Technische Universität Dresden
Fakultät Informatik

Diplomarbeit

mit dem Thema

Vergleich von Verfahren zur Elimination von Zwischenergebnissen bei funktionalen Programmen

Matthias Höff
Matrikelnummer 2368862

Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. Heiko Vogler
Lehrstuhl Grundlagen der Programmierung
Institut für Softwaretechnik I
Betreuer: Dr.-Ing. Armin Kühnemann

begonnen am 11.03.1999
abgegeben am 30.09.1999

Erklärung

Hiermit versichere ich, daß ich die vorliegende schriftliche Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen benutzt habe.

Dresden, am 30. 09. 1999

Matthias Höff

Inhaltsverzeichnis

Kapitel 1

Einleitung

Wenn man ein gegebenes Problem mit Hilfe eines Programms lösen möchte, muß man sich für eine geeignete Programmiersprache entscheiden. Dabei greift man oft auf imperative Programmiersprachen, wie zum Beispiel C oder Pascal, zurück. Hierbei muß der Programmierer mit einer Folge von Anweisungen der Sprache formulieren, wie das Problem gelöst werden soll.

Ist das Problem jedoch bereits hinreichend formal spezifiziert, so bieten sich besonders die deklarativen Programmiersprachen an. Diese erlauben es, die Spezifikation unter Beachtung der Syntax der Sprache direkt als Programm zu übernehmen. Solche Programme beschreiben dann also das Problem selbst, und wie es letztendlich gelöst wird, liegt ganz im Verantwortungsbereich der konkreten Implementation der jeweiligen Sprache.

Bei deklarativen Programmiersprachen unterscheidet man wiederum zwischen funktionalen Sprachen (z.B. Lisp, ML, Miranda und Haskell, zur Übersicht sei auf [Hud89] verwiesen) und Logikprogrammiersprachen (z.B. Prolog) sowie funktional-logischen Programmiersprachen, welche die Konzepte der beiden ersteren vereinigen. In dieser Arbeit befassen wir uns mit einem Problem aus der funktionalen Programmierung.

Die funktionale Programmierung basiert auf dem Lambda-Kalkül. Demnach sind die Hauptbestandteile funktionaler Programme Funktionsdefinitionen (Abstraktion) und deren Anwendungen (Applikation). Das wichtigste Programmierkonzept ist die Rekursion. Ein häufig anzutreffender modularer Programmierstil ist dadurch gekennzeichnet, daß Aufrufe von Funktionen zur Lösung von Teilproblemen so miteinander kombiniert werden, daß damit das Gesamtproblem gelöst wird. Dabei sind die als Zwischenergebnisse erzeugten Datenstrukturen die Bindeglieder zwischen den einzelnen Funktionen, indem sie als Eingabe der jeweils folgenden Funktion weiterverarbeitet werden. Die so entstehenden Programme sind oft sehr übersichtlich und verständlich, was die Entwicklung und Verifikation erleichtert (s. [Hug90]).

Eine zentrale Rolle kommt hierbei dem Datentyp der Liste zu. Funktionale Programmiersprachen bieten eine Vielzahl vordefinierter Funktionen zur Listenmanipulation an und unterstützen deren Benutzung noch durch spezielle syntaktische Konstrukte, wie z.B. list comprehension. Durch geschickte Kombination dieser Funktionen lassen sich eigene Funktionen definieren, die ebenfalls auf Listen arbeiten, wodurch sich viele Probleme elegant lösen lassen.

Betrachten wir zum Beispiel die Funktion `append l1 l2`, welche als Eingabe zwei Listen `l1` und `l2` nimmt und die zweite an die erste anhängt. In Haskell ([FHP97]) könnte die Definition folgendermaßen aussehen, wobei wie üblich `(x1:x2)` für den Listenkonstruktor `(Cons x1 x2)` steht und `[]` die leere Liste ist:

```
append (x1:r1) l2 = x1:(append r1 l2)
append []      l2 = l2
```

Wenn man nun drei Listen `l1`, `l2` und `l3` verketten möchte, kombiniert man einfach zwei Aufrufe von `append`:

```
(append (append l1 l2) l3)
```

Als Zwischenergebnis entsteht dabei die Aneinanderreihung von `l1` und `l2`, diese Liste wird dann durch den äußeren Funktionsaufruf wieder als Eingabe genommen und mit `l3` verkettet.

Unglücklicherweise hat dieser Programmierstil einen entscheidenden Nachteil: Die Programme sind nicht besonders effizient. Sind die Zwischenergebnisse strukturierte Objekte wie Listen oder Bäume, dann ist ihre Erzeugung und Konsumierung rechenzeit- und speicherplatzintensiv. Außerdem kann es z.B. passieren, daß Elemente mehrfach durchlaufen werden. Im Beispiel der `append`-Funktion wird die erste Liste zweimal durchlaufen, und das Ergebnis der Verkettung von `l1` und `l2` wird zunächst im Speicher angelegt und später wieder zerstört.

Oftmals gibt es für das jeweilige Problem aber effizientere Lösungsmöglichkeiten, die ohne Zwischenergebnisse auskommen. Für das Verketten von drei Listen könnte man zum Beispiel unter Benutzung von `append` eine neue Funktion `app3` programmieren, welche folgende Gestalt hat:

```
app3 (x1:r1) l2 l3 = x1:(app3 r1 l2 l3)
app3 [] l2 l3 = (append l2 l3)
```

Diese durchläuft die Liste `l1` nur einmal und erzeugt kein Zwischenergebnis. Das geht in der Regel jedoch zu Lasten der Lesbarkeit und Klarheit des Programms. Daher wäre es für den Programmierer wünschenswert, die erste Version schreiben zu können, die dann durch den Compiler automatisch in die effizientere Version transformiert wird. Diese Möglichkeit der automatischen Veränderung von Programmen zu Optimierungszwecken ist einer der Vorteile deklarativer Programmiersprachen.

Ein Beispiel für eine solche Transformation ist das bekannte Verfahren *Deforestation* von Wadler [Wad90]. Dieses Verfahren entfernt für eine eingeschränkte Klasse von Programmen beliebige als Zwischenergebnis vorkommende Datenstrukturen, insbesondere Listen und Bäume. Voraussetzung für dessen Anwendung ist, daß die umzuwandelnden Ausdrücke nur Funktionen enthalten, deren Definitionen in der sogenannten *treeless form* vorliegen. Dabei dürfen rekursive Funktionsaufrufe in den Argumenten nur Variablen enthalten, und die Regeln müssen linear in allen verwendeten Variablen sein. Durch diese Einschränkungen ist die Termination des Verfahrens gewährleistet, und es tritt kein Effizienzverlust auf. In der Regel wird sogar ein Gewinn erzielt.

Um das Verfahren für alle Programme erster Ordnung anwendbar zu machen, sind einige Anstrengungen unternommen worden. Die auf Wadler selbst zurückgehende Grundidee dabei ist, alle Teile des Programms zu markieren, die gegen die treeless-Eigenschaft verstoßen, und dann von der Transformation auszuschließen. Dies nennt man *blazing*. Außerdem finden sich in [HJ92] verschiedene Analysetechniken, welche einige dieser Markierungen verhindern, weil sie unnötig sind. Durch die Verhinderung einiger Transformationen entsteht jedoch das Problem, daß nicht mehr alle Zwischenergebnisse erreicht werden.

Ein völlig anderer Ansatz zur Optimierung funktionaler Programme stammt aus der Theorie der tree transducer (zur Übersicht siehe z.B. [FV98]). Dies sind formale Modelle der syntaxgesteuerten Semantik. Für uns besonders interessant sind die *macro tree transducer* ([Eng80, CF82]), welche eine Abstraktion der denotationalen Semantik mit Kontextinformationen darstellen. In [CF82] werden sie primitiv-rekursive Schemata mit Parametern genannt. Vom Standpunkt der funktionalen Programmierung aus sind es Systeme von simultan definierten Funktionen mit mehreren Parametern. Der erste (syntaktische) Parameter arbeitet auf der Eingabestruktur, was Bäume über einem Engabealphabet sind. Die übrigen (Kontext-)Parameter enthalten Bäume über einem Ausgabealphabet.

Ein macro tree transducer M übersetzt also Bäume über einem Eingaberangalphabet in Bäume über einem Ausgaberangalphabet. M benutzt dabei ein Rangalphabet von Funktionen mit dem Mindestrang 1 und eine Menge von Funktionsgleichungen. Jede Funktion ist ähnlich dem aus der funktionalen Programmierung bekannten pattern matching durch eine Fallunterscheidung über dem Wurzelsymbol c ihres ersten Arguments definiert. Die rechte Seite einer solchen definierenden Gleichung darf verschachtelte Funktionsaufrufe in den Kontextparametern von Funktionen enthalten, während das erste Argument stets eine Variable sein muß, die sich auf einen Nachfolger von c bezieht. Auf diese Weise repräsentieren macro tree transducer eine eingeschränkte Klasse von funktionalen Programmen erster Ordnung.

Die Verkettung von zwei Listen läßt sich beispielsweise durch einen macro tree transducer M_{app} realisieren. Da die sogenannten Rekursionsvariablen, welche Teilbäume des Rekursionsargumentes repräsentieren, ausschließlich im ersten Parameter von Funktionsaufrufen benutzt werden dürfen, müssen wir für alle n möglichen Listenelemente noch eine Identitätsfunktion id hinzufügen.

$$
\begin{array}{lllll}
M_{app}: & append\ (Cons\ x_1\ x_2)\ y_1 & = & Cons\ (id\ x_1)\ (append\ x_2\ y_1) \\
 & append\ Nil\ y_1 & = & y_1 \\
 & id\ Element_1 & = & Element_1 \\
 & & \vdots & \\
 & id\ Element_n & = & Element_n
\end{array}
$$

Wenn alle Funktionen von M den Rang 1 haben, so nennen wir M einen *top-down tree transducer* ([Rou70]). Für die Komposition eines macro tree transducers mit einem top-down tree transducer (und umgekehrt) wurde in [EV85] gezeigt, daß dafür ein einziger macro tree transducer konstruiert werden kann, welcher die selbe Berechnung durchführt. Weil dadurch die Erzeugung von Zwischenergebnissen vermieden wird, sind diese Kompositionsverfahren zur Optimierung funktionaler Programme geeignet.

Für lineare tree transducer wurden diese Verfahren in [Küh99] bereits qualitativ mit Deforestation verglichen. Dabei wurde festgestellt, daß es Programme gibt, für welche die Komposition die Zwischenergebnisse entfernt, Deforestation aber nicht. Der nichtlineare Fall wurde dort jedoch ausgeklammert, weil es dabei durch die Komposition (und natürlich auch durch Deforestation) zu Effizienzverlusten kommen kann, wie das folgende Beispiel zeigt. Darin betrachten wir zwei top-down tree transducer, wobei die Funktion f zu M_1 und die Funktion g zu M_2 gehört. M_2 besitzt eine Regel, welche nichtlinear ist, was durch Unterstreichung der doppelt vorkommenden Variable verdeutlicht wird.

$$
\begin{array}{llll}
M_1: & f\ (\delta\ x_1) & = & \delta\ (f\ x_1) \\
 & f\ (\gamma\ x_1) & = & \boxed{f\ x_1} \\
 & f\ \alpha & = & \alpha
\end{array}
\qquad
\begin{array}{llll}
M_2: & g\ (\delta\ x_1) & = & \sigma\ (g\ \underline{x_1})\ (g\ \underline{x_1}) \\
 & g\ \alpha & = & \alpha
\end{array}
$$

Nun komponieren wir die tree transducer derart, daß die Funktion f von M_1 auf einen Eingabebaum angewendet wird und das dabei erzeugte Zwischenergebnis durch die Funktion g von M_2 weiterverarbeitet wird. Dafür läßt sich ein neuer top-down tree transducer M konstruieren, der die gleiche Berechnung ohne Erzeugung eines Zwischenergebnisses durchführt. Die Funktion $\overline{fg}$ repräsentiert dabei die Hintereinanderausführung der Funktionen f und g in einem Schritt.

$$\begin{array}{llcl} M\colon & \overline{fg}\,(\delta\ x_1) & = & \sigma\ (\overline{fg}\ x_1)\ (\overline{fg}\ x_1) \\ & \overline{fg}\,(\gamma\ x_1) & = & \overline{fg}\ x_1 \\ & \overline{fg}\ \alpha & = & \alpha \end{array}$$

In der folgenden Abbildung ist ein möglicher Eingabebaum mit dem bei der Verarbeitung durch M_1 entstehende Zwischenergebnis und dem nach Konsumierung desselben durch M_2 erzeugten Endergebnis dargestellt. Außerdem ist die Anzahl der Reduktionsschritte eingetragen, welche die beiden tree transducer sowie der neue tree transducer für die Rechenvorgange benötigen.

