and $B$ be the following sets of trees:

$$A = \{x_1, f(a, x_2), f(x_3, h(x_4)), f(x_5, h(\star(g(b, b, x_6), g(x_7, b, x_8)))))\}$$

$$B = \{x_1, f(a, x_2), f(x_3, h(x_4)), f(x_5, h(g(b, b, x_6))), f(x_5, h(g(x_7, b, x_8)))\} \ .$$

Then the unification operation run on $A$ and $B$ outputs the following:

$$\psi(A) = \psi(B) = (f(a, h(g(b, b, z))), \psi_2(B)) \quad \text{where} \quad \psi_2(B) : \left| \begin{array}{l} x_1 \mapsto \varepsilon \\ x_2 \mapsto 2 \\ x_3 \mapsto 1 \\ x_4 \mapsto 21 \end{array} \right| \begin{array}{l} x_5 \mapsto 1 \\ x_6 \mapsto 213 \\ x_7 \mapsto 211 \\ x_8 \mapsto 213 \end{array}$$

## 3.2 Construction when $M_1$ is copying, and has no recursive erasing

This subsection is devoted to the main step in the composition procedure. Recall that $M_2$ is an nl-xtt, and assume that $M_1$ and $M_2$ are compatible. Then this step outputs an xtt $M_1;M_2$ with regular look-ahead that computes the composition of $M_1$ and $M_2$. From now on, we will not distinguish between $p(q(t))$ and $(p, q)(t)$ for states $p, q$ and tree $t$. Assume we have a set X of variables disjoint from $X_1 \cup X_2$, and large enough to allow us to pick up as (finitely) many fresh symbols as we need, and a new symbol $\star$ with no fixed arity. We will not distinguish between $\star(\{t_1, \ldots, t_l\})$ and $\star(t_1, \ldots, t_l)$. The set of states $Q_2'$ is the set of states $Q_2$ augmented by as (finitely) many fresh states as we need.

In order to obtain the rule set $R$, we first add $\{(p, q)(l_1) \to s_1 \mid p \in Q_2, q(l_1) \to s_1 \in R_1^\varepsilon\}$ where $R_1^\varepsilon$ stands for the subset of erasing rules of $R_1$. Then, consider every rule $r_2 = p(t) \to s \in R_2$ and state $q \in Q_1$, create a boolean variable $first\_step$ = true, and a new rule $r' = p(q(t)) \to s$ and rewrite it according to the following system. This way, left-hand sides of rules in $R_2$ will be successively built and the corresponding input trees will be identified.

1. If $r' \in Q_2(T_\Sigma(\mathrm{X})) \times T_\Gamma((Q_2 \times Q_1)(\mathrm{X}))$, we are done and augment the root by $q$.
2. If there is a position $z$ such that $\mathrm{lhs}(r')(z) = q \in Q_1$, do one of the following:
   - Choose a $q$ rule $q(\mathbf{t}) \to r_1 \in R_1$ such that there exists $t_1, \ldots, t_k \in T_\Delta(X_2)$ such that $\mathrm{lhs}(r')|_z = q(r_1[t_1, \ldots, t_k]_{w_1,\ldots,w_k})$ where $w_1 \leq \ldots \leq w_k$ are the elements of $\mathrm{pos}_{Q_1}(r_1)$. We replace $\mathrm{lhs}(r')|_z$ by $\mathbf{t}[x_i \leftarrow \star(\{q'(t_j) | \ r_1|_{w_j} = q'(x_i)\})]$. Then, replace each occurrence of $\star(\emptyset)$ by fresh variables (this corresponds to variables that are deleted). Finally, for each subtree of $\mathrm{lhs}(r')|_z$ that has shape $q'(y)$, where $q' \in Q_1$ and $y \in X_2$, replace $q'(y)$ in $\mathrm{lhs}(r')|_z$ by a fresh variable $x'$, and replace $y$ in $s$ by $q'(x')$. Finally, let $first\_step$ = false.
   - (This option is available only if $first\_step$ = false) Choose an erasing rule of shape $q(\mathbf{t}) \to q'(x_1) \in R_1$. Then replace $\mathrm{lhs}(r')|_z$ by $\mathbf{t}[x_1 \leftarrow q'(\mathrm{lhs}(r')|_{z1})]$, and rename all other variables occuring in $\mathbf{t}$ by fresh ones.
3. Otherwise, there is no state left in any subtrees, but $|\,\mathrm{pos}_\star(\mathrm{lhs}(r'))| \geq 1$. Let $z$ be the least position of $\mathrm{pos}_\star(\mathrm{lhs}(r'))$ in the lexicographic order. Then $\mathrm{lhs}(r')|_z = \star(t_1, \ldots, t_l)$ with each $t_i \in T_{\Sigma \cup \{\star\}}(\mathrm{X})$. If $l = 1$, replace $\mathrm{lhs}(r')|_z$ by $t_1$. Otherwise, let $(\mathbf{t}, Y) = \psi(\{t_1, \ldots, t_l\})$, and let $x$ be a fresh variable. If $\mathbf{t} = \perp$, then give up this branch of computation. Otherwise, replace $\mathrm{lhs}(r')|_z$ by $x$. Moreover, for every variable $x' \in \mathrm{supp}(Y)$ occurring in $s$, we replace $(p, q)(x')$ by $(p, q')(x)$, where $q'$ is a new state, and add in $R$ the rule $(p, q')(\mathbf{t}[x_0]_{z'}) \to (p, q)(x_0)$ where $x_0$ is a fresh variable, and $Y(x') = z'$. Finally, the look-ahead $c$ makes sure that $\mathrm{lhs}(r')|_z$ is an instance of $\mathbf{t}$.
4. Repeat this procedure.

Intuitively, for each rule of $M_2$, we try to reconstruct its left-hand side, starting from the root, with right-hand sides of $M_1$, and replace it by the matching input tree from $T_\Sigma(\mathrm{X})$, i.e. the left-hand sides of rules of $M_1$. Moreover, the special symbol $\star$ enables us to handle copying rules of $M_1$, by remembering all the intermediate trees in which the input tree should be derivable.

*Remark 5.* Note that, at each time the symbol $\star$ has only one child $t$, we can omit it and replace the subtree $\star(t)$ by $t$ at any time, without changing the result of the procedure.

## 3.3 Construction when the nl-xtt $M_1$ may have recursive erasing rules

Let us first define recursive erasing states and automata: if there is a set $R_\varepsilon$ of erasing rules in $R_1$ and a reachable state $q \in Q_1$ such that $q(t) \Rightarrow^{+}_{M_1} q(t')$ using only rules of $R_\varepsilon$, then $q$, and more generally, $M_1$ are *recursive erasing.* Obviously then there is a tree $u \in T_\Delta$ such that infinitely many trees are mapped to $u$ by $M_1$. Intuitively, this means that $M_1$ can erase an unbounded chain.

We go on to say that an $M_2$ rule $l \to r$ has recursive erasing potential at position $z$ if there is some state $q$ in $Q_1$, a context $C \in C_\Delta$ and a tree $t \in T_\Sigma$ such that $q(t) \Rightarrow^* C[u]$, where $u$ has a recursive erasing state at position $z$, and $u$ matches $l$, i.e. at every position $y$ of $\mathrm{pos}(u) \cap \mathrm{pos}(l)$, either $u(y) = l(y)$ or $u(y) \in Q_1 \cup X_1$ or $l(y) \in Q_2 \cup X_2$. If there are no variables in $l|_z$, then the position $z$ is called *harmless.*

In the case of recursive erasing, two symbols at an arbitrarily large distance in the input tree may become part of the same left-hand side of an $M_2$ rule. Moreover, this kind of erasing may trigger the impossibility to compute the composition by a single nl-xtt, as shown in Example 6.

*Example 6.* This example illustrates the composition of a recursive erasing nl-xtt $M$ and an nl-xtt $N$ such that the composition cannot be handled by a single nl-xtt. The composition should enable "Recursive erasing followed by reordering", which is known as a property that cannot be fulfilled by any nl-xtt. Let $M = (\{q_0, q, \mathrm{id}\}, \Sigma, \Delta, \{q_0\}, \{\rho_1, \rho_2, \rho_3, \mathrm{id}_\gamma, \mathrm{id}_\alpha\})$, where $\Sigma = \{\sigma, \gamma, \alpha\}$, $\Delta = \{\sigma', \gamma, \alpha\}$, and $N = (\{p, \mathrm{id}\}, \Delta, \Gamma, \{p\}, \{\mu, \mathrm{id}_\gamma, \mathrm{id}_\alpha\})$ where $\Gamma = \{\delta, \gamma, \alpha\}$ and

| | |
|---|---|
| $\rho_1 : q_0(\sigma(x_1, x_2)) \to \sigma'(q(x_1), \mathrm{id}(x_2))$ | $\mu : p(\sigma'(\sigma'(y_1, y_2), y_3)) \to \delta(\mathrm{id}(y_1), \mathrm{id}(y_2), \mathrm{id}(y_3))$ |
| $\rho_2 : q(\sigma(x_1, x_2)) \to \sigma'(\mathrm{id}(x_1), \mathrm{id}(x_2))$ | $\mathrm{id}_\gamma : \mathrm{id}(\gamma(x_1)) \to \gamma(\mathrm{id}(x_1))$ |
| $\rho_3 : q(\gamma(x_1)) \to q(x_1)$ | $\mathrm{id}_\alpha : \mathrm{id}(\alpha) \to \alpha$ |

The shape of possible derivations is shown on Figure 4. We observe that, in order to output the $\delta$ and its three sons, a rule of the composed transducers should be able to access the two sons of the second $\sigma$. This can not be achieved because the two $\sigma$ in the input tree may be separated by an unbounded chain of $\gamma$.

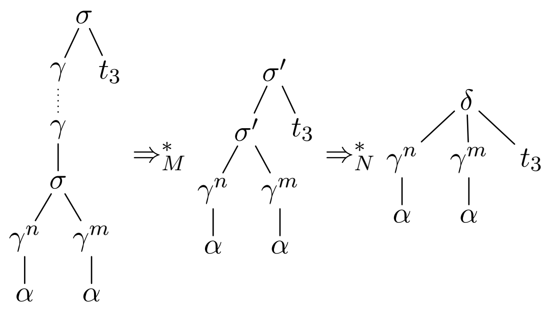


**Fig. 4.** Possible derivations in $M$ then $N$, see Example 6 ($t_3$ is an arbitrary tree).

The composition step is thus modified into the following procedure: note that the star symbol and the unification operation are not needed anymore, as $M_1$ is assumed to be linear and nondeleting.

As in the previous subsection, we will not distinguish between $p(q(t))$ and $(p, q)(t)$ for states $p, q$ and tree $t$. Assume we have a set X of variables disjoint from $X_1 \cup X_2$, and large enough to allow us to pick up as (finitely) many fresh symbols as we need. The set of states $Q'_2$ (resp. $Q'_1$) is the set of states $Q_2$ (resp. $Q_1$) augmented by as (finitely) many fresh states as we need.

In order to obtain the rule set $R$, we consider every rule $r_2 = p(t) \to s \in R_2$ and state $q_0 \in Q_1$, and create a new rule $r' = p(q_0(t)) \to s$ and rewrite it according to the following system. This way, left-hand sides of rules in $R_2$ will be successively built and the corresponding input trees will be identified.

1. If $r' \in Q_2(T_\Sigma(\mathrm{X})) \times T_\Gamma((Q_2 \times Q_1)(\mathrm{X}))$, we are done and augment the root by $q_0$.

2. If there is a position $z$ such that $\mathrm{lhs}(r')(z) = q \in Q_1$ and $q$ is not a recursive erasing state, choose a $q$ rule $q(\mathbf{t}) \to r_1 \in R_1$ such that there exists $t_1, \ldots, t_k \in T_\Delta(X_2)$ such that $\mathrm{lhs}(r')|_z = q(r_1[t_1, \ldots, t_k]_{w_1,\ldots,w_k})$ where $w_1 \leq \ldots \leq w_k$ are the elements of $\mathrm{pos}_{Q_1}(r_1)$. We replace $\mathrm{lhs}(r')|_z$ by $\mathbf{t}$ where we replace $x_i$ by $q'(t_j)$ if $r_1|_{w_j} = q'(x_i)$. Then, for each subtree of $\mathrm{lhs}(r')|_z$ that has shape $q'(y)$, where $q' \in Q_1$ and $y \in X_2$, replace $q'(y)$ in $\mathrm{lhs}(r')|_z$ by a fresh variable $x'$, and replace $y$ in $s$ by $q'(x')$.
3. Otherwise, there is a position $z$ such that $\mathrm{lhs}(r')|_z = q(t_0) \in Q_1$ and $q$ is a recursive erasing state. Let $w$ be the longest common prefix of $\mathrm{pos}_{\mathrm{var}(t_0)}(s)$.
   – If $z$ is a harmless position, i.e. $\mathrm{var}(t_0) = \emptyset$: we simply replace $\mathrm{lhs}(r')|_z$ by a sink variable $x$ which does not occur on the right-hand side of $r'$. Additionally, we have to check whether $\mathrm{lhs}(r')|_z$ is actually in the inverse image of $M_1$, starting in $q$. This is a regular language, and after $r'$ has been completely rewritten, all lookaheads can be combined into a lookahead for the whole rule. This will lead to a linear but deleting xtt.
   – If $\mathrm{var}(s|_w) = \mathrm{var}(t_0)$: let $x$ be a fresh variable, $p'$ a fresh state in $Q_2$, and replace $r'$ by
   $$\mathrm{lhs}(r')[x]_z \to s[(p', q, \mathrm{rec\_eras})(x)]_w$$
   and for all erasing rules of shape $q_1(l_1) \to q_2(x_1)$, add in $R'$ the rules
   $$(p', q_1, \mathrm{rec\_eras})(l_1) \to (p', q_2, \mathrm{rec\_eras})(x_1) \qquad \text{and} \qquad (p', q_2, \mathrm{rec\_eras})(x) \to (p', q_2)(x) \ .$$
   Finally, add the rule $p'(t_0) \to s|_w$ to $R_2$ (which means that we have to process this rule in the same way as the others to get a rule in $R'$)
   – Otherwise:
   Let $\{w_1, \ldots, w_k\} = \mathrm{pos}_{X_2}(t_0)$, and for all $i \in [k]$, let $w'_i \in \mathrm{pos}(s), p_i \in Q_2$ be such that $s(w'_i) = p_i(t_0(w_i))$ ($w_i$, $w'_i$ and $p_i$ are such that the variable at position $w_i$ in $t_0$ occurs under the state $p_i$ at position $w'_i1$ in $s$). Assume there exists $i_0 \in [k]$ such that for all $i \in [k] \setminus \{i_0\}$, the set
   $$\left\{ u' \in T_\Delta \,\middle|\, \begin{array}{l} \exists t \in T_\Sigma, \exists u_1, \ldots, u_k \in T_\Delta, \exists C' \in C_\Delta, \\ \exists u'_1, \ldots, u'_{i-1}, u'_{i+1}, \ldots, u'_k \in T_\Gamma, \exists \theta : Q_2(X_2) \cup (Q_2 \times Q_1)(\mathrm{X}) \to T_\Gamma \\ p(q_0(\mathrm{lhs}(r')[t]_z) \Rightarrow^*_{M_1} p(C'[q(t)]) \Rightarrow^*_{M_1} p(C'[t_0[u_1, \ldots, u_k]_{w_1,\ldots,w_k}]) \\ \Rightarrow^*_{M_2} (s[u'_1, \ldots, u'_{i-1}, u', u'_{i+1}, \ldots, u'_k]_{w'_1,\ldots,w'_k})\theta \end{array} \right\}$$
   is finite, say $\{u'_{i,1}, \ldots, u'_{i,n_i}\}$. Then for all $(j_1, \ldots, j_k) \in [n_1] \times \ldots \times [n_k]$, assume the set
   $$A = \left\{ (C, t_1, \ldots, t_{i_0-1}, t_{i_0+1}, \ldots, t_k) \,\middle|\, \begin{array}{l} C \in C_\Sigma(\mathrm{X}), t_1, \ldots, t_{i_0-1}, t_{i_0+1}, \ldots, t_k \in T_\Sigma, \\ \exists t_{i_0} \in T_\Sigma, \exists q_1, \ldots, q_k \in Q_1, \exists \theta : X_2 \to T_\Delta \\ q_0(C[t_1, \ldots, t_k]) \Rightarrow^{*\Delta}_{M_1} t_0[q_1(t_1), \ldots, q_k(t_k)] \Rightarrow^*_{M_1} t_0\theta \\ \text{and } p_i(q_i(t_i)) \Rightarrow^*_{M_1}; \Rightarrow^*_{M_2} u'_{i,j_i} \end{array} \right\}$$
   is finite ($\Rightarrow^{*\Delta}$ stands for a derivation without recursive erasing rules). Then let $p'$ be a new state in $Q_2$, $q_{i_0}$ as described in the definition of $A$, and add in $R'$ the rule
   $$\mathrm{lhs}(r')[x]_z \to s[u'_{1,j_1}, \ldots, u'_{i_0-1,j_{i_0-1}}, (p', q)(x), u'_{i_0+1,j_{i_0+1}}, \ldots, u'_{k,j_k}]_{w'_1,\ldots,w'_k}$$
   and for all $(C, t_1, \ldots, t_{i_0-1}, t_{i_0-1}, \ldots, t_k) \in A$ and $q' \in Q_1$, we add the rule
   $$(p', q')(C[t_1, \ldots, t_{i_0-1}, x_{i_0}, t_{i_0+1}, \ldots, t_k]) \to (p_{i_0}, q_{i_0})(x_{i_0})$$
   If some recursive erasing might happen in $M_1$ at position $w''_0$ of $C$ (if several, take the smallest), then we split the rule above into
   $$(p', q')(C[t_1, \ldots, t_{i_0-1}, x_{i_0}, t_{i_0+1}, \ldots, t_k][x]_{w''_0}) \to (p'', q', \mathrm{rec\_eras})(x)$$
   and
   $$(p'', q'')(C[t_1, \ldots, t_{i_0-1}, x_{i_0}, t_{i_0+1}, \ldots, t_k]|_{w''_0}) \to (p_{i_0}, q_{i_0})(x_{i_0})$$
   for all states $q'' \in Q_1$, where $p''$ is a new state in $Q_2$, $x$ is a fresh variable. If recursive erasing might happen again in $C[t_1, \ldots, t_{i_0-1}, x_{i_0}, t_{i_0+1}, \ldots, t_k]|_{w''_0}$, then we split it in the same way.

Finally, for all erasing rules of shape $q_1(l_1) \to q_2(x_1)$, and for all $p'$ new state added in $Q_2$, add in $R'$ the rule

$$(p', q_1, \text{rec\_eras})(l_1) \to (p', q_2, \text{rec\_eras})(x_1) \quad \text{and} \quad (p', q_2, \text{rec\_eras})(x_1) \to (p', q_2)(x_1)$$

If one of the assumptions above is false, then we call the position *deadly*, and return "Cannot conclude".

4. Repeat this procedure.

Intuitively, the recursive erasing can be handled in the composition by a single nl-xtt either if we can separate the variables in the right-hand side of the rule, in the same way as recusive erasing does; or if we can predict all but one trees that are represented by a variable under the recursive erasing. These different cases are illustrated in Appendix B, Examples 13, 14, 15. Moreover, recursive erasing at harmless position requires deletion to be computable in the composition.

## 4 Recombining the rules

This section is devoted to the recombining step, which occurs after the main step in the case where $M_1$ has lost its linearity and nondeletion in the preprocessing step. We want to build a nl-xtt that would be equivalent to the xtt output by the main step. However, since the subclass nl-xtt is not closed under composition, we know that this cannot always be achieved. We run the following procedure, that is illustrated in Appendix A:

Assume $M$ is not an nl-xtt. Keep only accessible states, and only rules starting from these states. Then, for all copying rules $r_1 = l_1 \to s_1 \in R'$, let $\varphi = \emptyset$ and $S = \{r_1\}$, and repeat the following while at least one of the conditions is fulfilled:

- If there is $z$ a copied variable that also appear in the left-hand side: let $w = \text{pos}_z(l_1)$ and $w'$ be such that $s_1|_{w'} = (p, q)(z)$ for some $(p, q) \in Q$. Choose a rule of shape $r_2 = (p, q)(l_2) \to s_2 \in R'$, and such that $l_1[l_2]_w$ and $c(r_1)$ can be unified, and rename it with fresh variables. Do the following:
  1. If $r_2 \in S$, return "Cannot conclude". Otherwise, add $r_2$ to $S$.
  2. Change $r_1$ into $l_1[l_2]_w \to s_1[s_2]_{w'}$.
  3. Modify the look-ahead $c(r_1)$ into $c(r_1)[c(r_2)]_w$.
  4. Add $(z, w)$ to $\varphi$.
  5. For all $z' \in \text{var}(l_2)$, add $(z', w.\,\text{pos}_{z'}(l_2))$ to $\varphi$.