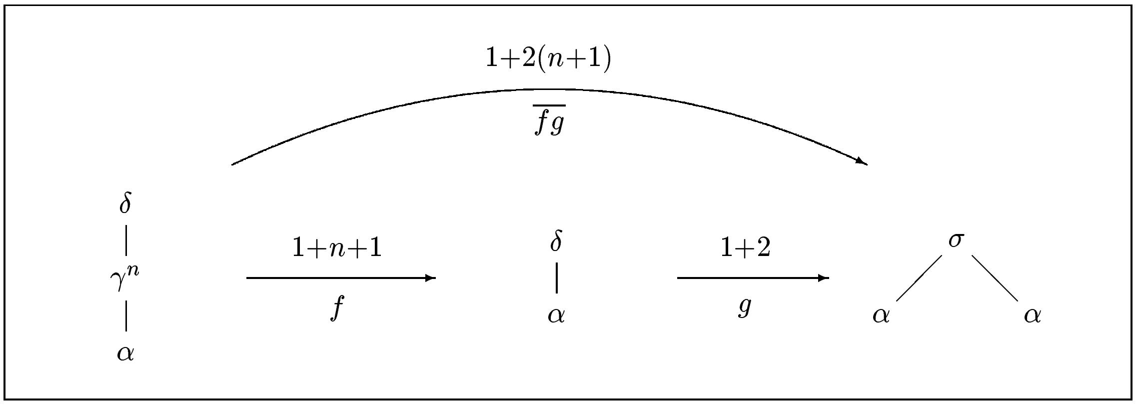

Abbildung 1.1: Effizienzverlust bei der Komposition nichtlinearer tree transducer

Wir erkennen, daß ab $n = 3$ ein Effizienzverlust durch die Komposition entsteht. Das liegt daran, daß M_1 beim Durchlauf durch den Eingabebaum entsprechend der eingerahmten Regel die γ-Symbole entfernt, so daß die wegen der Nichtlinearität von M_2 erforderlichen zwei Durchläufe durch das Zwischenergebnis mit konstanter Reduktionsschrittanzahl durchgeführt werden. Nach der Komposition sind diese beiden Durchläufe jedoch schon im Eingabebaum nötig, in welchem die γ-Symbole noch vorhanden sind. Dadurch steigt die Zahl der Reduktionsschritte mit zunehmender Anzahl solcher Symbole.

Wir wollen in dieser Arbeit die Untersuchungen dennoch auf den nichtlinearen Fall ausweiten. Durch quantitative Betrachtungen der ungewichteten Zahl der Reduktionsschritte bei Auswertung mittels call-by-need, die wir als Effizienzmaß zugrundelegen, können wir Aussagen zu den Effizienzsteigerungen oder -verlusten treffen. Dabei stellen wir fest, daß die Nichtlinearität des ersten tree transducers keine negativen Auswirkungen hat. Weil sich die Verfahren für die beiden Kompositionsmöglichkeiten unterscheiden, behandeln wir diese getrennt. Dabei nehmen wir jeweils weitere syntaktischen Einschränkungen vor, wobei sich dann verschiedene Resultate ergeben.

Bei der Komposition von top-down tree transducern mit macro tree transducern beweisen wir, daß für diesen Fall Deforestation und die Komposition äquivalent sind. Dann schränken wir aus beweistechnischen Gründen die macro tree transducer auf solche ein, welche sämtliche Kontextparameter bei der Berechnung verwenden. Dabei erhalten wir das Resultat, daß ein Effizienzverlust nur dann auftreten kann, wenn der macro tree transducer nichtlinear ist und nicht alle Regeln des top-down tree transducers Zwischenergebnisse erzeugen. In den übrigen Fällen wird ein Effizienzgewinn erzielt. Danach versuchen wir, diese Beobachtungen für macro tree transducer ohne Einschränkungen zu verallgemeinern.

Für die Komposition von macro tree transducern mit top-down tree transducern unterscheiden sich Komposition und Deforestation. Auch hier betrachten wir zunächst nur macro tree transducer, welche keine Kontextparameter während der Berechnung löschen. Dabei ist es erforderlich, den top-down tree transducer auf eine Funktion zu beschränken und zu fordern, daß er keinen Teil des Zwischenergebnisses ignoriert. Im Falle der Nichtlinearität des top-down tree transducers ist Deforestation erheblich anfälliger gegenüber einem Effizienzverlust als die Komposition. Ist der zweite tree transducer jedoch linear, sind keine Effizienzverluste möglich, wobei die Komposition mindestens so gut ist wie Deforestation, in vielen Fällen sogar besser.

Die Arbeit ist in 5 Kapitel gegliedert. Nach dieser Einleitung führen wir im Kapitel 2 zunächst einige grundlegende Begriffe ein, die wir dann im weiteren Verlauf der Arbeit verwenden, darunter macro tree transducer und die Verfahren Komposition und Deforestation. Im 3. Kapitel zeigen wir für die Komposition von top-down tree transducern mit macro tree transducern die Gleichheit von Komposition und Deforestation und untersuchen die dadurch erzielten Effizienzsteigerungen. Im 4. Kapitel stehen der Vergleich von Komposition und Deforestation für die umgekehrte Komposition sowie die Untersuchung der dabei auftretenden Effizienzveränderungen im Mittelpunkt. Kapitel 5 beschließt die Arbeit mit dem Ausblick.

Kapitel 2

Grundbegriffe

In diesem Kapitel wollen wir die in der Arbeit benötigten grundlegenden Begriffe, Notationen und Definitionen einführen.

2.1 Allgemeine Begriffe und Definitionen

Die Menge $\{0, 1, 2, \ldots\}$ der natürlichen Zahlen bezeichnen wir mit $\mathbb{N}$, die leere Menge mit $\emptyset$. Für alle $n \in \mathbb{N}$ kürzen wir die Menge $\{1, 2, \ldots, n\}$ mit $[n]$ ab, demnach gilt $[0] = \emptyset$. Ist K eine nichtleere, endliche Teilmenge von $\mathbb{N}$, so ist $max(K)$ der größte Wert aus K. Weiterhin verwenden wir die Mengen $X = \{x_1, x_2, \ldots\}$, $Y = \{y_1, y_2, \ldots\}$, $Z = \{z_0, z_1, z_2, \ldots\}$ und $V = X \cup Y \cup Z$, deren Elemente wir *Rekursionsvariablen*, *Kontextvariablen*, *Ausdrucksvariablen* bzw. *Variablen* nennen. Für die Mengen $\{x_1, \ldots, x_n\}$ und $\{y_1, \ldots, y_n\}$ mit $n \geq 0$ schreiben wir auch X_n bzw. Y_n. Dabei gilt insbesondere $X_0 = Y_0 = \emptyset$.

Seien A und B Mengen. Mit $card(A)$ bezeichnen wir die Mächtigkeit von A und mit $\mathcal{P}(A)$ die Potenzmenge von A. Eine *Relation* von A nach B ist eine Teilmenge R des kartesischen Produkts $A \times B$. Für $(a, b) \in R$ schreiben wir aRb. Eine Relation von A nach A nennen wir auch *Relation auf A*. Wenn für jedes $a \in A$ genau ein $b \in B$ existiert, so daß aRb gilt, so nennen wir R *Abbildung* oder *Funktion* vom Typ $R : A \to B$ und schreiben $R(a) = b$.

Sei $\Rightarrow$ eine Relation auf A. Mit $\Rightarrow^n$ und $\Rightarrow^*$ bezeichnen wir die n-fache Komposition bzw. die reflexive, transitive Hülle von $\Rightarrow$. Eine Sequenz $a_1 \Rightarrow a_2 \Rightarrow \ldots \Rightarrow a_{n+1}$ mit $n \geq 1$ und $a_1, \ldots, a_{n+1} \in A$ wird *Ableitung* genannt, falls $a_i \Rightarrow a_{i+1}$ für alle $i \in [n]$ gilt. Falls $a \Rightarrow^* a'$ für $a, a' \in A$ gilt und kein $a'' \in A$ mit $a' \Rightarrow a''$ existiert, dann heißt a' eine *Normalform von a bezüglich* $\Rightarrow$. Wenn diese existiert und eindeutig ist, dann bezeichnen wir sie mit $nf(\Rightarrow, a)$ oder einfach $nf(a)$, falls $\Rightarrow$ aus dem Kontext hervorgeht. $\Rightarrow$ heißt *terminierend*, sofern es keine unendlich langen Ableitungen gibt. $\Rightarrow$ heißt *konfluent*, wenn für alle $a, a_1, a_2 \in A$ mit $a \Rightarrow^* a_1$ und $a \Rightarrow^* a_2$ ein $a' \in A$ existiert, so daß $a_1 \Rightarrow^* a'$ und $a_2 \Rightarrow^* a'$ gilt.

Ein *Alphabet* ist eine nichtleere, endliche Menge; die Elemente dieser Menge nennen wir *Symbole*. Ein *Wort* über einem Alphabet Σ ist eine endliche Symbolfolge der Form $\sigma_1 \ldots \sigma_n$ mit $\sigma_1, \ldots, \sigma_n \in \Sigma$ und $n \geq 0$. Für $n = 0$ ergibt sich das *leere Wort* ε. Die Menge aller Wörter über Σ wird mit Σ^* bezeichnet.

Ist w ein Wort und sind $w_1, \ldots, w_n$ und $w'_1, \ldots, w'_n$ zwei Listen von Wörtern, so daß sich für alle $i, j \in [n]$ mit $i \neq j$ die Wörter w_i und w_j in w nicht überlappen, so bezeichnet $w[w_1/w'_1, \ldots, w_n/w'_n]$ das Wort, das aus w durch Ersetzung jedes Vorkommens von w_i durch w'_i für alle $i \in [n]$ entsteht. Für die Substitution $[w_1/w'_1, \ldots, w_n/w'_n]$ verwenden wir die abkürzende Schreibweise $[w_i/w'_i \mid w_i \in \{w_1, \dots, w_n\}, w'_i \in \{w'_1, \dots, w'_n\}]$ oder einfach $[w_i/w'_i]$, wenn die Quantifizierung aus dem Kontext hervorgeht.

2.2 Rangalphabete, Bäume und Funktionen auf Bäumen

Ein *Rangalphabet* ist ein Paar $(\Sigma, rank)$, wobei Σ ein Alphabet und $rank : \Sigma \to I\!N$ eine Abbildung ist, die jedem $\sigma \in \Sigma$ eine natürliche Zahl $rank(\sigma)$ zuordnet. Wenn $rank(\sigma) = n$, dann hat σ den *Rang* oder die *Stelligkeit* n, und wir schreiben auch $\sigma^{(n)}$. Für jedes $n \geq 0$ definieren wir $\Sigma^{(n)} = \{\sigma \in \Sigma \mid rank(\sigma) = n\}$. Falls die Funktion *rank* aus dem Kontext hervorgeht, so wird sie nicht explizit angegeben. Ein Rangalphabet Σ ist *monadisch*, wenn $\Sigma = \Sigma^{(0)} \cup \Sigma^{(1)}$.

Sei Σ ein Rangalphabet und $V' \subseteq V$ mit $\Sigma \cap V' = \emptyset$. Dann ist die Menge der *mit V' indizierten Bäume über* Σ, abgekürzt mit $T_\Sigma(V')$, die kleinste Teilmenge $T \subseteq (\Sigma \cup V' \cup \{(,)\})^*$, für die gilt:

- $\Sigma^{(0)} \cup V' \subseteq T$ und
- für alle $k \geq 1, \sigma \in \Sigma^{(k)}$ und $t_1, \dots, t_k \in T$ ist $(\sigma\ t_1 \dots t_k) \in T$.

Das äußerste Klammerpaar eines Baumes kann weggelassen werden. Die Menge $T_\Sigma(\emptyset)$ wird mit T_Σ abgekürzt. Für Bäume über einem monadischen Rangalphabet benutzen wir eine einfachere Notation. Statt $\sigma_1(\sigma_2(\dots(\sigma_n\sigma)\dots))$ schreiben wir $\sigma_1\sigma_2\dots\sigma_n\sigma$, wobei $n \geq 1$, $\sigma_1, \dots, \sigma_n \in \Sigma^{(1)}$ und $\sigma \in \Sigma^{(0)}$. Für n gleiche Symbole σ mit $n \in I\!N$ schreiben wir σ^n.

Wir definieren im folgenden einige Funktionen zur Charakterisierung eines Baumes $t \in T_\Sigma(V')$ induktiv über seine Struktur: $occ : T_\Sigma(V') \to \mathcal{P}(I\!N^*)$ liefert die *Menge der Pfade in t*, $|.|_{\Sigma'} : T_\Sigma(V') \to I\!N$ berechnet die *Anzahl der Vorkommen von Symbolen aus einer Teilmenge Σ' von Σ in t* und $var : T_\Sigma(V') \to \mathcal{P}(V')$ gibt die *Menge der in t verwendeten Variablen* zurück.

- Falls $t \in \Sigma^{(0)} \cup V'$, dann
  - $occ(t) = \{\varepsilon\}$
  - $|t|_{\Sigma'} = \begin{cases} 0 & \text{falls } t \notin \Sigma' \\ 1 & \text{falls } t \in \Sigma' \end{cases}$
  - $var(t) = \begin{cases} \emptyset & \text{falls } t \in \Sigma^{(0)} \\ \{t\} & \text{falls } t \in V' \end{cases}$
- Falls $t = (\sigma\ t_1 \dots t_k)$ mit $k \geq 1, \sigma \in \Sigma^{(k)}$ und $t_1, \dots, t_k \in T_\Sigma(V')$, dann
  - $occ(t) = \{\varepsilon\} \cup \{iw \mid i \in [k], w \in occ(t_i)\}$
  - $|t|_{\Sigma'} = \begin{cases} \sum_{i=1}^{k} |t_i|_{\Sigma'} & \text{falls } \sigma \notin \Sigma' \\ 1 + \sum_{i=1}^{k} |t_i|_{\Sigma'} & \text{falls } \sigma \in \Sigma' \end{cases}$
  - $var(t) = \bigcup_{i=1}^{k} var(t_i)$.