  If there was no such $r_2$, give up this branch of computation.
- If there is a variable $z \in \text{var}(s_1) \setminus \text{var}(l_1)$, i.e. there is a variable occuring in the right-hand side but not in the left-hand side: let $w = \varphi(z)$ and $w'$ be such that $s_1|_{w'} = (p, q)(z)$ for some $(p, q) \in Q$.
  If $l_1(w) = z' \in \mathrm{X}$, then replace $z$ by $z'$ in $s_1$. Otherwise:
  Choose a rule $r_2 = (p, q)(l_2) \to s_2$ such that $l_1[l_2]_w$ and $c(r_1)$ can be unified, and rename it with fresh variables. Let $(t, \varphi') = \psi(l_1|_w, l_2)$ (unification operation). Assume that $t$ can be unified with $c(r_2)$. Do the following:
  1. If $r_2 \in S$, return "Cannot conclude". Otherwise, add $r_2$ to $S$.
  2. Modify $r_1$ into $l_1[t]_w \to s_1[s_2]_{w'}$
  3. Modify the look-ahead $c(r_1)$ into $c(r_1)[c(r_2)]_w$.
  4. For all $z' \in \text{var}(l_2)$, add $(z', w.\varphi'(z'))$ to $\varphi$.

  If there was no such $r_2$, or else if $t = \perp$, or $t$ is not unifiable with $c(r_2)$, then give up this branch of computation.

Before going on to another copying rule, delete all rules that do not start from an accessible state.

At the end of the procedure, we decide in the following way:

- If $M$ is linear nondeleting, we are done, and return $M$.
- Otherwise, return "Cannot conclude".

Intuitively, we try to get rid of copied variables by merging this rule with rules that can process the copied variables. Not only may this rebuild partially or totally the rules that were cut in the preprocessing step, but this can build even larger rules. The process stops whenever entering a loop, that is to say we need to apply twice the same rule.

# Part II - Composition of Weighted Xtt

This part is devoted to weighted compositions of selected xtt with $\varepsilon$tdtt. The reader is strongly encouraged to read Section 3 of Appendix C for a presentation of the general approach used in most composition constructions. Due to length restrictions, all examples have been removed. The reader can find all of them in Appendix C.

## 1 An Equivalent Model

In this section, we introduce an alternative description for weighted extended top-down tree transducers that will be useful for our composition construction. Essentially, we introduce explicit rule identifiers that stand for a specific rule. A mapping that becomes part of the specification assigns weighted rules to identifiers. This indirection allows us to use multiple rules with the same left- and right-hand side and even the same weight. In our composition constructions we use this facility to establish a more concise and simpler relation between the constructed rules of the composed xtt and the original rules of the input xtt.

**Definition 7.** *A* weighted extended (top-down) tree transducer with rule identifiers *is a system* $(Q, \Sigma, \Delta, I, \mathcal{R}, \chi)$*, where*

- $Q$*,* $\Sigma$*,* $\Delta$*, and* $I$ *are the same as the corresponding elements of an xtt,*
- $\mathcal{R}$ *is a finite set of* rule identifiers*, and*
- $\chi\colon \mathcal{R} \to Q(T_\Sigma(\mathrm{X})) \times A \times T_\Delta(Q(\mathrm{X}))$ *is a* rule assignment *that maps each rule identifier* $\rho \in \mathcal{R}$ *to its content* $\chi(\rho) = (l, a, r)$ *such that* $\{l, r\} \not\subseteq Q(\mathrm{X})$*,* $l$ *is linear, and* $\mathrm{var}(r) \subseteq \mathrm{var}(l)$*.*

In accordance with our notation for rules, we often write $l \xrightarrow{a} r$ instead of $(l, a, r)$ for elements of $Q(T_\Sigma(\mathrm{X})) \times A \times T_\Delta(Q(\mathrm{X}))$. Moreover, we let $\mathrm{wt}\colon \mathcal{R} \to A$ be such that $\mathrm{wt}(\rho) = a$ for all $\rho \in \mathcal{R}$ with $\chi(\rho) = l \xrightarrow{a} r$. Intuitively, 'wt' maps a rule identifier to the weight of its identified rule.

The semantics of the xtt $M = (Q, \Sigma, \Delta, I, \mathcal{R}, \chi)$ with rule identifiers $\mathcal{R}$ is given by rewriting in essentially the same way as before. Let $\Sigma'$ and $\Delta'$ be two alphabets such that $\Sigma \subseteq \Sigma'$ and $\Delta \subseteq \Delta'$ and $Q \cap (\Sigma' \cup \Delta') = \emptyset$. Again, we call a position $w \in \mathrm{pos}_Q(\xi)$ in a sentential form $\xi \in T_{\Delta'}(Q(T_{\Sigma'}(\mathrm{X})))$ *reducible (for $M$)* if there exists a rule $\rho \in \mathcal{R}$ with $\chi(\rho) = l \xrightarrow{a} r$ and a substitution $\theta\colon \mathrm{X} \to T_{\Sigma'}(\mathrm{X})$ such that $\xi|_w = l\theta$. Now, let $\xi, \zeta \in T_{\Delta'}(Q(T_{\Sigma'}(\mathrm{X})))$, $\rho \in \mathcal{R}$, and $\chi(\rho) = l \xrightarrow{a} r$. We say that $\xi$ *rewrites to $\zeta$ using $\rho$*, denoted by $\xi \Rightarrow^\rho_M \zeta$, if there exists a substitution $\theta\colon \mathrm{X} \to T_{\Sigma'}(\mathrm{X})$ such that $\xi|_w = l\theta$ and $\zeta = \xi[r\theta]_w$ where $w$ is the least reducible position in $\mathrm{pos}_Q(\xi)$ with respect to the lexicographic total order on $\mathbb{N}^*$. The *(extended) weighted relation $\tau'_M$ computed by $M$* is given by

$$\tau'_M(\xi, \zeta) = \sum_{\substack{\rho_1, \dots, \rho_k \in \mathcal{R} \\ \xi \Rightarrow^{\rho_1}_M ; \cdots ; \Rightarrow^{\rho_k}_M \zeta}} \Big( \prod_{i=1}^{k} \mathrm{wt}(\rho_i) \Big)$$

for every $\xi, \zeta \in T_{\Delta'}(Q(T_{\Sigma'}(\mathrm{X})))$. As for xtt, the *semantics* $\tau_M$ of the xtt $M$ with rule identifiers is the weighted relation $\tau_M\colon T_\Sigma \times T_\Delta \to A$ such that $\tau_M(t, u) = \sum_{q \in I} \tau'_M(q(t), u)$ for every $t \in T_\Sigma$ and $u \in T_\Delta$. The properties of rules and xtt defined in Sect. 3 generalize straightforwardly to xtt with rule identifiers.

The reader can find an example of such an xtt in Example 4 of Section 4 in Appendix C.

We can easily see that the two models of xtt are equally expressive. For every xtt $(Q, \Sigma, \Delta, I, R)$ we can construct an equivalent xtt $(Q, \Sigma, \Delta, I, R, \chi)$ with rule identifiers by setting $\chi = \mathrm{id}_R$, where $\mathrm{id}_R$ is the identity on $R$. Conversely, given an xtt $M = (Q, \Sigma, \Delta, I, \mathcal{R}, \chi)$ with rule identifiers we can obtain an equivalent xtt $(Q, \Sigma, \Delta, I, R)$ by setting

$$R(l, r) = \sum_{\rho \in \mathcal{R}\colon \chi(\rho) = (l, a, r)} \mathrm{wt}(\rho) \tag{2}$$

for all $l \in Q(T_\Sigma(\mathrm{X}))$ and $r \in T_\Delta(Q(\mathrm{X}))$.[8] The construction is illustrated in Example 5, Section 4 of Appendix C.

[8] The sum (2) returns 0, as desired, if $M$ has no rules with left-hand side $l$ and right-hand side $r$.

## 2 Composition of an xtt with a tdtt with $\varepsilon$-rules

This section is devoted to compositions of tree transformations computed by xtt $M$ and $N$, of which the xtt $N$ is a top-down tree transducer with $\varepsilon$-rules [37]. Roughly speaking, a top-down tree transducer with $\varepsilon$-rules is an xtt, in which simple and $\varepsilon$-rules are allowed. In the unweighted setting, this scenario was investigated in [37], and we essentially report the results of [37], which we adjusted to our weighted setting. Let us start with the formal definition of the requirements of this section. For the rest of this section, let $M = (Q, \Sigma, \Gamma, I_1, R_1)$ and $N = (P, \Gamma, \Delta, I_2, R_2)$ be the xtt that we want to compose.

**Definition 8 (cf. [15, Definition 4] and [37, Definition 1]).**

- *The xtt $M$ is* shallow *if $|\mathrm{pos}_\Gamma(r)| \leq 1$ for every $l \to r \in R_1$.*
- *The xtt $N$ is a* tdtt with $\varepsilon$-rules *if $|\mathrm{pos}_\Gamma(l)| \leq 1$ for every $l \to r \in R_2$.*

Clearly, each tdtt is a tdtt with $\varepsilon$-rules, but a tdtt need not be shallow.

Now we can formally define the goal of this section. We will investigate compositions of xtt $M$ and $N$ such that $M$ is shallow and $N$ is a tdtt with $\varepsilon$-rules. We additionally assume here that $N$ does only have producing rules. In this case, there can only be finitely many rule applications generating the output tree $u$, which limits the size of the intermediate tree. Thus, all compositions are well-defined in the cases of this section.

### 2.1 Construction

For the sake of simplicity, we identify elements of $P(Q(T_\Sigma(\mathrm{X})))$ with elements of $(P \times Q)(T_\Sigma(\mathrm{X}))$ in the obvious manner.

**Definition 9 (cf. [37, Definition 9]).** *The* $\varepsilon$-composition *$M \,;_\varepsilon\, N$ of $M$ and $N$ is the xtt $(P \times Q, \Sigma, \Delta, I_2 \times I_1, \mathcal{R}, \chi)$ with rule identifiers*

$$\begin{aligned}\mathcal{R} = \{\langle \rho, p, \varepsilon\rangle \mid \textit{erasing } \rho \in R_1, p \in P\} \cup{} \\ \cup \{\langle \rho, p, \mu\rangle \mid \textit{producing } \rho \in R_1, p \in P,\ \textit{consuming } \mu \in R_2\} \cup{} \\ \cup \{\langle \varepsilon, q, \mu\rangle \mid q \in Q,\ \varepsilon\textit{-rule } \mu \in R_2\}\end{aligned}$$

*such that*

- *$\chi(\langle l \to r, p, \varepsilon\rangle) = (p(l), R_1(l \to r), p(r))$ for every erasing rule $l \to r \in R_1$ and $p \in P$,*
- *$\chi(\langle l \to r, p, \mu\rangle) = (p(l), a, r')$, where*

$$a = \begin{cases} R_1(l \to r) \cdot R_2(\mu) & \textit{if } p(l) \Rightarrow_M^{(l,r)} ; \Rightarrow_N^\mu r' \\ 0 & \textit{otherwise} \end{cases}$$

*for every producing $l \to r \in R_1$, $p \in P$, and consuming $\mu \in R_2$, and*
- *$\chi(\langle \varepsilon, q, l \to r\rangle) = (l\theta, R_2(l \to r), r\theta)$, where $\theta(x) = q(x)$ for every $x \in \mathrm{X}$, $q \in Q$, and $\varepsilon$-rule $l \to r \in R_2$.*

Note that the only differences to the construction of [37] are the presence of (i) non-simple left-hand sides in rules of $M$ and (ii) weights. Let us discuss the three sets of rule identifiers mentioned in Definition 9. Rule identifiers of the form $\langle \rho, p, \varepsilon\rangle$ refer to variants of an erasing rule $\rho$ of $R_1$. For each state $p \in P$, we obtain a variant by annotating the two states (in the left- and right-hand side) by $p$. In other words, we perform a step using $M$, but since no intermediate symbol is produced, we do not perform a step using $N$. Second, the rule identifiers of the form $\langle \rho, p, \mu\rangle$ contain rules that are obtained in the usual way by processing the right-hand side of a producing rule of $M$ by consuming rules of $N$. Since $M$ is shallow and $N$ is a tdtt with $\varepsilon$-rules, each producing rule of $M$ contains exactly one intermediate symbol and each consuming rule of $N$ contains exactly one intermediate symbol. Thus, the derivation only succeeds if the producing rule of $M$ produces exactly the symbol that the consuming rule of $N$ consumes. Finally, rule identifiers of the form $\langle \varepsilon, q, \mu\rangle$ refer to a variant of an $\varepsilon$-rule $\mu$ of $N$ that is annotated with the state $q \in Q$.

Let us quickly check whether the obtained rules are admissible; i.e., whether $\{l, r\} \not\subseteq P(Q(\mathrm{X}))$ for every rule $l \to r$ . Clearly, identifiers of the form $\langle \rho, p, \varepsilon \rangle$ yield admissible rules because they contain just copies of rules of $M$. The same reasoning applies to rules with identifiers of the form $\langle \varepsilon, q, \mu \rangle$, which are copies of rules of $N$. Finally, rules with identifiers like $\langle \rho, p, \mu \rangle$ are always producing because each rule $\mu \in R_2$ is producing. Clearly, producing rules are admissible. The construction is illustrated in Example 18, in Section 6 of Appendix C.

## 2.2 Correctness

Let us start by recalling the two cases in which the composition construction of [37], of which our construction in Definition 9 is an adaptation, is successful in the unweighted setting. Recall that $M$ is shallow and $N$ is a tdtt with $\varepsilon$-rules. If

– $N$ is linear, and
– $M$ is total or $N$ is nondeleting,

then $\tau_M ; \tau_N$ can be computed by an xtt [37, Theorem 17]. The different cases were already presented in Table 1.

Let us start with Case (a) (see Table 1): $N$ is linear and nondeleting. This case does not cause further problems in the weighted setting, and we will sketch the correctness proof for our composition construction of Definition 9.

**Theorem 10.** *If $M$ is a shallow xtt and $N$ is a linear and nondeleting tdtt with $\varepsilon$-rules, then $\tau_{M ;_\varepsilon N} = \tau_M ; \tau_N$.*

*Proof (sketch).* Let $\xi \in P(Q(T_\Sigma))$ and $u \in T_\Delta$. We claim that there is a weight-preserving bijection between the derivations of the form

$$\xi \; (\Rightarrow_M^{\rho_1} ; \cdots ; \Rightarrow_M^{\rho_k}) ; (\Rightarrow_N^{\mu_1} ; \cdots ; \Rightarrow_N^{\mu_n}) \; u \enspace ,$$

and the derivations of the form $\xi \Rightarrow_{M ;_\varepsilon N}^{\nu_1} ; \cdots ; \Rightarrow_{M ;_\varepsilon N}^{\nu_\ell} u$.

We construct the bijection by induction on $k$. Let $s \in T_\Sigma$, $p \in P$, and $q \in Q$ be such that $\xi = p(q(s))$. Next, we distinguish whether the first applied rule $\rho_1$ is erasing. If it is, then we start the derivation using $M ;_\varepsilon N$ with the rule $\langle \rho_1, p, \varepsilon \rangle$, which has the same weight as $\rho_1$. Otherwise, the rule $\rho_1$ produces exactly one intermediate symbol $\gamma \in \Gamma$ that will be consumed by exactly one rule $\mu_i$ for some $i \in \mathbb{N}$. The symbol $\gamma$ is consumed by exactly one rule because $N$ is linear and nondeleting. Clearly, all rules $\mu_1, \ldots, \mu_{i-1}$ before $\mu_i$ must be $\varepsilon$-rules because otherwise they would consume the symbol $\gamma$. In $M ;_\varepsilon N$ we simulate this derivation by starting with the $\varepsilon$-rules $\langle \varepsilon, q, \mu_1 \rangle, \ldots, \langle \varepsilon, q, \mu_{i-1} \rangle$ followed by the consuming rule $\langle \rho_1, p', \mu_i \rangle$, where $p'$ is the (unique) state that occurs in the right-hand side of the rule $\mu_{i-1}$.[9] Clearly, this part of the derivation has the same weight as the product of the weight of rule $\rho_1$ and the weights of the rules $\mu_1, \ldots, \mu_i$. Now we covered all three cases and shortened the derivation using $M$. The remainder of the derivation can then be processed using the induction hypothesis. Thus, our construction relates derivations bijectively and preserves the weight. Given this bijective and weight-preserving relation, the main statement follows trivially. □

The construction used in the proof of Theorem 10 is illustrated Example 20 of Section 6 in Appendix C.

Let us move on to Case (b) (see Table 1): $N$ is linear and $M$ is total. This case is problematic in the weighted setting because the $\varepsilon$tdtt $N$ can delete an intermediate subtree $t'$ that was output by $M$ as the result of processing an input subtree $s'$. In the composed tdtt, the input subtree $s'$ is deleted right away without processing it. This phenomenon is abstractly illustrated in Fig. 5.

Here, we avoid the problem by requiring (i) that the xtt $M$ is BOOLEAN and (ii) that the semiring $A$ is idempotent (i.e., $1 + 1 = 1$). This yields that essentially, $\tau_M(s, t) = 1$ for all $(s, t) \in \operatorname{supp}(\tau_M)$ because $M$ is BOOLEAN and $A$ is idempotent, and for every $s \in T_\Sigma$ there exists $t \in T_\Gamma$ such that $(s, t) \in \operatorname{supp}(\tau_M)$ due to the totality of $M$. Thus we can predict the weight of the intermediate deleted subtree.

[9] If $i = 1$, then we let $p' = p$.

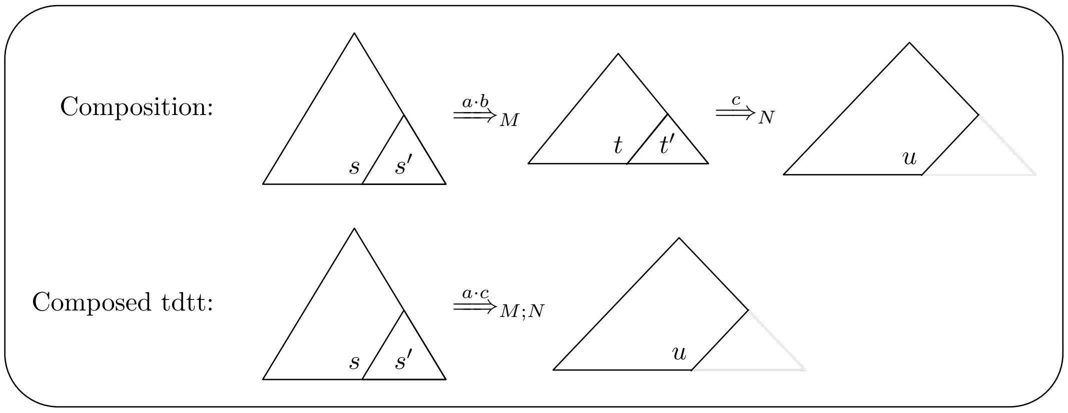

Composition:
$\overset{a\cdot b}{\Longrightarrow}_M$
$\overset{c}{\Longrightarrow}_N$
$s$
$s'$
$t$
$t'$
$u$
Composed tdtt:
$\overset{a\cdot c}{\Longrightarrow}_{M;N}$

**Fig. 5.** Difference between composition and the composed tdtt. Atop the arrows we mark the weight and next to it the tdtt, in which the derivation happens. More precisely, weight $a$ is charged for processing $s$ (without $s'$), weight $b$ is charged for processing $s'$, and weight $c$ is charged for processing $t$ (without $t'$).

**Theorem 11.** *If the shallow xtt $M$ is total and* Boolean*, the tdtt $N$ with $\varepsilon$-rules is linear, and the semiring $A$ is idempotent, then $\tau_{M;_\varepsilon N} = \tau_M \,;\, \tau_N$.*

*Proof (sketch).* Let $\xi \in P(Q(T_\Sigma))$ and $u \in T_\Delta$. We claim that there is a weight-preserving surjective mapping $f$ from the derivations of the form

$$\xi\ (\Rightarrow_M^{\rho_1} ; \cdots ; \Rightarrow_M^{\rho_k}) \,;\, (\Rightarrow_N^{\mu_1} ; \cdots ; \Rightarrow_N^{\mu_n})\ u\ ,$$

and the derivations of the form $\xi \Rightarrow_{M;_\varepsilon N}^{\nu_1} ; \cdots ; \Rightarrow_{M;_\varepsilon N}^{\nu_\ell} u$.