Bei $\Sigma' = \Sigma$ schreiben wir $|t|$ statt $|t|_\Sigma$ und nennen diesen Ausdruck die *Größe des Baumes t*. Außerdem definieren wir $label : \{(o, t) \mid t \in T_\Sigma(V'), o \in occ(t)\} \to \Sigma \cup V'$ als die *Beschriftung des Knotens in einem Baum t, der durch einen Pfad $o \in occ(t)$ erreicht wird* und $subtree : \{(o, t) \mid t \in T_\Sigma(V'), o \in occ(t)\} \to T_\Sigma(V')$ als den *durch einen Pfad $o \in occ(t)$ erreichten Teilbaum von t* wie folgt:

- Falls $t \in \Sigma^{(0)} \cup V'$ und $o = \varepsilon$, dann
  - $label(o, t) = t$
  - $subtree(o, t) = t$
- Falls $t = (\sigma\, t_1 \ldots t_k)$ und $o = \varepsilon$ mit $k \geq 1, \sigma \in \Sigma^{(k)}$ und $t_1, \ldots, t_k \in T_\Sigma(V')$, dann
  - $label(o, t) = \sigma$
  - $subtree(o, t) = t$
- Falls $t = (\sigma\, t_1 \ldots t_k)$ und $o = io'$ mit $k \geq 1$, $\sigma \in \Sigma^{(k)}$, $t_1, \ldots, t_k \in T_\Sigma(V')$, $o' \in occ(t)$ und $i \in [k]$, dann
  - $label(o, t) = label(o', t_i)$
  - $subtree(o, t) = subtree(o', t_i)$

Da Bäume auch Zeichenketten sind, ist die dafür beschriebene Substitution von Teilzeichenketten ebenfalls möglich. Ein häufiger Anwendungsfall ist die Ersetzung von Variablen in einem Baum $t \in T_\Sigma(V')$ durch andere Bäume.

2.3 Macro Tree Transducer und funktionale Programme

2.3.1 Grundlegende Definitionen

In diesem Abschnitt führen wir macro tree transducer ein, welche als spezielle funktionale Programme angesehen werden können. Um eine lesbarere Definition zu erhalten, beschreiben wir zunächst das Aussehen der rechten Seiten der Regeln eines macro tree transducers.

Definition 2.1 Seien F und C disjunkte Rangalphabete und $k, n \geq 0$. Wir definieren die Menge $RHS(F, C, k, n)$ der *rechten Seiten über F, C, k und n* als die kleinste Teilmenge RHS von $T_{F \cup C}(X_k \cup Y_n)$, die folgende Bedingungen erfüllt:

(i) $Y_n \subseteq RHS$.

(ii) Für jedes $l \geq 0$, $c \in C^{(l)}$ und $r_1, \ldots, r_l \in RHS$ ist auch $(c\, r_1 \ldots r_l)$ in RHS.

(iii) Für jedes $l \geq 0$, $f \in F^{(1+l)}$, $r_1, \ldots, r_l \in RHS$ und $i \in [k]$ ist auch $(f\, x_i\, r_1 \ldots r_l)$ in RHS. □

Die Bäume aus der Menge $RHS(F, C, k, n)$ bestehen also aus Kontextvariablen mit dem maximalen Index n, Konstruktorsymbolen aus der Menge C sowie Symbolen aus der Menge F, wobei deren erster Nachfolger eine Rekursionsvariable mit maximalem Index k ist. Nun können wir die Definition für macro tree transducer angeben. Abweichend von der üblichen Notation in der Literatur (z.B. [FV98]) verzichten wir auf die initiale Funktion.

Definition 2.2 Ein *macro tree transducer* ist ein Tupel $M = (F, C^i, C^o, R)$ mit folgenden Eigenschaften:

- F ist ein Rangalphabet, dessen Elemente *Funktionen* oder *Zustände* heißen und einen Rang von mindestens 1 haben.

- C^i und C^o sind Rangalphabete mit $F \cap (C^i \cup C^o) = \emptyset$. Sie heißen *Eingabealphabet* bzw. *Ausgabealphabet* und enthalten die *Eingabe-* bzw. *Ausgabesymbole*. Diese Symbole nennen wir auch *Konstruktoren*.

- R ist eine endliche Menge von *Regeln* oder *Funktionsgleichungen*, die für jedes $c \in C^{i(k)}$ mit $k \geq 0$ und $f \in F^{(1+n)}$ mit $n \geq 0$ genau eine Regel der Form

$$f(c\ x_1 \ldots x_k)\ y_1 \ldots y_n = r$$

enthält, wobei $r \in RHS(F, C^o, k, n)$. Diese Regel nennen wir (f,c)-Regel, und die rechte Seite r dieser Regel bezeichnen wir mit $rhs(f,c)$. □

Die Menge aller macro tree transducer bezeichnen wir mit MAC. Man beachte, daß dies im Gegensatz zu der sonst in der Theorie der tree transducer betrachteten semantischen Klasse (die Klasse von Baumübersetzungen, die durch macro tree transducer berechnet werden können, siehe z.B. [FV98]) eine syntaktische Klasse ist. Einen macro tree transducer $M = (F, C^i, C^o, R)$ mit $F = F^{(1)}$ nennen wir *top-down tree transducer*. Die Klasse aller top-down tree transducer bezeichnen wir mit TOP.

Wir werden in dieser Arbeit einen macro tree transducer M nur durch Angabe seiner Regelmenge R spezifizieren. Die Menge der darin vorkommenden Funktionen bezeichnen wir mit F_M, die Menge der Konstruktorsymbole der linken Seiten mit C^i_M und die Menge der Konstruktorsymbole aus den rechten Seiten mit C^o_M. Dann gilt $M = (F_M, C^i_M, C^o_M, R)$.

Beispiel 2.3 Wir definieren die macro tree transducer M_{bsp} und M_{ex} durch Angabe ihrer Regelmengen.

$$\begin{array}{lllllll}
M_{bsp}: & f\ (\delta\ x_1)\ y_1 & = & \sigma\ (f\ x_1\ y_1)\ (f\ x_1\ y_1) & M_{ex}: & ex\ (A\ x_1) = & B\ (ex\ x_1) \\
& f\ (\gamma\ x_1)\ y_1 & = & \gamma\ (f\ x_1\ (f\ x_1\ y_1)) & & ex\ (B\ x_1) = & A\ (ex\ x_1) \\
& f\ \alpha_0\ y_1 & = & \alpha & & ex\ N = & N \\
& f\ \alpha_1\ y_1 & = & y_1 & & & \\
& f\ \alpha_2\ y_1 & = & \sigma\ y_1\ y_1 & & &
\end{array}$$

M_{bsp} ist ein macro tree transducer mit einem Kontextparameter. Für ein Eingabesymbol δ wird ein zweistelliges Symbol σ erzeugt, wodurch aus einem monadischen Baum ein Binärbaum erzeugt wird. Für ein Eingabesymbol γ wird dieses entweder identisch repliziert, wenn der Baum mit α_0 endet, ansonsten aber exponentiell vervielfacht.

M_{ex} ist ein top-down tree transducer mit $F_{M_{ex}} = \{ex^{(1)}\}$. Dieser ersetzt in einer Liste jedes Element A durch B und umgekehrt. Listen werden hierbei durch monadische Bäume repräsentiert, wobei mögliche Listenelemente A und B sind und die leere Liste durch N repräsentiert wird. □

Nun wollen wir einige syntaktische Einschränkungen für macro tree transducer definieren.

Definition 2.4 Sei $M = (F, C^i, C^o, R)$ ein macro tree transducer. Weiterhin sei $f \in F^{(1+n)}$ mit $n \geq 0$ und $c \in C^{i(k)}$ mit $k \geq 0$. Die (f,c)-Regel heißt

- *linear im Rekursionsargument*, wenn $|rhs(f,c)|_{\{x_i\}} \leq 1$ für alle $i \in [k]$ gilt, ansonsten *nichtlinear im Rekursionsargument*,

- *linear in den Kontextparametern*, wenn $|rhs(f,c)|_{\{y_j\}} \leq 1$ für alle $j \in [n]$ gilt, andernfalls *nichtlinear in den Kontextparametern*,

- *linear*, wenn sie linear im Rekursionsargument und in den Kontextparametern ist, sonst *nichtlinear*,
- *nondeleting im Rekursionsargument*, wenn $|rhs(f,c)|_{\{x_i\}} \geq 1$ für alle $i \in [k]$ gilt, andernfalls *deleting im Rekursionsargument*,
- *nondeleting in den Kontextparametern*, wenn $|rhs(f,c)|_{\{y_j\}} \geq 1$ für alle $j \in [n]$ gilt, andernfalls *deleting in den Kontextparametern*,
- *nondeleting*, wenn sie nondeleting im Rekursionsargument und in den Kontextparametern ist, sonst *deleting*,
- *producing*, wenn $|rhs(f,c)|_{C^o} \geq 1$ gilt, ansonsten *nonproducing*.

M heißt *linear*, *nondeleting im Rekursionsargument*, *nondeleting in den Kontextparametern* bzw. *producing*, wenn alle Regeln aus R die jeweilige Eigenschaft haben, ansonsten heißt M *nichtlinear*, *deleting im Rekursionsargument*, *deleting in den Kontextparametern* bzw. *nonproducing*. M heißt *aus einer Funktion bestehend*, wenn $card(F) = 1$, ansonsten *aus mehreren Funktionen bestehend*. Die durch diese Einschränkungen erzeugten Teilmengen der Klassen MAC und TOP kennzeichnen wir durch Voranstellen eines entsprechenden Kürzels. Diese lauten *lin*, *ndr*, *ndc*, *prod* und $1f$ bzw. *nlin*, *dr*, *dc*, *nprod* und nf. □

Beispiel 2.5 Wir betrachten die tree transducer aus Beispiel 2.3 und zeigen daran die verschiedenen Eigenschaften. Die (f,δ)-Regel von M_{bsp} ist nichtlinear im Rekursionsargument, producing und nondeleting im Kontextparameter, die (f,γ)-Regel ebenfalls. Die (f,α_0)-Regel ist deleting im Kontextparameter, die (f,α_1)-Regel nondeleting und linear im Kontextparameter und die (f,α_2)-Regel nondeleting und nichtlinear im Kontextparameter.

Der top-down tree transducer M_{ex} besteht nur aus einer Funktion und ist in allen Regeln linear und nondeleting im Rekursionsargument sowie producing, weshalb $M_{ex} \in 1f$*-lin-ndr-prod-*TOP gilt. □

Die Regeln eines macro tree transducers interpretieren wir als Funktionsdefinitionen einer einfachen funktionalen Programmiersprache erster Ordnung. Ein *Programm* in dieser Sprache wird von einem oder mehreren macro tree transducern gebildet. Die Menge aller solcher Programme bezeichnen wir mit P. Für ein Programm $p \in P$ bezeichnen wir die Menge aller in diesem Programm auftretenden Funktionen und Konstruktoren mit F_p bzw. C_p. Das in Definition 2.1 festgelegte Aussehen der rechten Seiten der Funktionsgleichungen unserer Programmiersprache mag auf dem ersten Blick zwar ziemlich restriktiv erscheinen, aber in der Praxis haben viele nützliche Funktionen über Listen und Bäumen diese Struktur.

Beispiel 2.6 Die aus der funktionalen Programmierung bekannten Funktionen `reverse` und `append` lassen sich als macro tree transducer darstellen. Wir definieren M_{rev} und M_{app} durch Angabe ihrer Regelmengen.

$$\begin{array}{llcl} M_{rev}: & rev\ (A\ x_1)\ y_1 & = & rev\ x_1\ (A\ y_1) \\ & rev\ (B\ x_1)\ y_1 & = & rev\ x_1\ (B\ y_1) \\ & rev\ N\ y_1 & = & y_1 \end{array}$$

$$\begin{array}{llcl} M_{app}: & app\ (A\ x_1)\ y_1 & = & A\ (app\ x_1\ y_1) \\ & app\ (B\ x_1)\ y_1 & = & B\ (app\ x_1\ y_1) \\ & app\ N\ y_1 & = & y_1 \end{array}$$

Listen werden hierbei durch monadische Bäume repräsentiert, wobei mögliche Listenelemente A und B sind und die leere Liste durch N repräsentiert wird. □

Deforestation transformiert im allgemeinen ein Programm $p \in P$ in ein neues Programm p', in dessen rechten Regelseiten Funktionsaufrufe der Form $f'\ y_j\ r_1 \ldots r_n$ auftreten können. Dabei steht im Rekursionsargument eine Kontextvariable anstelle einer Rekursionsvariablen. Die Menge aller Programme dieser Zielsprache, welche also Funktionen enthalten können, die aufgrund solcher Regeln zu keinem macro tree transducer gehören, bezeichnen wir mit P^+. Für ein Programm $p' \in P^+$ bezeichnen wir die Menge aller in diesem Programm auftretenden Funktionen und Konstruktoren ebenfalls mit $F_{p'}$ bzw. $C_{p'}$. Außerdem bezeichnen wir für ein $f \in F_p$ die Menge aller Konstruktoren, für welche f definiert ist, mit C_f^i sowie die Menge aller in den rechten Seiten der Regeln von f auftretenden Konstruktoren mit C_f^o. Wegen $P \subset P^+$ läßt sich diese Notation auch für Programme aus P verwenden. Sei M ein macro tree transducer in einem Programm $p \in P$ und $f \in F_M$. Dann ist $C_f^i = C_M^i$ und $C_f^o = C_M^o$.