Clearly, the derivation sequence $\rho_1 \cdots \rho_k \mu_1 \cdots \mu_n$ is successful. Let $\perp$ be a fresh symbol, and let $l \to r \in R_1$ be a rule of $M$. The *mutilated copy* of $l \to r$ is the rule $l \to \perp(r)$. We denote the mutilated copy of $\rho \in R_1$ by $\overline{\rho}$. Next, we obtain a rule sequence $\rho'_1 \cdots \rho'_k$ from $\rho_1 \cdots \rho_k$ by replacing maximally many rules $\rho_i$ by their mutilated copy $\overline{\rho_i}$ such that

$$\xi\ (\Rightarrow_M^{\rho'_1} ; \cdots ; \Rightarrow_M^{\rho'_k}) \,;\, (\Rightarrow_N^{\mu_1} ; \cdots ; \Rightarrow_N^{\mu_n})\ u\ .$$

In order words, the new rule sequence is still a successful derivation from $\xi$ to $u$. Clearly, this derivation can only be successful if $N$ ignores (i.e., deletes) the subtrees created by mutilated rules because $N$ cannot process the symbol $\perp$. In the next step we drop all mutilated rules from the rule sequence $\rho'_1 \cdots \rho'_k$ and relate the obtained rule sequence in the same way as in the proof of Theorem 10 to the derivation of the composed xtt. The obtained derivation using the composed xtt has the same weight as the original derivation because we only dropped rules of $R_1$, which have weight 1 because $M$ is Boolean. It is not difficult to see that this mapping is surjective because we can always recover one subderivation for parts that we dropped due to the totality of $M$. This approach is illustrated in an example following the proof.

Now we complete the proof as follows:

$$(\tau'_M \,;\, \tau'_N)(\xi, u) = \sum_{\substack{\rho_1,\dots,\rho_k \in R_1 \\ \mu_1,\dots,\mu_n \in R_2 \\ \xi(\Rightarrow_M^{\rho_1};\cdots;\Rightarrow_M^{\rho_k});(\Rightarrow_N^{\mu_1};\cdots;\Rightarrow_N^{\mu_n})u}} \Big(\prod_{i=1}^{k} R_1(\rho_i) \cdot \prod_{i=1}^{n} R_2(\mu_i)\Big)$$

$$= \sum_{\substack{\nu_1,\dots,\nu_\ell \in \mathcal{R} \\ d\colon \xi \Rightarrow_{M;_\varepsilon N}^{\nu_1};\cdots;\Rightarrow_{M;_\varepsilon N}^{\nu_\ell} u}} \Big( \sum_{d' \in f^{-1}(d)} \Big(\prod_{i=1}^{\ell} \mathrm{wt}(\nu_i)\Big)\Big)$$

(because the second sum is never empty due to surjectivity of $f$)

$$= \sum_{\substack{\nu_1,\dots,\nu_\ell \in \mathcal{R} \\ \xi \Rightarrow_{M;_\varepsilon N}^{\nu_1};\cdots;\Rightarrow_{M;_\varepsilon N}^{\nu_\ell} u}} \Big(\prod_{i=1}^{\ell} \mathrm{wt}(\nu_i)\Big) = \tau'_{M;_\varepsilon N}(\xi, u)$$

because $A$ is idempotent and $f^{-1}(d) \neq \emptyset$. Thus, we conclude that $M \,;_\varepsilon N$ computes $\tau_M \,; \tau_N$. $\square$

The reader can find an illustration of the construction used in the proof of Theorem 11 in Example 22 of Section 6 in Appendix C.

# Conclusion

Although it lasted only ten weeks, my internship was a true immersion in the research world. First, I had to familiarize myself with the new notion of transducer, and with all its variants. Then, it was very interesting to see the difference between writing a survey, that summarizes existing results and slightly extends some of them, and an original research work such as the procedure for unweighted nl-xtt.

This procedure must be implemented in the next months, and tested on a large number of transducers. The aim is to evaluate the number of cases where linearity and nondeletion is lost, and the number of cases where the procedure fails. The next nice step would be to improve the procedure such that it never fails: either it would return the composed nl-xtt, or it would affirm that the composition cannot be achieved in such a device.

As already mentionned in the introduction, the survey is to appear as a book chapter in the Lecture Notes for Computer Science, Springer. The other part will be turned into an article, and submitted in a Natural Language Processing conference.

## References

1. Alexandrakis, A., Bozapalidis, S.: Weighted grammars and Kleene's theorem. Inf. Process. Lett. 24(1), 1–4 (1987)
2. Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., Mohri, M.: OpenFst — a general and efficient weighted finite-state transducer library. In: CIAA 2007. LNCS, vol. 4783, pp. 11–23. Springer-Verlag, Berlin, Heidelberg (2007)
3. Arnold, A., Dauchet, M.: Bi-transductions de forêts. In: Michaelson, S., Milner, R. (eds.) ICALP 1976. pp. 74–86. Edinburgh University Press (1976)
4. Arnold, A., Dauchet, M.: Morphismes et bimorphismes d'arbres. Theoret. Comput. Sci. 20(4), 33–93 (1982)
5. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)
6. Baker, B.S.: Composition of top-down and bottom-up tree transductions. Inform. and Control 41(2), 186–213 (1979)
7. Berstel, J., Reutenauer, C.: Recognizable formal power series on trees. Theoret. Comput. Sci. 18(2), 115–148 (1982)
8. Borchardt, B.: The Theory of Recognizable Tree Series. Ph.D. thesis, Technische Universität Dresden (2005)
9. Borchardt, B., Vogler, H.: Determinization of finite state weighted tree automata. J. Autom. Lang. Combin. 8(3), 417–463 (2003)
10. Bozapalidis, S., Louscou-Bozapalidou, O.: The rank of a formal tree power series. Theoret. Comput. Sci. 27(1–2), 211–215 (1983)
11. Dauchet, M.: Transductions inversibles de forêts. Thèse 3ème cycle, Université de Lille (1975)
12. Engelfriet, J.: Bottom-up and top-down tree transformations: A comparison. Math. Systems Theory 9(3), 198–231 (1975)
13. Engelfriet, J.: Top-down tree transducers with regular look-ahead. Math. Systems Theory 10(1), 289–303 (1976)
14. Engelfriet, J., Fülöp, Z., Vogler, H.: Bottom-up and top-down tree series transformations. J. Autom. Lang. Combin. 7(1), 11–70 (2002)
15. Engelfriet, J., Lilin, E., Maletti, A.: Extended multi bottom-up tree transducers. In: Ito, M., Toyama, F.M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 289–300. Springer, Heidelberg (2008)
16. Ésik, Z., Kuich, W.: Formal tree series. J. Autom. Lang. Combin. 8(2), 219–285 (2003)
17. Fülöp, Z., Vogler, H.: Tree series transformations that respect copying. Theory Comput. Systems 36(3), 247–293 (2003)

18. Golan, J.S.: Semirings and their Applications. Kluwer Academic, Dordrecht (1999)
19. Graehl, J.: Carmel finite-state toolkit. ISI/USC, `http://www.isi.edu/licensed-sw/carmel` (1997)
20. Graehl, J., Knight, K., May, J.: Training tree transducers. Computational Linguistics 34(3), 391–427 (2008)
21. Hebisch, U., Weinert, H.J.: Semirings — Algebraic Theory and Applications in Computer Science. No. 5 in Series in Algebra, World Scientific, Singapore (1998)
22. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison Wesley (1979)
23. Kanthak, S., Ney, H.: FSA: an efficient and flexible $C^{++}$ toolkit for finite state automata using on-demand computation. In: ACL 2004. pp. 510–517 (2004)
24. Kaplan, R.M., May, M.: Regular models of phonological rule systems. Computational Linguistics 20(3), 331–378 (1994)
25. Knight, K., Graehl, J.: An overview of probabilistic tree transducers for natural language processing. In: Gelbukh, A.F. (ed.) CICLing 2005. LNCS, vol. 3406, pp. 1–24. Springer, Heidelberg (2005)
26. Kühnemann, A.: Benefits of tree transducers for optimizing functional programs. In: Arvind, V., Ramanujam, R. (eds.) FSTTCS 1998. LNCS, vol. 1530, pp. 146–157. Springer-Verlag, Berlin, Heidelberg (1998)
27. Kuich, W.: Formal power series over trees. In: Bozapalidis, S. (ed.) DLT 1997. pp. 61–101. Aristotle University of Thessaloniki (1998)
28. Kuich, W.: Full abstract families of tree series I. In: Karhumäki, J., Maurer, H.A., Paun, G., Rozenberg, G. (eds.) Jewels are Forever, pp. 145–156. Springer, Heidelberg (1999)
29. Kuich, W.: Tree transducers and formal tree series. Acta Cybernet. 14(1), 135–149 (1999)
30. Lagoutte, A., Maletti, A.: Survey: Weighted extended top-down tree transducers — part III: Composition. In: Kuich, W., Rahonis, G. (eds.) Algebraic Foundations in Computer Science. Springer (2011), preprint available at: `http://www.ims.uni-stuttgart.de/∼maletti/pub/lagmal11.pdf`
31. Maletti, A.: Compositions of tree series transformations. Theoret. Comput. Sci. 366(3), 248–271 (2006)
32. Maletti, A.: The Power of Tree Series Transducers. Ph.D. thesis, Technische Universität Dresden (2006)
33. Maletti, A.: Compositions of extended top-down tree transducers. Inform. and Comput. 206(9–10), 1187–1196 (2008)
34. Maletti, A.: Survey: Weighted extended top-down tree transducers — Part I: Basics and expressive power. Acta Cybernet. (2011), preprint available at: `http://www.ims.uni-stuttgart.de/∼maletti/pub/mal11.pdf`
35. Maletti, A.: Survey: Weighted extended top-down tree transducers — Part II: Application in machine translation. Fund. Inform. (2011), preprint available at: `http://www.ims.uni-stuttgart.de/∼maletti/pub/mal11b.pdf`
36. Maletti, A., Graehl, J., Hopkins, M., Knight, K.: The power of extended top-down tree transducers. SIAM J. Comput. 39(2), 410–430 (2009)
37. Maletti, A., Vogler, H.: Compositions of top-down tree transducers with $\varepsilon$-rules. In: Yli-Jyrä, A., Kornai, A., Sakarovitch, J., Watson, B. (eds.) FSMNLP 2009. LNAI, vol. 6062, pp. 69–80. Springer, Heidelberg (2010)
38. May, J., Knight, K.: TIBURON: A weighted tree automata toolkit. In: CIAA. LNCS, vol. 4094, pp. 102–113. Springer (2006)
39. May, J., Knight, K., Vogler, H.: Efficient inference through cascades of weighted tree transducers. In: ACL 2010. pp. 1058–1066. Association for Computational Linguistics (2010)
40. Mohri, M.: Handbook of Weighted Automata, chap. Weighted Automata Algorithms, pp. 209–252. Springer (2009)
41. Mohri, M., Pereira, F.C.N., Riley, M.: The design principles of a weighted finite-state transducer library. Theoret. Comput. Sci. 231(1), 17–32 (2000)
42. Rounds, W.C.: Mappings and grammars on trees. Math. Systems Theory 4(3), 257–287 (1970)
43. Thatcher, J.W.: Generalized$^2$ sequential machine maps. J. Comput. System Sci. 4(4), 339–367 (1970)
44. Yamada, K., Knight, K.: A decoder for syntax-based statistical MT. In: ACL 2002. pp. 303–310. Association for Computational Linguistics (2002)

# Appendices

## A Running example for the full procedure

This appendix is devoted to an example on which we successively run the different procedures presented in Part I. Let $\Sigma = \{\sigma, \alpha, \gamma\}$, $\Delta = \{f, g_1, g_2, h_1, h_2, d_1, d_2, e, b, a\}$ , $\Gamma = \{\delta', \sigma', \alpha, \beta\}$, and then let

$$M_1 = (\{q\}, \Sigma, \Delta, \{q\}, \{\rho_1, \ldots, \rho_5\}) \qquad \text{and} \qquad M_2 = (\{p\}, \Delta, \Gamma, \{p\}, \{\mu_1, \ldots, \mu_5\})$$

The rules of $M_1$ and $M_2$ are displayed in Figure 6, and we can see that $M_1$ and $M_2$ are linear nondeleting. However, $M_1$ is not compatible with $M_2$. We have to split the first three rules of $M_1$ according to the preprocessing step. We end up with a new xtt

$$M_1' = (\{q, q_1, q_2, \overline{q_3}, q_4\}, \Sigma, \Delta, \{q\}, \{\rho_{11}, \rho_{12}, \rho_{21}, \rho_{22}, \rho_{31}, \rho_{32}, \rho_{33}, \rho_4, \rho_5\})$$

whose new rules are displayed on Figure 7, and which is no longer linear-nondeleting. We apply the composition step to get the xtt $M_1'; M_2$. Figure 8 explicit the processing of rule $\mu_2$ to get three rules for the composed xtt. Note that, according to Remark 5, at each time the symbol $\star$ has only one child, we omit $\star$ and plug only the child. The remaining rules are shown on Figure 9. As the composed xtt is not linear-nondeleting, we recombine its rules according to the postprocessing step: we start by processing $\nu_1$, which is detailed in Figure 10, to get a new rule $\nu_1'$. Then, we remove all unaccesible states, and all rules starting from these states. Only $\nu_1'$, $\nu_4$ and $\nu_6$ are left, which are linear and nondeleting: we have a nl-xtt computing the composition of $M_1$ and $M_2$, which was our initial goal.

## B Example for recursive erasing

This appendix is devoted to examples that illustrate the different bullets of Step 3, in the composition procedure that handle recursive erasing.

*Example 12.* This example illustrates the first bullet, when recursive erasing happens at a harmless position. The shape of possible derivations is shown on Figure 11.

The rules of $M$ and $N$ are given (in addition to *id* rules, which, for every symbol in state *id*, rewrite this symbol and propagate the state *id*.), by the following:

$\rho_1 : q_0(\sigma(x_1, x_2)) \to \sigma(q_{rec}(x_1), id(x_2))$ $\mu_1 : p(\sigma(\gamma(\alpha), y_1)) \to \delta(\beta, id(y_1), \beta)$
$\rho_2 : \quad q_{rec}(\gamma(x_1)) \to q_{rec}(x_1)$
$\rho_3 : \quad q_{rec}(\gamma(x_1)) \to id(x_1)$

In this case, we can handle the composition by checking that the input tree can produce the right pattern, and delete the subtree where recursive erasing happens. The linear but deleting rule produced by the composition step is:

$$\nu_1 : (p, q_0)(\sigma(y, y_1)) \to \delta(\beta, (id, id)(y_1), \beta)$$

with a regular look-ahead that checks that $y$ is in $M^{-1}(\gamma(\alpha))$.

*Example 13.* This example illustrates the second bullet. The shape of possible derivations is shown on Figure 12, and rules of $M$ and $N$ are shown on Figure 13 (in addition to *id* rules, which, for every symbol in state *id*, rewrite this symbol and propagate the state *id*.). In this case, we can handle the composition with the rules shown on Figure 14. The idea is that we can split the rule of $N$ in order to allow recursive erasing between the two $\sigma'$.

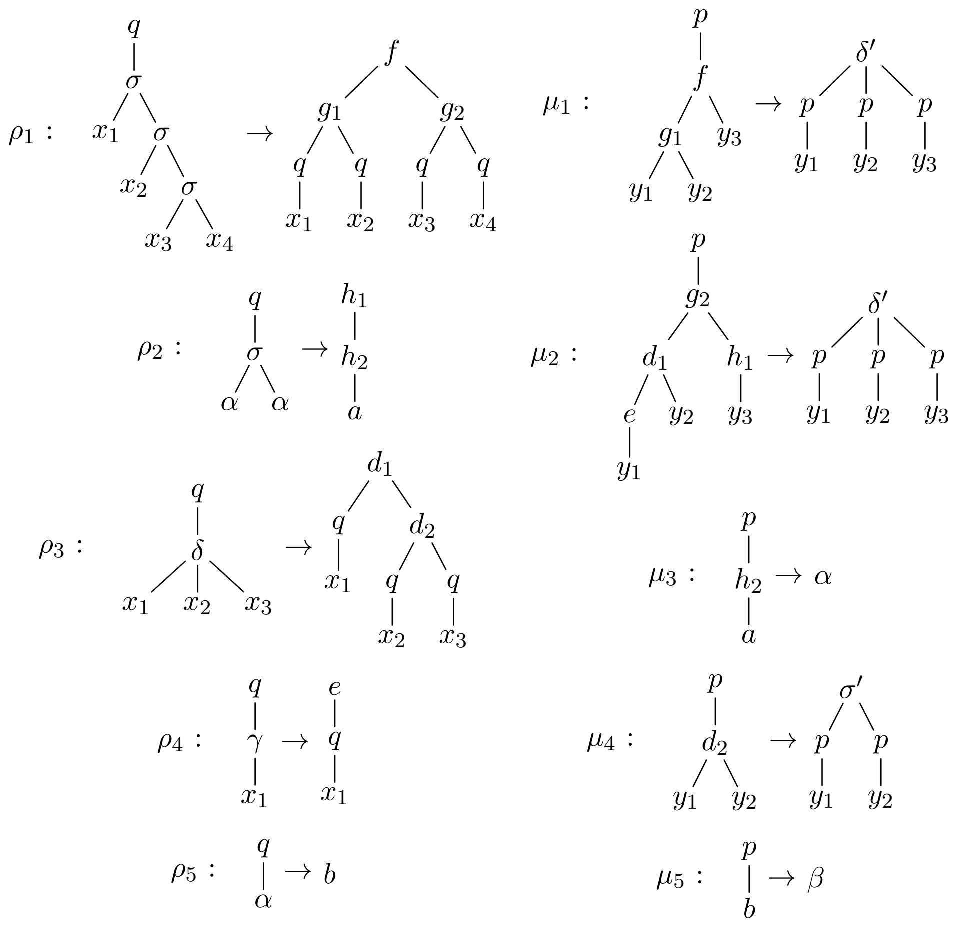
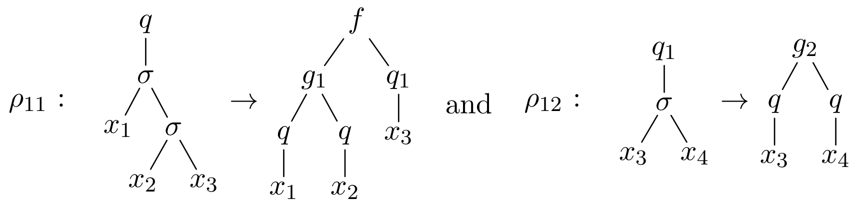
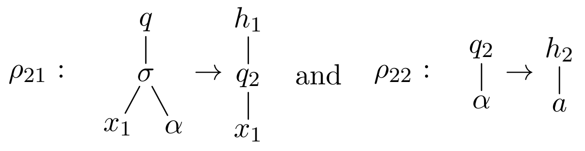


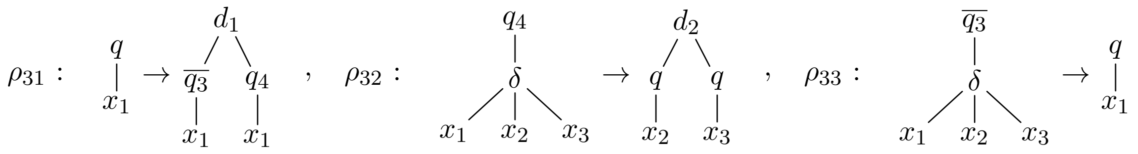
**Fig. 6.** Rules of $M_1$ and $M_2$



(a) Decomposition of $\rho_1$: $w = 1$, $w' = 122$, $A = \emptyset$

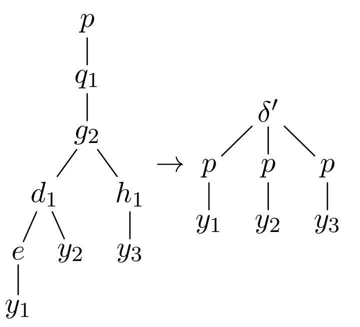


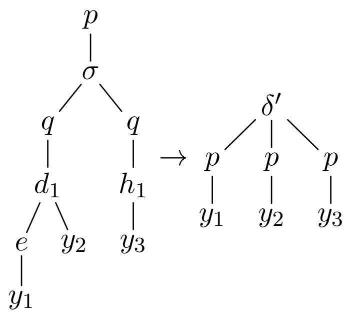
(b) Decomposition of $\rho_2$: $w = 1$, $w' = 11$, $A = \emptyset$

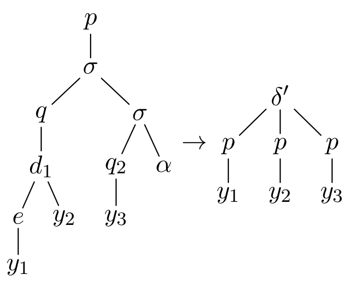


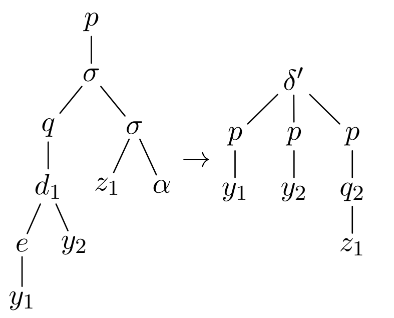
(c) Decomposition of $\rho_3$: $w = 1$, $w' = 1$, $A = \{x_1\}$

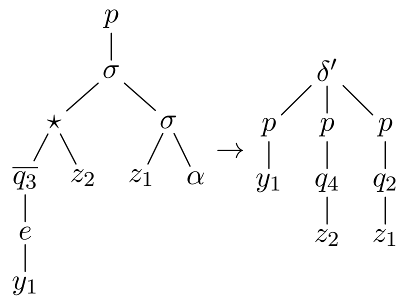
**Fig. 7.** Decomposition of the rules of $M_1$ to be compatible with $M_2$

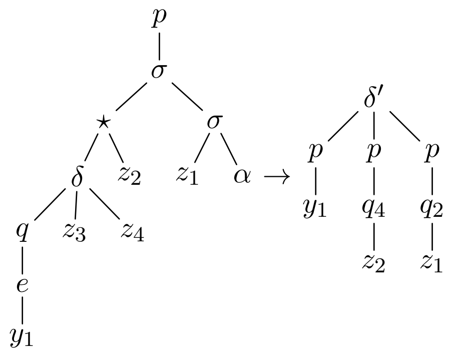

(a) Step 1: starting from $\mu_2$ and state $q_1$.