2.3.2 Reduktionsvorgänge

In diesem Abschnitt befassen wir uns mit den Berechnungen, die mit Hilfe der im vorigen Abschnitt eingeführten Programme durchgeführt werden können. Zunächst legen wir die Semantik der Sprache P^+ fest, also die Berechnungsvorschrift, nach welcher ein Ausdruck mit Hilfe der Regeln eines Programms ausgewertet wird. Wir entscheiden uns für die Auswertungsstrategie „call-by-need“. Diese definieren wir als Erweiterung der „call-by-name“-Strategie, welche die am weitesten außen vorkommenden Funktionsaufrufe auszuwerten versucht. Damit ist ein Funktionsaufruf gemeint, der nicht mehr in einem Parameter eines anderen Aufrufs steht, also nur noch von Kontruktoren umgeben ist. Ein solcher Funktionsaufruf kann reduziert werden, wenn dessen Rekursionsargument schon so weit ausgewertet ist, daß eine Regel angewendet werden kann. Dies besagt Punkt 1. in Definition 2.7. Wenn es mehrere äußerste Funktionsaufrufe in einem Ausdruck gibt, dann werden diese von links nach rechts ausgewertet, was in Punkt 2. formalisiert wird. Wenn im Rekursionsargument des äußersten Funktionsaufrufs ein weiterer Funktionsaufruf steht, dann muß dieser innere Funktionsaufruf zunächst so weit ausgewertet werden, bis dadurch ein Konstruktor entsteht, so daß die Auswahl einer Regel für den äußeren Aufruf möglich ist. Diese Vorgehensweite formalisiert Punkt 3. der folgenden Definition.

Definition 2.7 Sei $p \in P^+$. Die *von p induzierte call-by-name (oder leftmost-outermost) Ableitungsrelation*, bezeichnet mit $\Rightarrow_p$, ist die kleinste auf $T_{F_p \cup C_p}$ definierte binäre Relation $\Rightarrow$, so daß

1. Für jede Regel $f\ (c\, x_1 \ldots x_k)\, y_1 \ldots y_n = rhs(f,c)$ aus p mit $f \in F_p^{(1+n)}$, $c^{(k)} \in C_f^i$ und $n, k \in I\!N$ sowie für alle Bäume $t_1, \ldots, t_k \in T_{F_p \cup C_p}$ und $t'_1, \ldots, t'_n \in T_{F_p \cup C_p}$ gilt
$$(f\ (c\, t_1 \ldots t_k)\, t'_1 \ldots t'_n) \Rightarrow rhs(f,c)[x_1/t_1, \ldots, x_k/t_k][y_1/t'_1, \ldots, y_n/t'_n].$$

2. Für alle $k \in I\!N$, $c \in C_p^{(k)}$, $i \in [k]$, $t_1, \ldots, t_{i-1} \in T_{C_p}$, $t_i, \ldots, t_k, t'_i \in T_{F_p \cup C_p}$ ist mit $t_i \Rightarrow t'_i$ auch
$$(c\, t_1 \ldots t_i \ldots t_k) \Rightarrow (c\, t_1 \ldots t'_i \ldots t_k).$$

3. Für alle $n \in I\!N$, $f \in F_p^{(1+n)}$, $n' \in I\!N$, $f' \in F_p^{(1+n')}$, $t_1, \ldots, t_n, t'_0, \ldots, t'_{n'}, t' \in T_{F_p \cup C_p}$ ist mit $(f'\ t'_0\ t'_1 \ldots t'_{n'}) \Rightarrow t'$ auch

$$(f\ (f'\ t'_0\ t'_1 \ldots t'_{n'})\ t_1 \ldots t_n) \Rightarrow (f\ t'\ t_1 \ldots t_n).$$

Sei M ein macro tree transducer in p. Die *von M induzierte call-by-name (oder leftmost-outermost) Ableitungsrelation*, bezeichnet mit $\Rightarrow_M$, entsteht aus $\Rightarrow_p$ durch Beschränkung im Punkt 1. auf die Regelmenge von M. $\square$

Für alle $p \in P$ und $e \in T_{F_p \cup C_p}$ existiert die Normalform $nf(\Rightarrow_p, e)$. Dies geht auf ein Resultat aus [EV85] zurück, wonach die (nichtdeterministische) Ableitungsrelation für macro tree transducer konfluent und terminierend ist.

Die in dieser Form definierte Ableitungsrelation hat den Nachteil, daß sie für nichtlineare Regeln nicht besonders effizient ist. Wenn gemäß Punkt 1. in Definition 2.7 eine mehrfach vorkommende Variable in einer rechten Seite durch einen aufwendig zu berechnenden Ausdruck ersetzt wird, dann muß diese Berechnung auch mehrfach durchgeführt werden. Deshalb verwendet man in der Praxis eine abgewandelte Version dieser Relation, die sogenannte call-by-need-Strategie. Dabei werden die Variablen nicht direkt durch die aktuellen Bäume ersetzt, sondern durch Referenzen darauf, und zwar gleiche Variablen durch die gleiche Referenz. Wenn im weiteren Verlauf der Ableitung der Wert einer Variablen benötigt wird, so wird der referenzierte Ausdruck ausgewertet. Sämtliche spätere Referenzierungen dieses Ausdrucks können das Ergebnis dann direkt verwenden, ohne den Ausdruck noch einmal auswerten zu müssen. Ansonsten verhalten sich beide Relationen gleich, weswegen wir kein neues Relationszeichen für call-by-need einführen. Im folgenden meinen wir mit $\Rightarrow_p$ bzw. $\Rightarrow_M$ jedoch immer die Auswertung mittels call-by-need, sofern wir nicht ausdrücklich call-by-name erwähnen. Den Unterschied zwischen den beiden beschriebenen Strategien bei der Anzahl der benötigten Reduktionsschritte wollen wir noch an einem Beispiel veranschaulichen.

Beispiel 2.8 Gegeben sei folgendes Programm, welches aus zwei top-down tree transducern M_1 und M_2 besteht, wobei $F_{M_1} = \{f^{(1)}\}$ und $F_{M_2} = \{g^{(1)}\}$.

$$\begin{array}{lcl} f\ (\gamma\ x_1) & = & \gamma\ (f\ x_1) \\ f\ \alpha & = & \alpha \\[1ex] g\ (\gamma\ x_1) & = & \sigma\ (g\ x_1)\ (g\ x_1) \\ g\ \alpha & = & \alpha \end{array}$$

M_1 berechnet die Identität eines monadischen Baumes. M_2 erzeugt aus einem monadischen Baum, der aus n γ-Symbolen besteht, einen aus σ-Symbolen aufgebauten Binärbaum mit n Verzweigungsniveaus. Nun reduzieren wir den Ausdruck $g\ (f\ (\gamma^n\ \alpha))$ und zeigen in Abbildung 2.1 den Beginn des Reduktionsvorgangs mittels call-by-name. Dabei ist zu beachten, daß aus Gründen der Anschaulichkeit beim Übergang vom dritten zum vierten Baum von der tatsächlichen Reduktionsreihenfolge abgewichen wurde, da der linke Teilbaum eigentlich erst vollständig reduziert werden müßte, bevor der rechte Aufruf von g ausgewertet werden könnte. Diese Änderung ist jedoch zulässig, da das Ergebnis davon unbeeinflußt bleibt und die beiden vorgezogenen Reduktionsschritte später exakt auf diese Weise durchgeführt werden.

Wir erkennen an dieser Abbildung, daß sich mit jedem neuen Verzweigungsniveau die Zahl der Aufrufe der Funktion f verdoppelt. Zum Schluß sind z.B. 2^n Aufrufe der Form $f\ \alpha$

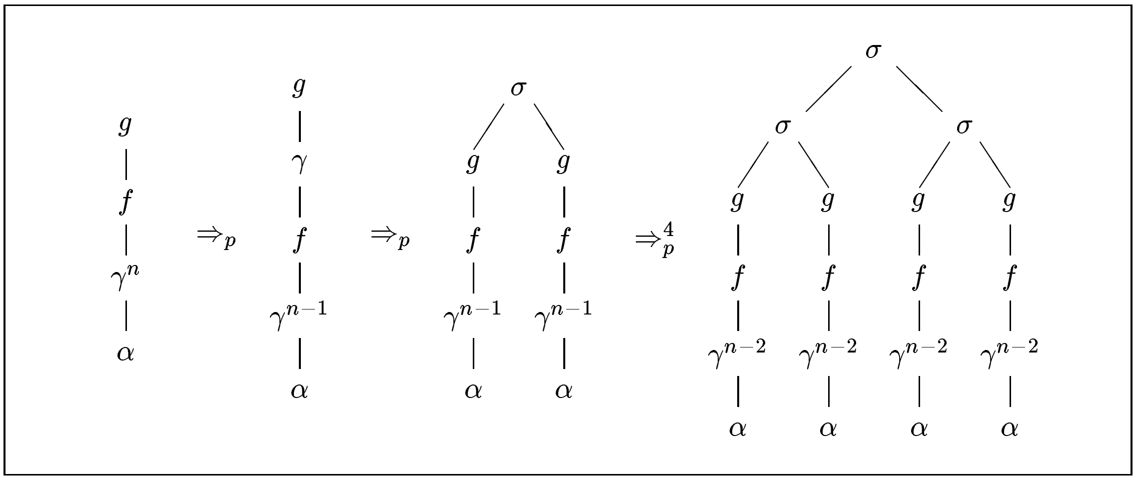

Abbildung 2.1: Reduktionsvorgang mittels call-by-name

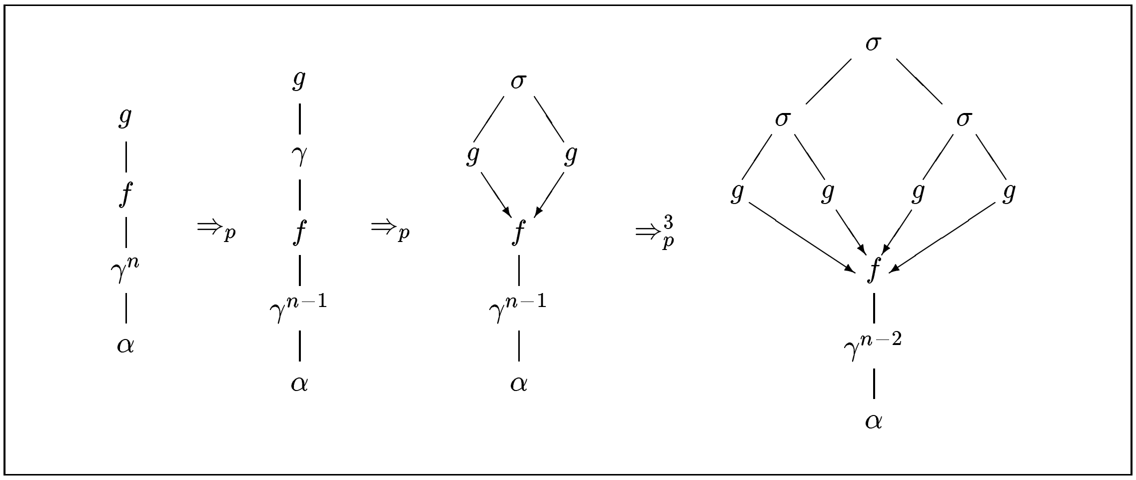

Abbildung 2.2: Reduktionsvorgang mittels call-by-need

einzeln zu reduzieren. Nun verdeutlichen wir in Abbildung 2.2 den Beginn des Reduktionsvorgangs mit Hilfe der call-by-need-Strategie. Dabei sind die Referenzen durch Pfeile veranschaulicht.

Dieser Abbildung ist zu entnehmen, daß unabhängig vom Verzweigungsniveau jeweils genau ein Aufruf der Funktion f zu reduzieren ist, da das Ergebnis dieses Reduktionsvorgangs von den übrigen Aufrufen der Funktion g gleich mitverwendet werden kann. Die call-by-need-Strategie spart also einen erheblichen Teil der Reduktionsschritte ein. □

Mit Hilfe dieser Ableitungsrelation können für ein Programm $p \in P$ nun Ausdrücke, die aus Funktionssymbolen und Konstruktoren bestehen, in nur noch aus Konstruktoren bestehende Ausdrücke umgeformt werden. Da wir die Transformationen, die wir im nächsten Abschnitt einführen werden, jeweils für eine ganze Klasse von Ausdrücken definieren wollen, verwenden wir zusätzlich Ausdrucksvariablen. Der Aufbau solcher Eingabeausdrücke für Programme wird in der folgenden Definition festgelegt. Dabei wird sichergestellt, daß die Argumente von Funktionen jeweils Bäume über dem richtigen Rangalphabet sind.

Definition 2.9 Sei $p \in P^+$. Die Menge aller *Berechnungsausdrücke für p*, bezeichnet mit E_p, ist die kleinste Teilmenge E von $T_{F_p \cup C_p}(Z)$, die folgenden Bedingungen genügt:

- $Z \subseteq E$
- $T_{C_p} \subseteq E$
- Mit $f \in F_p$ und $e_0 \dots e_n \in E$ ist auch $(f\ e_0\ e_1 \dots e_n) \in E$, falls
  - $e_0 \in Z$ oder $e_0 \in T_{C_f^i}$ oder $e_0 = (f'\ e'_0\ e'_1 \dots e'_{n'})$ mit $f' \in F_p$, $e'_0 \dots e'_n \in E$ und $C_f^i = C_{f'}^o$
  - Für alle $i \in [n]$ ist $e_i \in Z$ oder $e_i \in T_{C_f^o}$ oder $e_i = (f'\ e'_0\ e'_1 \dots e'_{n'})$ mit $f' \in F_p$, $e'_0 \dots e'_n \in E$ und $C_f^o = C_{f'}^o$ □

Ein Ausdruck mit Ausdrucksvariablen beschreibt die Menge aller Ausdrücke, die durch Belegungen der Ausdrucksvariablen mit konkreten Konstruktortermen erzeugt werden können. Bei dieser Ersetzung sind nur bestimmte Bäume zulässig, damit die Bedingungen für die eben definierten Ausdrücke nicht verletzt werden. Deswegen führen wir den Begriff der passenden Belegung ein.

Definition 2.10 Sei $p \in P^+$ und $e \in E_p$. Die Substitution $\sigma = [v/t_v \mid v \in var(e), t_v \in T_{C_p}]$ heißt *passend*, wenn $e\sigma$ ein Berechnungsausdruck ist. □

Die in dieser Arbeit untersuchten Transformationstechniken nehmen Elemente der Menge $\{(p, e) \mid p \in P, e \in E_p\}$ als Eingabe, wobei die Ausdrücke in diesen Paaren folgendes Aussehen haben.

Definition 2.11 Sei $p \in P$ ein Programm, welches zwei macro tree transducer M_1 und M_2 mit $C_{M_1}^o = C_{M_2}^i$ enthält. Ferner sei $f_1 \in F_{M_1}^{(1+n_1)}$ und $f_2 \in F_{M_2}^{(1+n_2)}$ mit $n_1, n_2 \in I\!N$. Einen Ausdruck $e \in E_p$ der Form $e = f_2\ (f_1\ z_0\ z_1 \dots z_{n_1})\ z_{n_1+1} \dots z_{n_1+n_2}$ nennen wir *Kompositionsausdruck bezüglich* M_1 *und* M_2.

Seien K_1 und K_2 jeweils die Klasse MAC oder Teilklassen davon. Die *Kompositionsklasse* $(K_1; K_2)$ ist die Menge aller Paare (p, e), für die $p \in P$ ein Programm ist, in dem zwei tree transducer $M_1 \in K_1$ und $M_2 \in K_2$ enthalten sind, und für die e ein Kompositionsausdruck bezüglich M_1 und M_2 ist. □

Bei der Transformation eines Ausdrucks und des zugehörigen Programms sollen nach Belegung der Ausdrucksvariablen die bei der Reduktion entstehenden Ergebnisse nicht verändert werden, die Semantik soll also erhalten bleiben. In diesem Fall sprechen wir von Äquivalenz, die wir folgendermaßen definieren.

Definition 2.12 Sei $p_1, p_2 \in P^+$, $e_1 \in E_{p_1}$ und $e_2 \in E_{p_2}$. Die Paare (p_1, e_1) und (p_2, e_2) heißen *äquivalent*, bezeichnet mit $(p_1, e_1) \equiv (p_2, e_2)$, wenn $C_{p_1} = C_{p_2}$ und für alle passenden Substitutionen σ der Ausdrucksvariablen aus e_1 und e_2 durch Bäume aus $T_{C_{p_1}}$ gilt:

$$nf(\Rightarrow_{p_1}, e_1\sigma) = nf(\Rightarrow_{p_2}, e_2\sigma)$$ □

Zum Vergleich der Effizienz äquivalenter Paare benutzen wir folgende Notationen.

Definition 2.13 Sei $p_1, p_2 \in P^+$, $e_1 \in E_{p_1}$, $e_2 \in E_{p_2}$ und $(p_1, e_1) \equiv (p_2, e_2)$. Wir nennen (p_1, e_1) *genauso effizient wie* (p_1, e_1), bezeichnet durch $\mathcal{E}(p_1, e_1) = \mathcal{E}(p_2, e_2)$, bzw. *genauso effizient oder effizienter als* (p_2, e_2), bezeichnet durch $\mathcal{E}(p_1, e_1) \geq \mathcal{E}(p_2, e_2)$, bzw. *effizienter als* (p_2, e_2), bezeichnet durch $\mathcal{E}(p_1, e_1) > \mathcal{E}(p_2, e_2)$, wenn für alle passenden Substitutionen σ der Ausdrucksvariablen aus e_1 und e_2 durch Bäume aus $T_{C_{p_1}}$ und für die daraus resultierenden Reduktionen $e_1\sigma \Rightarrow_{p_1}^{r_{1,\sigma}} nf(e_1\sigma)$ und $e_2\sigma \Rightarrow_{p_2}^{r_{2,\sigma}} nf(e_2\sigma)$ mit $r_{1,\sigma}, r_{2,\sigma} \in I\!N$ gilt:

$$r_{1,\sigma} = r_{2,\sigma} \text{ bzw. } r_{1,\sigma} \leq r_{2,\sigma} \text{ bzw. } r_{1,\sigma} < r_{2,\sigma}.$$ □

Für den Effizienzvergleich muß die Anzahl von Reduktionsschritten berechnet werden. Für die erforderlichen näheren Untersuchungen von Reduktionsvorgängen benötigen wir einige Begriffe und abkürzende Bezeichnungen. Im folgenden sei $p \in P^+$ ein Programm. Außerdem verwenden wir die Funktionen $f_0^{(1+n_0)}, f^{(1+n)}, f'^{(1+n')} \in F_p$ mit $n_0, n, n' \in I\!N$, den Eingabebaum $t \in T_{C^i_{f_0}}$ nebst dessen Teilbäumen $t_1, \ldots, t_k$, den Berechnungsausdruck $(f_0\ t\ t_1'^0 \ldots t_{n_0}'^0)$ mit $t_1'^0, \ldots, t_{n_0}'^0 \in T_{C_p \cup F_p}$, den Ausgabebaum $t' = nf(\Rightarrow_p, (f_0\ t\ t_1'^0 \ldots t_{n_0}'^0))$ und das Vorkommen $o \in occ(t)$ mit $label(o,t) = c^{(k)}$, $c \in C^i_{f_0}$ und $k \in I\!N$.

Definition 2.14 Seien $t_a, t_b \in T_{C_p \cup F_p}$ zwei Bäume, die während der Berechnung von t' auftreten und durch einen Reduktionsschritt ineinander überführt werden, d.h. es gelte $(f_0\ t\ t_1'^0 \ldots t_{n_0}'^0) \Rightarrow_p^* t_a \Rightarrow_p t_b \Rightarrow_p^* t'$. Dabei sei $subtree(o,t) = t^o = (c\, t_1 \ldots t_k)$, $o' \in occ(t')$, $subtree(o', t_a) = (f\ t^o\ t_1' \ldots t_n')$ und $subtree(o', t_b) = rhs(f,c)[x_i/t_i][y_j/t_j']$, wobei $t_1', \ldots, t_n' \in T_{C^o_{f_0} \cup F_p \cup C^i_{f_0}}$. Dann nennen wir den Reduktionsschritt $t_a \Rightarrow_p t_b$ einen *Besuch des Vorkommens o durch f* und das Vorkommen o' das *durch Übersetzung des Vorkommens o mit f entstehende Vorkommen* oder ein *Bild von o bezüglich f*. □

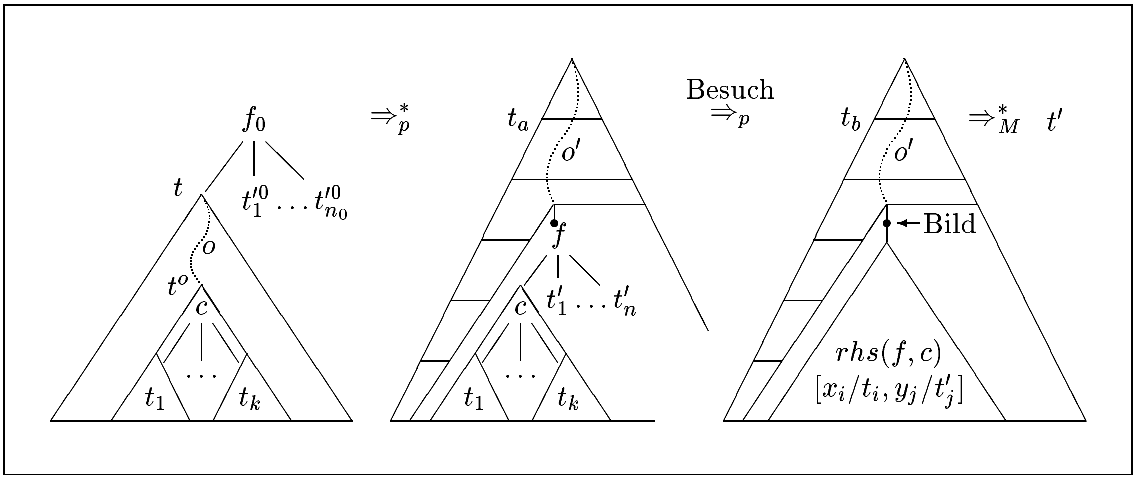

Abbildung 2.3: Besuch eines Vorkommens o durch f und das entstehende Bild o'

Die in Definition 2.14 verwendeten Bäume und eingeführten Begriffe sind in Abbildung 2.3 veranschaulicht. Die schraffierten Teile von t_a und t_b, welche sich links und oberhalb des Funktionsaufrufs befinden, bestehen aufgrund der Reduktionsreihenfolge leftmost-outermost bereits vollständig aus Ausgabesymbolen und erscheinen deshalb in dieser Form auch im Ausgabebaum t', weswegen $o' \in occ(t')$ gilt.

Da bei nichtlinearen tree transducern ein Vorkommen auch mehrfach besucht werden kann, benötigen wir die folgende Notation für die Anzahl der Besuche.

Definition 2.15 Wir bezeichnen mit $m_f^{f_0,t}(o)$ die *Anzahl der Besuche des Vorkommens o durch die Funktion f*, wenn die Übersetzung mit f_0 an der Wurzel von t begonnen wird. Weiterhin bezeichnen wir mit $m^{f_0,t}(o)$ die *Gesamtanzahl der Besuche des Vorkommens o*, wenn die Übersetzung mit einem Besuch der Wurzel von t durch f_0 gestartet wird, d.h. es gilt

$$m^{f_0,t}(o) = \sum_{f \in F_p} m_f^{f_0,t}(o)$$ □

Wenn ein Vorkommen mehrfach besucht wird, entstehen durch die Übersetzung unter Umständen auch mehrere verschiedene Vorkommen im Ausgabebaum. In der folgenden Definition führen wir eine Notation ein, um jedem Vorkommen eines Eingabebaums die zugehörigen Bildvorkommen im Ausgabebaum zuordnen zu können.

Definition 2.16 Für die *Menge der Bilder des Vorkommens o bezüglich f* schreiben wir $bild_f^{f_0,t}(o)$. Die *Menge der Bilder des Vorkommens o*, welche alle Vorkommen aus t' enthält, die durch Übersetzung des Vorkommens o entstehen, bezeichnen wir mit $bild^{f_0,t}(o)$ und definieren sie als

$$bild^{f_0,t}(o) = \bigcup_{f \in F_p} bild_f^{f_0,t}(o)$$ □

Zwischen einem Vorkommen im Ausgabebaum und den Bildmengen der Vorkommen aus dem Eingabebaum sind folgende Beziehungen möglich:

1. Das Vorkommen ist in keiner Bildmenge enthalten. Das ist der Fall, wenn bei einem Besuch im Eingabebaum mehr als ein Ausgabesymbol im Ausgabebaum erzeugt wird. Unter diesen Symbolen existiert genau eines (nämlich das oberste), dessen Pfad am kürzesten ist, und nur dieser Pfad ist ein Bildvorkommen.

2. Das Vorkommen ist in genau einer Bildmenge enthalten.

3. Das Vorkommen ist in mehr als einer Bildmenge enthalten.

   (a) Das Vorkommen ist in mehreren Bildmengen des gleichen Eingabevorkommens bezüglich verschiedener Funktionen enthalten. Dies geschieht, wenn in der rechten Seite einer Regel ein verschachtelter Funktionsaufruf mit gleichen Rekursionsargumenten steht, wobei während der Reduktion des äußeren Aufrufs solange kein Ausgabesymbol erzeugt wird, bis der innere Funktionsaufruf erreicht wird. Dann sind die Pfade zu den Bildern des äußeren und inneren Aufrufs gleich.