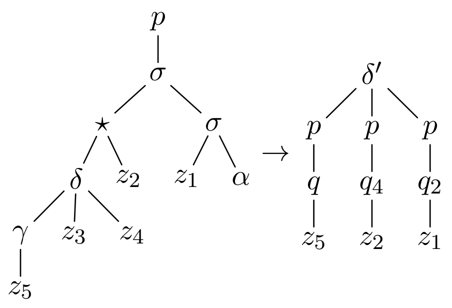

(b) Step 2: Case 2 of the procedure, choose $\rho_{12}$.

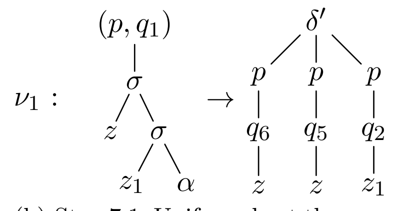

(c) Step 3 (beginning) : Case 2 of the procedure, choose $\rho_{21}$.

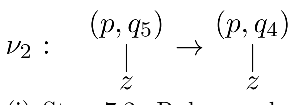

(d) Step 3 (end): $q_2(y_3)$ is replaced by a fresh variable $z_1$ in the left-hand side, and $y_3$ is replaced by $q_2(z_1)$ in the right-hand side.


(e) Step 4: Case 2 of the procedure, choose $\rho_{31}$.


(f) Step 5: Case 2 of the procedure, choose $\rho_{33}$.


(g) Step 6: Case 2 of the procedure, choose $\rho_4$.


(h) Step 7.1: Unify and get the new rule $\nu_1$. Then augment the root by $q_1$.


(i) Step 7.2: Rule $\nu_2$ obtained by the unification.

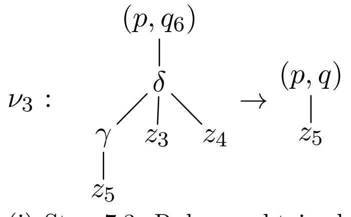

(j) Step 7.3: Rule $\nu_3$ obtained by the unification.

**Fig. 8.** Composition procedure run on $\mu_2$.

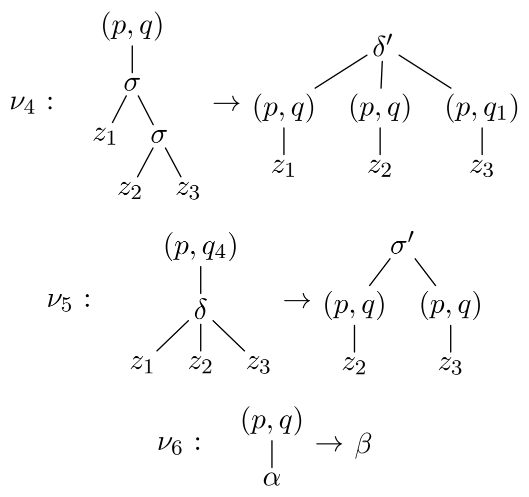


**Fig. 9.** Rules of $M_1; M_2$, in addition to $\nu_1$, $\nu_2$, $\nu_3$

*Example 14.* This example illustrates the third bullet. The shape of possible derivations is shown on Figure 15.

The rules of $M$ and $N$ are given, in addition to $id$ rules, by the following (some of them are illustrated on Figure 16):

| | |
|---|---|
| $\rho_1 : \quad q_0(\sigma(x_1, x_2)) \to \sigma'(q_{rec}(x_1), id(x_2))$ | $\mu_1 : p(\sigma'(\sigma'(y_1, y_2), y_3)) \to \delta(p_{rec}(y_1), id(y_2), id(y_3))$ |
| $\rho_2 : q_{rec}(\sigma(x_1, x_2)) \to \sigma'(q_{\alpha\beta}(x_1), id(x_2))$ | $\mu_2 : \quad p_{rec}(\gamma(x_1)) \to p_{rec}(x_1)$ |
| $\rho_3 : \quad q_{rec}(\gamma(x_1)) \to q_{rec}(x_1)$ | $\mu_3 : \quad p_{rec}(\alpha) \to \alpha'$ |
| $\rho_4 : \quad q_{\alpha\beta}(x_1) \to q_{\alpha\beta}(\gamma(x_1))$ | $\mu_4 : \quad p_{rec}(\beta) \to \beta'$ |
| $\rho_5 : \quad q_{\alpha\beta}(\alpha) \to \alpha$ | |
| $\rho_6 : \quad q_{\alpha\beta}(\beta) \to \beta$ | |

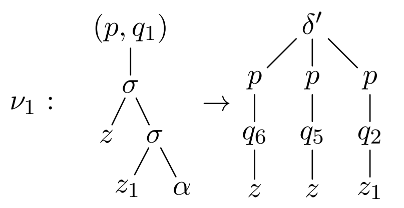
In this case, we can handle the composition with the rules shown on Figure 17. The idea is that we can predict the first son of $\delta$, and check the prediction thanks to a finite number of new states.

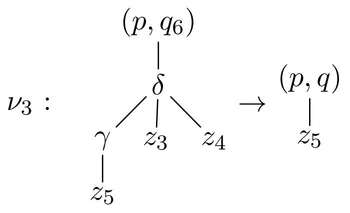
*Example 15.* This example illustrates the third bullet, when we need to split the new created rule. The shape of possible derivations is shown on Figure 18.

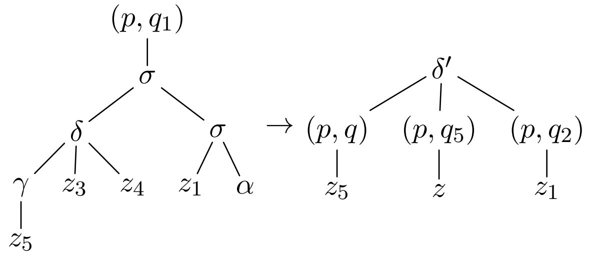
The rules of $M$ and $N$ are given, in addition to $id$ rules, by the following (some of them are illustrated on Figure 19):

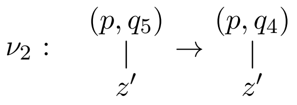
| | |
|---|---|
| $\rho_1 : \quad q_0(\sigma(x_1, x_2)) \to \sigma'(q_{rec}(x_1), id(x_2))$ | $\mu_1 : p(\sigma'(h(\sigma'(y_1, y_2), y_3))) \to \delta(p(y_1), id(y_2), id(y_3))$ |
| $\rho_2 : q_{rec}(\sigma_2(x_1, x_2)) \to \sigma'(q_\alpha(x_1), id(x_2))$ | $\mu_2 : \quad p(\alpha) \to \alpha'$ |
| $\rho_3 : \quad q_{rec}(\gamma(x_1)) \to q_{rec}(x_1)$ | |
| $\rho_4 : \quad q_{rec}(h(x_1)) \to h(q_{rec}(x_1))$ | |
| $\rho_5 : \quad q_\alpha(\alpha) \to \alpha$ | |

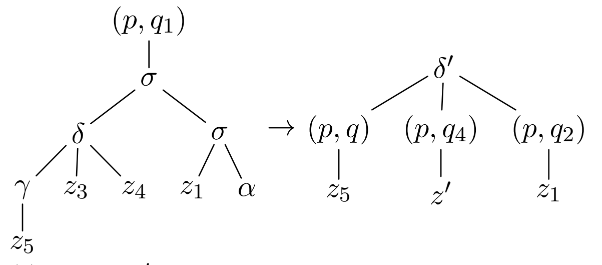
In this case, we can handle the composition with the following rules. The idea is that, on one hand we can predict the $\alpha$, and in the other hand, we need to allow recursive erasing between the $h$ and the $\sigma_2$.

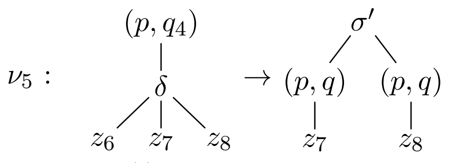

(a) Step 1: Starting from the copying rule $\nu_1$. $z$ is a copied variable, $w = 11$, $w' = 1$. Choose $\nu_3$ and let $\varphi(z) = 11$.

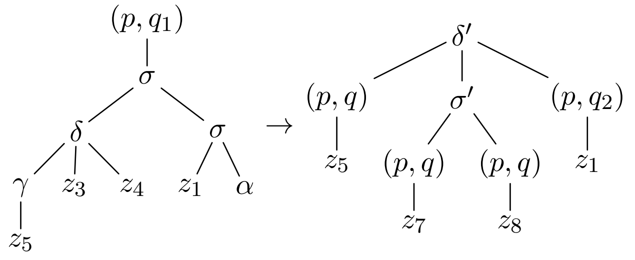

(b) Chosen rule $\nu_3$


(c) Step 2: $z$ is occurring only on the right-hand side, $w = 11$, $w' = 2$. Choose $\nu_2$ and let $\varphi(z') = 11$.