   (b) Das Vorkommen ist in mehreren Bildmengen verschiedener Eingabevorkommen enthalten. Dies geschieht, wenn in der rechten Seite einer Regel ein Funktionsaufruf steht, ohne daß sich darüber noch Ausgabesymbole befinden. Dann ist das Bild des Besuchs, bei dem diese Regel angewendet wurde, gleich dem bei der Reduktion des Funktionsaufrufs entstehenden Bild. Eine weitere Möglichkeit für diesen Fall besteht darin, daß die rechte Seite nur eine einzelne Kontextvariable enthält. Wenn in dem dafür eingesetzten Baum ein solcher Funktionsaufruf an der Wurzel steht, gleicht das Bild dieses Aufrufs ebenfalls dem beim vorangegangenen Besuch entstehenden Bild.

Im folgenden Lemma geben wir Bedingungen an, für die Fall 3.(a) nicht auftritt.

Lemma 2.17 Sei $M \in TOP$ oder $M \in lin\text{-}MAC$ oder $M \in prod\text{-}MAC$ und $f_0, f, f' \in F_M$. Dann gilt die Implikation

$$f \neq f' \Rightarrow bild_f^{f_0,t}(o) \cap bild_{f'}^{f_0,t}(o) = \emptyset$$

Beweis: Wenn $M \in TOP$, so können keine verschachtelten Funktionsaufrufe auftreten. Wenn $M \in lin\text{-}MAC$, dann haben verschachtelte Funktionsaufrufe verschiedene Rekursionsargumente. Wenn $M \in prod\text{-}MAC$, dann erzeugt der äußere Funktionsaufruf stets mindestens einen Konstruktor im Ausgabebaum, bevor der innere Aufruf erreicht wird. Dieser muß durch eine Regel aktiviert werden, die eine Kontextvariable außerhalb der Kontextparameter eines Funktionsaufrufs besitzt. Damit diese Regel producing wird, muß über der Kontextvariablen mindestens ein Ausgabesymbol stehen, sonst bestünde die Regel nur aus dieser Variablen. Dieses Symbol wird aber im Ausgabebaum eingesetzt, bevor der innere Funktionsaufruf ausgewertet wird. □

Das nächste Lemma gibt eine Bedingung an, für die Fall 3.(b) nicht auftritt.

Lemma 2.18 Sei $M \in prod\text{-}TOP$, $f_0 \in F_M$ und $o' \in occ(t)$. Dann gilt die Implikation

$$o \neq o' \Rightarrow bild^{f_0,t}(o) \cap bild^{f_0,t}(o') = \emptyset$$

Beweis: Wegen $M \in prod\text{-}TOP$ enthalten alle rechten Regelseiten Kontruktoren, und da top-down tree transducer keine Kontextparameter besitzen, müssen diese Konstruktoren außerhalb von Funktionsaufrufen stehen. □

Zum Schluß führen wir noch eine Notation für die Anzahl der Reduktionsschritte ein, die insgesamt zur Reduktion eines Ausdrucks gebraucht werden.

Definition 2.19 Sei $e \in T_{F_p \cup C_p}$ ein Berechnungsausdruck ohne Ausdrucksvariablen. Die *Anzahl der Reduktionsschritte, die zur Reduktion von e benötigt werden*, bezeichnen wir mit r^e, d.h. es gilt $e \Rightarrow_p^{r^e} nf(\Rightarrow_p, e)$.

Falls $e = (f_0\ t\ t_1'^0 \dots t_{n_0}'^0)$ und $t_1'^0, \dots, t_{n_0}'^0 \in T_{C_{f_0}^o}$, kürzen wir $r^{(f_0\ t\ t_1'^0 \dots t_{n_0}'^0)}$ mit $r^{f_0,t}$ ab. □

Die Anzahl der Reduktionsschritte bei der Übersetzung eines Baumes kann mit Hilfe der Anzahl der Besuche berechnet werden.

Lemma 2.20 Seien alle Regeln aus p nondeleting in den Kontextparametern und $e = (f_0\ t\ t_1'^0 \dots t_{n_0}'^0)$. Dann gilt:

$$r^e = \sum_{o \in occ(t)} m^{f_0,t}(o) + \sum_{i=1}^{n_0} r^{t_i'^0}$$

Falls $t_1'^0, \dots, t_{n_0}'^0 \in T_{C_{f_0}^o}$, gilt $r^{f_0,t} = \sum\limits_{o \in occ(t)} m^{f_0,t}(o)$.

Beweis: Jeder Besuch im Eingabebaum entspricht nach Definition 2.14 einem Reduktionsschritt. Wenn alle Regeln des Programms nondeleting in den Kontextparametern sind, geht keiner der Bäume $t_1'^0, \dots, t_{n_0}'^0$ verloren. Wegen der call-by-need-Auswertungsstrategie werden sie genau einmal ausgewertet. Falls diese wie im zweiten Teil des Lemmas nur aus Konstruktoren bestehen, beeinflussen sie die Anzahl der Reduktionsschritte nicht. □

Einige der eben eingeführten Notationen und Begriffe wollen wir noch an einem Beispiel veranschaulichen.

Beispiel 2.21 Wir benutzen den macro tree transducer M_{bsp} aus Beispiel 2.3. Dabei betrachten wir den Eingabebaum $t_e = \delta\gamma\alpha_1$ und belegen den Kontextparameter mit dem Baum α. Dann erhalten wir $nf(\Rightarrow_{M_{bsp}}, (f\ t_e\ \alpha)) = \sigma\ (\gamma\alpha)\ (\gamma\alpha)$. Für die Zahl der Besuch gilt $m_f^{f,t_e}(\varepsilon) = 1$, $m_f^{f,t_e}(1) = 2$ und $m_f^{f,t_e}(1.1) = 4$. Da M_{bsp} nur eine Funktion besitzt, gilt nach Definition 2.15 für alle Vorkommen $o_e \in occ(t_e)$ die Beziehung $m_f^{f,t_e}(o_e) = m^{f,t_e}(o_e)$. Nach Definition 2.19 und Lemma 2.20 erhalten wir die Gesamtzahl der Reduktionsschritte als Summe der Besuche aller Vorkommen, damit gilt $r^{f,t_e} = 7$. Für die Bildmengen gilt $bild_f^{f,t_e}(\varepsilon) = \{\varepsilon\}$, $bild_f^{f,t_e}(1) = \{1,2\}$ und $bild_f^{f,t_e}(1.1) = \{1.1, 2.1\}$. Dabei wird für die Eingabevorkommen ε und 1 bei jedem Besuch im Ausgabebaum genau ein Bildvorkommen erzeugt, während jeweils zwei Besuche des Vorkommens 1.1 das selbe Bild besitzen. $\square$

2.4 Komposition

In [Rou70] wurde gezeigt, daß die Komposition zweier top-down tree transducer durch einen einzigen top-down tree transducer simuliert werden kann. In [EV85] wurde dieses Resultat auf die Komposition eines macro-tree transducers mit einem top-down tree transducer bzw. umgekehrt erweitert, wobei entsprechende Techniken angegeben wurden, welche den neuen macro tree transducer konstruieren. Außerdem wurde dort bewiesen, daß es eine solche Konstruktion für die Komposition zweier macro tree transducer nicht geben kann, weil die Klasse der durch alle macro tree transducer berechneten Übersetzungen von einem Baum in einen anderen unter der Komposition nicht abgeschlossen ist.

Die Kernidee für die hier angegebenen Techniken zur Komposition zweier tree transducer M_1 und M_2 liefert die Beobachtung, daß Zwischenergebnisse durch die Konstruktoren der rechten Regelseiten von M_1 erzeugt werden. Anstelle dieses Zwischenergebnis nun mit M_2 zu übersetzen, übersetzen wir die rechten Seiten mit M_2 und erhalten so die Regeln für einen neuen tree transducer M, der das Endergebnis in einem Durchgang erzeugt und somit ohne Zwischenergebnisse auskommt.

Weil die rechten Seiten von M_1 im allgemeinen auch noch Funktionen enthalten, also nicht vollständig von M_2 übersetzt werden können, benutzen wir als Funktionsmenge von M das kartesische Produkt der Funktionsmengen von M_1 und M_2. Ein Paar (f_1, f_2) aus $F_{M_1} \times F_{M_2}$ bedeutet dabei, daß die Funktion f_2 bei der Übersetzung auf f_1 gestoßen ist. Es faßt die Hintereinanderausführung von f_1 und f_2 zu einem Schritt zusammen und wird im folgenden mit $\overline{f_1 f_2}$ abgekürzt.

2.4.1 Die Kompositionsklasse $(TOP; MAC)$

Die Techniken für die Kompositionsklassen $(TOP; MAC)$ und $(MAC; TOP)$ unterscheiden sich in der Behandlung der Kontextparameter. Wir geben zuerst ein Konstruktionsverfahren für die Klasse $(TOP; MAC)$ an. Dabei wird zunächst ein neuer tree transducer definiert, mit dessen Hilfe dann die eigentliche Transformation durchgeführt werden kann.

Transformation 2.22 Sei $(p, e) \in (TOP; MAC)$. Sei $M_1 = (F_{M_1}^{(1)}, C_{M_1}^i, C_{M_1}^o, R_{M_1})$ ein top-down tree transducer und $M_2 = (F_{M_2}, C_{M_2}^i, C_{M_2}^o, R_{M_2})$ ein macro tree transducer aus p mit $C_{M_1}^o = C_{M_2}^i$. Weiterhin sei $e = f_2\ (f_1\ z_0)\ z_1 \ldots z_{n_2}$ mit $f_1 \in F_{M_1}^{(1)}$ und $f_2 \in F_{M_2}^{(1+n_2)}$. Wir konstruieren ein neues Programm $p' \in P$ und einen neuen Berechnungsausdruck $e' \in E_{p'}$, den wir *Ergebnisausdruck* nennen, wie folgt:

1. Sei $mx = max\{k \mid k \in I\!N, C^{i(k)}_{M_1} \neq \emptyset\}$ und $my = max\{n \mid F^{(1+n)}_{M_2} \neq \emptyset\}$. Wir konstruieren aus M_2 einen neuen macro tree transducer $M_2' = (F_{M_2'}, C^i_{M_2'}, C^o_{M_2'}, R_{M_2'})$, welcher in der Lage ist, rechte Seiten von Gleichungen von M_1 zu übersetzen. Dieser ist jedoch nicht Bestandteil des neuen Programms p'.

   - $F_{M_2'} = F_{M_2} \cup \{id^{(1)}\}$
   - $C^i_{M_2'} = C^i_{M_2} \cup F^{(1)}_{M_1} \cup X^{(0)}_{mx}$
   - $C^o_{M_2'} = C^o_{M_2} \cup \{\overline{fg}^{(1+n)} \mid f \in F^{(1)}_{M_1}, g \in F^{(1+n)}_{M_2} \text{ mit } n \in I\!N\} \cup X^{(0)}_{mx} \cup Y^{(0)}_{my}$
   - $R_{M_2'}$ enthält die Regeln aus R_{M_2} und wird folgendermaßen erweitert:
     - Für alle $f \in F^{(1)}_{M_1}$ und $g \in F^{(1+n)}_{M_2}$ mit $n \in I\!N$ enthält $R_{M_2'}$ folgende Gleichung:

       $g\ (f\ x_1)\ y_1 \dots y_n = \overline{fg}\ (id\ x_1)\ y_1 \dots y_n$

       Die Rekursionsvariable x_1 des einstelligen Eingabesymbols f von M_2' enthält das Rekursionsargument der einstelligen Funktion f des top-down tree transducers M_1. Dieses Rekursionsargument besteht wieder aus einer Rekursionsvariablen, die wir aber als nullstelliges Eingabesymbol betrachten und mit Hilfe der Identitätsfunktion übersetzen.
     - Wir fügen für alle $c \in C^{i(k)}_{M_2'}$ mit $k \in I\!N$ Regeln für die Identitätsfunktion id hinzu:

       $id\ (c\ x_1 \dots x_k) = c\ (id\ x_1) \dots (id\ x_k)$

       Entscheidend sind davon die Regeln mit $c = x_i^{(0)}$ zur Übersetzung der Rekursionsvariablen aus X_{mx}, die wir ja als Konstruktoren aus dem Eingabealphabet von M_2' ansehen. Diese sind nicht zu verwechseln mit den Rekursionsvariablen von Regeln aus R_{M_2} für mehr als nullstellige Konstruktoren.

       $id\ x_i = x_i$

2. p' entsteht aus p, indem M_1 und M_2 durch einen neuen macro tree transducer $M = (F_M, C^i_M, C^o_M, R_M)$ ersetzt werden, welcher folgendermaßen entsteht:

   - $F_M = \{\overline{fg}^{(1+n)} \mid f \in F^{(1)}_{M_1}, g \in F^{(1+n)}_{M_2} \text{ mit } n \in I\!N\}$
   - $C^i_M = C^i_{M_1}$
   - $C^o_M = C^o_{M_2}$
   - R_M enthält für jedes $f \in F^{(1)}_{M_1}$, $g \in F^{(1+n)}_{M_2}$ mit $n \in I\!N$ und $c \in C^{i(k)}_{M_1}$ mit $k \in I\!N$ die Regel

     $$\overline{fg}\ (c\ x_1 \dots x_k)\ y_1 \dots y_n = nf(\Rightarrow_{M_2'}, g\ (rhs(f,c))\ y_1 \dots y_n)$$

     Die Kontextparameter werden dabei mit Bäumen initialisiert, die aus nullstelligen Ausgabesymbolen aus der Menge $Y^{(0)}_{my}$ bestehen.

3. $e' = \overline{f_1 f_2}\ z_0\ z_1 \dots z_{n_2}$

Das Transformationsergebnis (p', e') bezeichnen wir mit $Com(p, e)$. □

Lemma 2.23 Für alle $(p, e) \in (TOP; MAC)$ gilt $(p, e) \equiv Com(p, e)$. □

Wir verzichten hier auf den Beweis und weisen darauf hin, daß in [EV85] anstelle der direkten Konstruktion ein anderer Weg beschritten wurde. Dort wurde der macro tree transducer zunächst in einen top-down tree transducer und eine Substitutionsfunktion zerlegt, dann die beiden top-down tree transducer komponiert und der resultierende top-down tree transducer wieder mit der Ersetzungsfunktion komponiert. Das Transformationsresultat ist jedoch das gleiche. Nun führen wir die Komposition an einem Beispiel durch.

Beispiel 2.24 Das Programm $p \in P$ bestehe aus den beiden tree transducern M_{ex} und M_{rev} aus den Beispielen 2.3 und 2.6, und $e = (rev\ (ex\ z_0)\ z_1) \in E_p$ ist ein Kompositionsausdruck für dieses Programm. Es gilt $(p, e) \in (TOP; MAC)$. M'_{rev} besteht dann aus folgenden Gleichungen:

$$\begin{array}{lcllcl}
rev\ (A\ x_1)\ y_1 & = & rev\ x_1\ (A\ y_1) & id\ (A\ x_1) & = & A\ (id\ x_1) \\
rev\ (B\ x_1)\ y_1 & = & rev\ x_1\ (B\ y_1) & id\ (B\ x_1) & = & B\ (id\ x_1) \\
rev\ N\ y_1 & = & y_1 & id\ N & = & N \\
rev\ (ex\ x_1)\ y_1 & = & \overline{exrev}\ (id\ x_1)\ y_1 & id\ (ex\ x_1) & = & ex\ (id\ x_1) \\
 & & & id\ x_1 & = & x_1
\end{array}$$

Das neue Programm p' enthält nur einen macro tree transducer mit folgender Regelmenge:

$$\begin{array}{lcl}
\overline{exrev}\ (A\ x_1)\ y_1 & = & nf(\Rightarrow_{M'_{rev}}, rev\ (rhs(ex, A))\ y_1) \\
 & = & nf(\Rightarrow_{M'_{rev}}, rev\ (B\ (ex\ x_1))\ y_1) \\
 & = & nf(\Rightarrow_{M'_{rev}}, rev\ (ex\ x_1)\ (B\ y_1)) \\
 & = & nf(\Rightarrow_{M'_{rev}}, \overline{exrev}\ (id\ x_1)\ (B\ y_1)) \\
 & = & \overline{exrev}\ x_1\ (B\ y_1) \\
\overline{exrev}\ (B\ x_1)\ y_1 & = & \overline{exrev}\ x_1\ (A\ y_1) \\
\overline{exrev}\ N\ y_1 & = & y_1
\end{array}$$

Der neue Berechnungsausdruck ist $e' = \overline{exrev}\ z_0\ z_1$. Also ist $Com(p, e) = (p', e')$. □

2.4.2 Die Kompositionsklasse $(MAC; TOP)$

Nun behandeln wir die Kompositionsklasse $(MAC; TOP)$. Hierbei sind noch zusätzlich die Kontextparameter zu berücksichtigen, da die Funktionen des top-down tree transducers bei der Übersetzung auf Kontextvariablen stoßen können. Weil es dafür keine Regeln gibt, übersetzen wir diese schon im voraus und führen alle möglichen Kombinationen mit, weil wir vorher nicht wissen, welche Paare bei der Übersetzung tatsächlich auftreten. Die Funktionen des neuen tree transducers müssen also vom Rang her entsprechend vergrößert werden.

Transformation 2.25 Sei $(p, e) \in (MAC; TOP)$. Sei $M_1 = (F_{M_1}, C^i_{M_1}, C^o_{M_1}, R_{M_1})$ ein macro tree transducer und $M_2 = (F^{(1)}_{M_2}, C^i_{M_2}, C^o_{M_2}, R_{M_2})$ ein top-down tree transducer aus p mit $C^o_{M_1} = C^i_{M_2}$. Sei $(g_1, \ldots, g_h)$ mit $h = card(F_{M_2})$ eine beliebige, aber feste Ordnung der Funktionen aus F_{M_2}. Weiterhin sei $e = f_2\ (f_1\ z_0\ z_1 \ldots z_{n_1})$ ein Kompositionsausdruck mit $f_1 \in F^{(1+n_1)}_{M_1}$ und $f_2 \in F^{(1)}_{M_2}$. Wir konstruieren ein neues Programm $p' \in P$ und einen neuen Berechnungsausdruck $e' \in E_{p'}$, den wir *Ergebnisausdruck* nennen, wie folgt:

1. Sei $mx = max\{k \mid C^{i(k)}_{M_1} \neq \emptyset\}$ und $my = max\{n \mid F^{(1+n)}_{M_1} \neq \emptyset\}$. Um rechte Seiten von Gleichungen aus M_1 übersetzen zu können, konstruieren wir zunächst aus M_2 einen neuen top-down tree transducer $M'_2 = (F_{M'_2}, C^i_{M'_2}, C^o_{M'_2}, R_{M'_2})$. Dieser ist jedoch nicht Bestandteil des neuen Programms p'.

   - $F_{M'_2} = F_{M_2} \cup \{id^{(1)}\}$

- $C^i_{M_2'} = C^i_{M_2} \cup F_{M_1} \cup X^{(0)}_{mx} \cup Y^{(0)}_{my}$
- $C^o_{M_2'} = C^o_{M_2} \cup \{\overline{fg}^{(1+h\cdot n)} \mid n \in \mathbb{N}, f \in F^{(1+n)}_{M_1}, g \in F^{(1)}_{M_2}\} \cup X^{(0)}_{mx} \cup Y^{(0)}_{h\cdot my}$
- $R_{M_2'}$ enthält die Regeln aus R_{M_2} und wird folgendermaßen erweitert:
  - Für alle $f \in F^{(1+n)}_{M_1}$ und $g \in F^{(1)}_{M_2}$ enthält $R_{M_2'}$ folgende Gleichung:
    $g\ (f\ \underline{x_1}\ x_{1+1} \dots x_{1+n})$
    $= \overline{fg}\ (id\ x_1)\ (g_1\ x_{1+1}) \dots (g_h\ x_{1+1}) \dots (g_1\ x_{1+n}) \dots (g_h\ x_{1+n})$
  - Wir fügen für alle $c \in C^{i\,(k)}_{M_2'}$ mit $k \in \mathbb{N}$ Regeln für die Identitätsfunktion id hinzu:
    $id\ (c\ x_1 \dots x_k) = c\ (id\ x_1) \dots (id\ x_k)$
    Entscheidend sind davon die Regeln mit $c = x_i$ zur Übersetzung der Rekursionsvariablen aus X_{mx}, die wir ja als nullstellige Konstruktoren ansehen:
    $id\ x_i = x_i$
  - Für alle $g_m \in F_{M_2}$ mit $m \in [h]$ fügen wir für alle $j \in [my]$ zur Übersetzung der Kontextvariablen aus Y_{my}, die wir ebenfalls als nullstellige Konstruktoren ansehen, folgende Regel hinzu:
    $g_m\ y_j = y_{h\cdot(j-1)+m}$

2. p' entsteht aus p, indem M_2 beibehalten wird und M_1 durch einen neuen macro tree transducer $M = (F_M, C^i_M, C^o_M, R_M)$ ersetzt wird, welcher folgendermaßen entsteht:
   - $F_M = \{\overline{fg}^{(1+h\cdot n)} \mid n \in \mathbb{N}, f \in F^{(1+n)}_{M_1}, g \in F^{(1)}_{M_2}\}$
   - $C^i_M = C^i_{M_1}$
   - $C^o_M = C^o_{M_2}$
   - R_M enthält für jedes $f \in F^{(1+n)}_{M_1}$ mit $n \in \mathbb{N}$, $g \in F^{(1)}_{M_2}$ und $c \in C^{i\,(k)}_{M_1}$ die Regel
   $$\overline{fg}\ (c\ x_1 \dots x_k)\ y_1 \dots y_{h\cdot n} = nf(\Rightarrow_{M_2'}, g\ (rhs(f,c)))$$
3. $e' = \overline{f_1 f_2}\ z_0\ (g_1\ z_1) \dots\ (g_h\ z_1) \dots (g_1\ z_{n_1}) \dots (g_h\ z_{n_1})$

Das Transformationsergebnis (p', e') bezeichnen wir ebenfalls mit $Com(p,e)$. □

Man beachte, daß für die Kompositionsklasse $(TOP; TOP)$ diese Transformation und Transformation 2.22 das gleiche Ergebnis liefern, weil top-down tree transducer eine Teilmenge der macro tree transducer sind.

Lemma 2.26 Für alle $(p,e) \in (MAC; TOP)$ gilt $(p,e) \equiv Com(p,e)$. □

Der Beweis befindet sich in [EV85].

Beispiel 2.27 Das Programm $p \in P$ bestehe aus den beiden tree transducern M_{ex} und M_{rev} aus den Beispielen 2.3 und 2.6, und $e = (ex\ (rev\ z_0\ z_1)) \in E_p$ ist ein Kompositionsausdruck für dieses Programm. Es gilt $(p,e) \in (MAC; TOP)$. Der top-down tree transducer M'_{ex} besteht dann aus folgenden Gleichungen:

$$\begin{array}{lcllcl}
ex\ (A\ x_1) & = & B\ (ex\ x_1) & id\ (A\ x_1) & = & A\ (id\ x_1) \\
ex\ (B\ x_1) & = & A\ (ex\ x_1) & id\ (B\ x_1) & = & B\ (id\ x_1) \\
ex\ N & = & N & id\ N & = & N \\
ex\ (rev\ x_1\ x_2) & = & \overline{revex}\ (id\ x_1)\ (ex\ x_2) & id\ (rev\ x_1\ x_2) & = & rev\ (id\ x_1)\ (id\ x_2) \\
ex\ y_1 & = & y_1 & id\ x_1 & = & x_1 \\
 & & & id\ y_1 & = & y_1
\end{array}$$

Das neue Programm p' enthält nur einen macro tree transducer mit folgender Regelmenge:

$$\begin{array}{lcl}
\overline{revex}\ (A\ x_1)\ y_1 & = & nf(\Rightarrow_{M'_{ex}}, ex\ (rhs(rev, A))) \\
 & = & nf(\Rightarrow_{M'_{ex}}, ex\ (rev\ x_1\ (A\ y_1))) \\
 & = & nf(\Rightarrow_{M'_{ex}}, \overline{revex}\ (id\ x_1)\ (ex\ (A\ y_1))) \\
 & = & nf(\Rightarrow_{M'_{ex}}, \overline{revex}\ x_1\ (ex\ (A\ y_1))) \\
 & = & \overline{revex}\ x_1\ (B\ y_1) \\
\overline{revex}\ (B\ x_1)\ y_1 & = & \overline{revex}\ x_1\ (A\ y_1) \\
\overline{revex}\ N\ y_1 & = & y_1
\end{array}$$

Der neue Berechnungsausdruck ist $e' = \overline{revex}\ z_0\ (ex\ z_1)$. Also ist $Com(p, e) = (p', e')$. □

2.5 Deforestation

Dieses Verfahren zur Optimierung funktionaler Programme stammt von Wadler ([Wad90]) und ist eine algorithmische Instanz der in [BD77] beschriebenen fold/unfold-Technik. Die Elimination von Zwischenergebnissen wird erreicht, indem der zu transformierende Ausdruck einer call-by-name-Reduktion unterzogen wird. Dabei werden Funktionsaufrufe mit einem Konstruktor im Rekursionsargument durch die rechte Seite der passenden Regel ersetzt (unfold-Schritt). Da die Ausdrücke Variablen enthalten können, ist auch das Auftreten von Funktionsaufrufen mit Variablen im Rekursionsargument, deren Werte zur Transformationszeit nicht bekannt sind, möglich. Dann muß eine Fallunterscheidung durchgeführt werden, bei der alle möglichen Konstruktoren für diese Variable eingesetzt werden. Dies geschieht durch Definition neuer Funktionen, deren definierende Gleichungen die Fallunterscheidung repräsentieren. Im Falle des erneuten Auftretens eines solchen Ausdrucks wird dieser durch einen Aufruf der bereits definierten neuen Funktion ersetzt (fold-Schritt).

Für Programme in *treeless form* (s. [Wad90]) ist die Termination des Verfahrens gewährleistet. Da hierbei gefordert wird, daß alle Argumente einer Funktion Variablen sind, sind die Programme aus P im allgemeinen nicht treeless, weil bei macro tree transducern laut Definition 2.1 verschachtelte Ausdrücke in den Kontextparametern eines Funktionsaufrufs vorkommen können. Diese erzeugen jedoch keine Zwischenergebnisse, sondern sind Teil des Endergebnisses. In [HJ92] ist eine Erweiterung des Deforestation-Verfahrens angegeben, die solche Fälle behandeln kann und auf der *blazed*-Deforestation (s. [Wad90]) basiert. Dabei werden Funktionsargumente, die keine Variablen sind, aber keine Zwischenergebnisse erzeugen, mit Hilfe von `let` extrahiert.