(d) Chosen rule $\nu_2$


(e) Step 3: $z'$ is occurring only on the right-hand side, $w = 11$, $w' = 2$. Choose $\nu_5$ and let $\varphi(z_7) = 112$ and $\varphi(z_8) = 113$.


(f) Chosen rule $\nu_5$


(g) Step 4: $z_7$ and $z_8$ are occuring only on the right-hand side. Positions $\varphi(z_7)$ and $\varphi(z_8)$ point at variables $z_3$ and $z_4$ respectively, so we replace $z_7$ and $z_8$ by these variables.

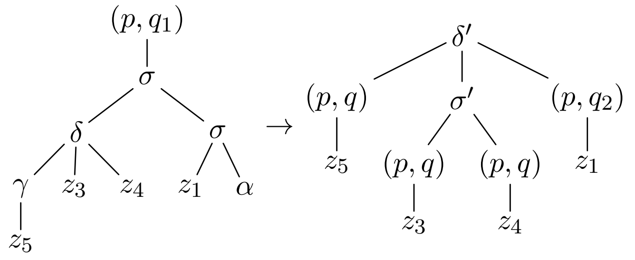

(h) Step 5: We end up with a linear and nondeleting rule.

**Fig. 10.** Recombining step run on $\nu_1$

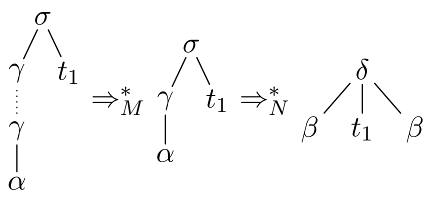


**Fig. 11.** Possible derivations in $M$ then $N$, see Example 12

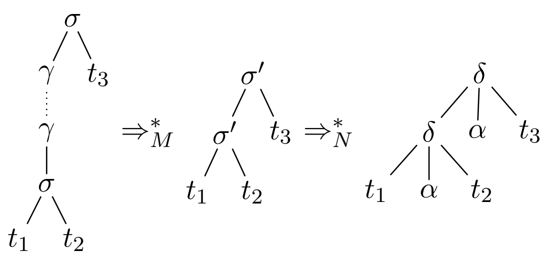
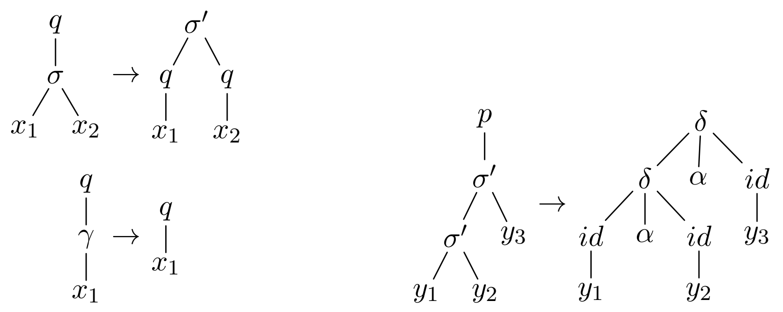


**Fig. 12.** Possible derivations in $M$ then $N$, see Example 13



(a) Rules of $M$ (b) Rules of $N$

**Fig. 13.** Illustration of the rules of $M$ and $N$, see Example 13

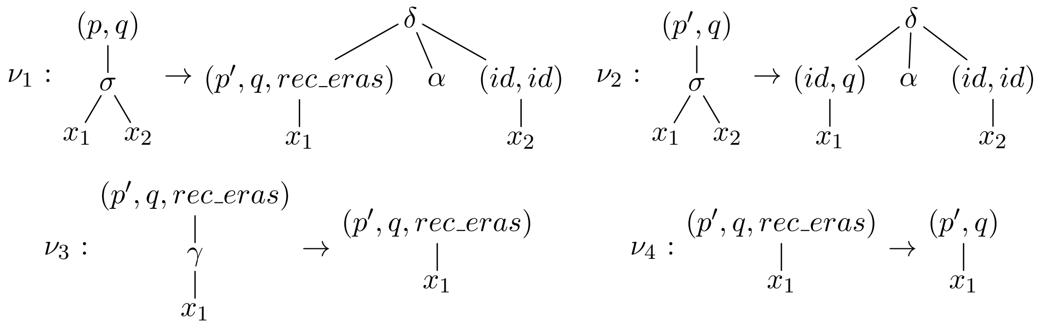


**Fig. 14.** Rules that handle the composition, see Example 13

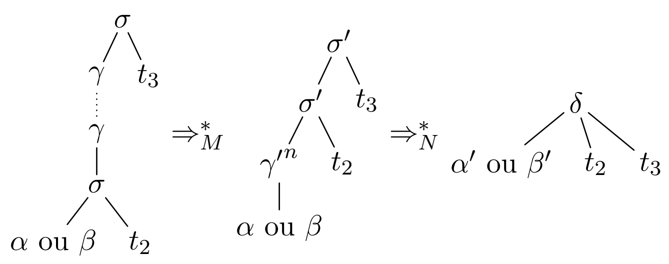


**Fig. 15.** Possible derivations in $M$ then $N$, see Example 14

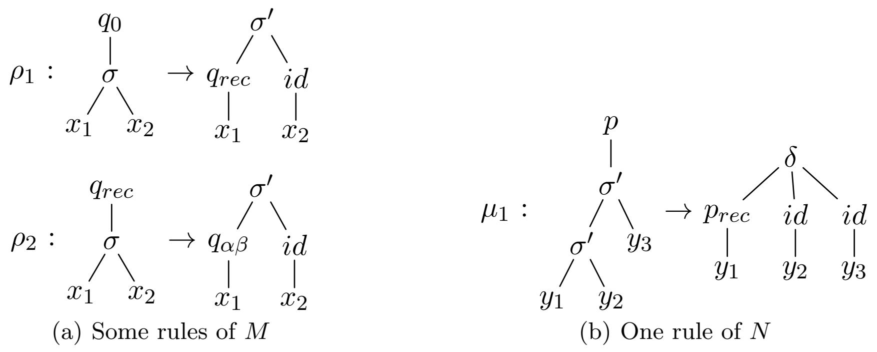
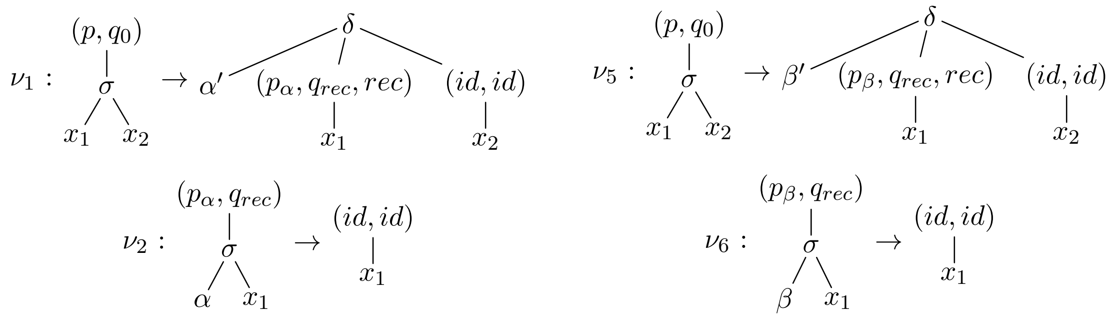


**Fig. 16.** Illustration of some rules of $M$ and $N$, see Example 14



$\nu_3 : (p_\alpha, q_{rec}, rec)(\gamma(x_1)) \to (p_\alpha, q_{rec}, rec)(x_1)$ $\quad$ $\nu_7 : (p_\beta, q_{rec}, rec)(\gamma(x_1)) \to (p_\beta, q_{rec}, rec)(x_1)$

$\nu_4 : (p_\alpha, q_{rec}, rec)(\gamma(x_1)) \to (p_\alpha, q_{rec})(x_1)$ $\quad$ $\nu_8 : (p_\beta, q_{rec}, rec)(\gamma(x_1)) \to (p_\beta, q_{rec})(x_1)$

**Fig. 17.** Rules that handle the composition, see Example 14

$$\begin{aligned}
\nu_1 :&\quad (p, q_0)(\sigma(x_1, x_2)) \to \delta(\alpha', (p_1, q_{rec}, \text{rec\_eras})(x_1), (id, id)(x_2))\\
\nu_2 :&\quad (p_1, q_{rec})(h(x_1)) \to h((p_2, q_{rec}, \text{rec\_eras})(x_1))\\
\nu_3 :&\quad (p_2, q_{rec})(\sigma_2(\alpha, x_1)) \to (id, id)(x_1)\\
\nu_4 :&\quad (p_1, q_{rec}, \text{rec\_eras})(\gamma(x_1)) \to (p_1, q_{rec}, \text{rec\_eras})(x_1)\\
\nu_5 :&\quad (p_1, q_{rec}, \text{rec\_eras})(\gamma(x_1)) \to (p_1, q_{rec})(x_1)\\
\nu_6 :&\quad (p_2, q_{rec}, \text{rec\_eras})(\gamma(x_1)) \to (p_2, q_{rec}, \text{rec\_eras})(x_1)\\
\nu_7 :&\quad (p_2, q_{rec}, \text{rec\_eras})(\gamma(x_1)) \to (p_2, q_{rec})(x_1)
\end{aligned}$$

The context $C$ was $h(\sigma_2(x_1, x_2))$ and had to be split at position 1.

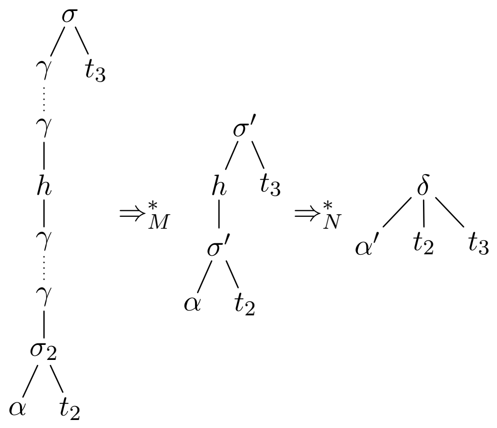


**Fig. 18.** Possible derivations in $M$ then $N$, see Example 15

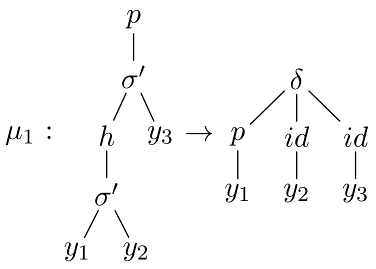


**Fig. 19.** Selected rule of $N$, see Example 15

## C Survey