Nun geben wir in Transformation 2.28 den Deforestation-Algorithmus an. Wir verwenden die Notation aus [Küh99], welche eine Anpassung der Notation von [SGJ94] an die Syntax unserer Sprache P unter Berücksichtigung von [HJ92] darstellt. Wir benutzen aus Gründen der Übersichtlichkeit eine implizite `let`-Extraktion, der Ausdruck $(f x_i\ t_1 \ldots t_n)$ wird also wie $(\texttt{let}\ v_1 = t_1, \ldots, v_n = t_n\ \texttt{in}\ (f x_i\ v_1 \ldots v_n))$ behandelt.

Transformation 2.28 Sei $(p, e) \in (MAC; MAC)$. Wir definieren eine Transformationsfunktion $\mathcal{D}$, welche den Kompositionsausdruck e explizit und das Programm p implizit als Eingabe nimmt. Die Menge der Ausdrücke, die nach dem Start des Verfahrens mit einem Kompositionsausdruck entstehen können, bezeichnen wir mit T. Die fünf Regeln stellen eine vollständige und nicht überlappende Fallunterscheidung für solche Ausdrücke dar. Insbesondere können Ausdrücke der Form $(f_3\ (f_2\ (f_1 \ldots) \ldots) \ldots)$ und $(f_2\ (f_1\ (c \ldots) \ldots) \ldots)$ in T nicht auftreten. Die Notation $\widetilde{f}$ bzw. $\widetilde{f_1 f_2}$ in den Regeln $(D4)$ und $(D5)$ bezieht sich auf eine Funktion f bzw. $f_1 f_2$, die entweder so konstruiert wird, wie hinter dem Wort

„mit“ angegeben (dieser Teil der Regel erscheint nicht im Transformationsergebnis), oder die bereits konstruiert wurde. Auf diese Weise liefert $\mathcal{D}[\![e]\!]$ einen neuen Ausdruck e' als Ergebnis sowie ein neues Programm $p' \in P^+$, welches alle Regeln enthält, die in $(D4)$ und $(D5)$ erzeugt wurden.

Die verwendeten Symbole v, c, f und t, ggf. mit Indizes und Apostrophen versehen, stehen für Elemente aus den Mengen V, C_p, F_p bzw. T, wobei in den Regeln $(D4)$ und $(D5)$ die Menge C_p auf die Menge C_f bzw. C_{f_1} beschränkt sei, welche jeweils q Elemente mit $q > 0$ enthält.

$$
\begin{array}{lrcl}
(D1) & \mathcal{D}[\![v]\!] & = & v \\
(D2) & \mathcal{D}[\![c\, t_1 \dots t_k]\!] & = & c\, \mathcal{D}[\![t_1]\!] \dots \mathcal{D}[\![t_k]\!] \\
(D3) & \mathcal{D}[\![f\ (c\, t_1 \dots t_k)\ t'_1 \dots t'_n]\!] & = & \mathcal{D}[\![rhs(f,c)[x_i/t_i][y_j/t'_j]]\!] \\
(D4) & \mathcal{D}[\![f\ v\ t'_1 \dots t'_n]\!] & = & \widetilde{f}\ v\ \mathcal{D}[\![t'_1]\!] \dots \mathcal{D}[\![t'_n]\!] \\
 & \text{mit}\quad \widetilde{f}\ (c_1\ x_1 \dots x_{k_1})\ y_1 \dots y_n & = & \mathcal{D}[\![rhs(f,c_1)]\!] \\
 & & \vdots & \\
 & \widetilde{f}\ (c_q\ x_1 \dots x_{k_q})\ y_1 \dots y_n & = & \mathcal{D}[\![rhs(f,c_q)]\!] \\
(D5) & \mathcal{D}[\![f_2\ (f_1\ v\ t'_1 \dots t'_{n_1})\ t''_1 \dots t''_{n_2})]\!] & = & \widetilde{f_1 f_2}\ v\ \mathcal{D}[\![t'_1]\!] \dots \mathcal{D}[\![t'_{n_1}]\!]\ \mathcal{D}[\![t''_1]\!] \dots \mathcal{D}[\![t''_{n_2}]\!] \\
 & \text{mit}\quad \widetilde{f_1 f_2}\ (c_1\ x_1 \dots x_{k_1})\ y_1 \dots y_{n_1+n_2} & = & \mathcal{D}[\![f_2\ rhs(f_1,c_1)\ y_{n_1+1} \dots y_{n_1+n_2}]\!] \\
 & & \vdots & \\
 & \widetilde{f_1 f_2}\ (c_q\ x_1 \dots x_{k_q})\ y_1 \dots y_{n_1+n_2} & = & \mathcal{D}[\![f_2\ rhs(f_1,c_q)\ y_{n_1+1} \dots y_{n_1+n_2}]\!]
\end{array}
$$

Wir bezeichnen das Transformationsergebnis (p', e') mit $Def(p, e)$. □

Lemma 2.29 Für alle $(p, e) \in (MAC; MAC)$ gilt $(p, e) \equiv Def(p, e)$. □

Wir verzichten auf den Beweis, machen aber einige Anmerkungen zu den Gemeinsamkeiten und Unterschieden unserer Notation zu den sonst in der Literatur verwendeten. Zunächst einmal terminiert $\mathcal{D}[\![.]\!]$, weil bei der Komposition zweier macro tree transducer M_1 und M_2 mit h_1 bzw. h_2 Funktionen höchstens $h_1 h_2 + h_1 + h_2$ Funktionen konstruiert werden. Die implizite let-Extraktion läßt sich auf die explizite Behandlungsweise mit Hilfe der dafür üblichen Regel von $\mathcal{D}[\![.]\!]$ (siehe z.B. [Wad90]) zurückführen, wie wir am Beispiel der Regel $(D4)$ zeigen wollen. Die Rechnung für die Regel $(D5)$ ist analog.

$$
\begin{array}{cl}
 & \mathcal{D}[\![f\ v\ t'_1 \dots t'_n]\!] \\
= & \mathcal{D}[\![\texttt{let}\ v_1 = t'_1, \dots, v_n = t'_n\ \texttt{in}\ (f\ v\ t'_1 \dots t'_n)]\!] \\
= & \texttt{let}\ v_1 = \mathcal{D}[\![t'_1]\!], \dots, v_n = \mathcal{D}[\![t'_n]\!]\ \texttt{in}\ \mathcal{D}[\![f\ v\ t'_1 \dots t'_n]\!] \\
= & \texttt{let}\ v_1 = \mathcal{D}[\![t'_1]\!], \dots, v_n = \mathcal{D}[\![t'_n]\!]\ \texttt{in}\ (\widetilde{f}\ v\ t'_1 \dots t'_n)\ \text{mit} \dots \\
= & \widetilde{f}\ v\ \mathcal{D}[\![t'_1]\!] \dots \mathcal{D}[\![t'_n]\!]\ \text{mit} \dots
\end{array}
$$

Im Gegensatz zu Regel (4) aus [SGJ94] führen wir in $(D3)$ keine neue Funktion ein. Zum einen sind die sogenannten *f-Funktionen* aus [SGJ94], welche nicht durch pattern matching definiert sind, bei macro tree transducern nicht erlaubt. Zum anderen führt dieser fold-Schritt nicht zu einer weiteren Elimination von Zwischenergebnissen, sondern nur gegebenenfalls zu einer früheren Termination der Transformation. Insgesamt wird die Termination jedoch nicht beeinträchtigt, da an den kritischen Stellen, an denen eine Fallunterscheidung für Variablen vorgenommen wird, neue Funktionen eingeführt werden.

Für Berechnungsausdrücke wie $(app\ z_0\ (app\ z_1\ z_2))$, welche keine Kompositionsausdrücke sind, liefert unser Algorithmus ein anderes Ergebnis als das originale Verfahren aus [Wad90], da $(app\ z_0\ (app\ z_1\ z_2))$ wie der Ausdruck $\texttt{let}\ v = (app\ z_1\ z_2)\ \texttt{in}\ (app\ z_0\ v)$ behandelt wird.

Zum Schluß zeigen wir die Durchführung des Verfahrens noch an zwei Beispielen.

Beispiel 2.30 Das Programm $p \in P$ bestehe aus den beiden tree transducern M_{ex} und M_{rev} aus den Beispielen 2.3 und 2.6, und $e = (rev\ (ex\ z_0)\ z_1) \in E_p$ ist ein Kompositionsausdruck für dieses Programm. Es gilt $(p, e) \in (TOP; MAC)$. Dann ist $e' = \mathcal{D}[\![rev\ (ex\ z_0)\ z_1]\!] = (exrev\ z_0\ \mathcal{D}[\![z_1]\!]) = (exrev\ z_0\ z_1)$ mit

$$\begin{array}{rcl@{\qquad}rcl}
exrev\ (A\ x_1)\ y_1 & = & \mathcal{D}[\![rev\ (rhs(ex, A))\ y_1]\!] & exrev\ (B\ x_1)\ y_1 & = & \ldots \\
& = & \mathcal{D}[\![rev\ (B\ (ex\ x_1))\ y_1]\!] & & = & exrev\ x_1\ (A\ y_1) \\
& = & \mathcal{D}[\![rev\ (ex\ x_1)\ (B\ y_1)]\!] & & & \\
& = & exrev\ x_1\ \mathcal{D}[\![(B\ y_1)]\!] & exrev\ N\ y_1 & = & y_1 \\
& = & exrev\ x_1\ (B\ \mathcal{D}[\![y_1]\!]) & & & \\
& = & exrev\ x_1\ (B\ y_1) & & &
\end{array}$$

Das neue Programm p' besteht aus diesen drei Gleichungen. Es ist $Def(p, e) = (p', e')$. Hier werden die Zwischenergebnisse eliminiert, da in den Gleichungen die Konstruktoren bereits ersetzt wurden. □

Beispiel 2.31 Das Programm $p \in P$ bestehe aus den beiden tree transducern M_{ex} und M_{rev} aus den Beispielen 2.3 und 2.6, und $e = (ex\ (rev\ z_0\ z_1)) \in E_p$ ist ein Kompositionsausdruck für dieses Programm. Es gilt $(p, e) \in (MAC; TOP)$. Dann ist $e' = \mathcal{D}[\![ex\ (rev\ z_0\ z_1)]\!] = (revex\ z_0\ \mathcal{D}[\![z_1]\!]) = (revex\ z_0\ z_1)$ mit

$$\begin{array}{rcl@{\qquad}rcl}
revex\ (A\ x_1)\ y_1 & = & \mathcal{D}[\![ex\ (rhs(rev, A))]\!] & revex\ N\ y_1 & = & \mathcal{D}[\![ex\ (rhs(rev, N))]\!] \\
& = & \mathcal{D}[\![ex\ (rev\ x_1\ (A\ y_1))]\!] & & = & \mathcal{D}[\![ex\ y_1]\!] \\
& = & revex\ x_1\ \mathcal{D}[\![(A\ y_1)]\!] & & = & ex\ y_1 \quad \text{mit} \\
& = & revex\ x_1\ (A\ \mathcal{D}[\![y_1]\!]) & & & \\
& & & ex & \text{ist} & \text{definiert wie in } M_{ex} \\
revex\ (B\ x_1)\ y_1 & = & \ldots & & & \\
& = & revex\ x_1\ (B\ y_1) & & &
\end{array}$$

Das neue Programm p' besteht aus diesen drei Gleichungen und den Gleichungen aus M_{ex}. Es ist $Def(p, e) = (p', e')$. Man beachte, daß die Zwischenergebnisse hier nicht entfernt werden, da der einzige Unterschied zwischen rev und $revex$ darin besteht, daß der Aufruf von ex auf das Resultat von rev in die Gleichung für N integriert wurde. □

Kapitel 3

Vergleiche für die Kompositionsklasse $(TOP; MAC)$

In diesem Kapitel betrachten wir Kompositionsausdrücke aus der Kompositionsklasse $(TOP; MAC)$. Wir werden untersuchen, welche Auswirkungen die Verfahren Deforestation und Komposition auf die Effizienz haben.

3.1 Bisherige Resultate

Die Äquivalenz von Deforestation und Komposition für die hier behandelte Kompositionsklasse wurde schon in [Küh99] festgestellt. Dort wurde jedoch nur eine Beweisidee angegeben, welche wir hier vollständig durchführen.

In [Wad90] wird behauptet, daß bei der Transformation mittels Deforestation von Ausdrücken, deren Funktionsdefinitionen in treeless form vorliegen, kein Effizienzverlust auftritt. Dabei wird unter anderem auch gefordert, daß alle Funktionen linear sind. Wegen der eben erwähnten Äquivalenz wurde dieses Resultat in [Küh99] für die Kompositionsklasse $(lin\text{-}TOP; lin\text{-}MAC)$ auf die Komposition übertragen.

Wir wollen hier versuchen, die Betrachtungen auch auf den nichtlinearen Fall auszudehnen und die möglichen Effizienzsteigerungen quantitativ anzugeben und zu beweisen.

3.2 Die Äquivalenz von Deforestation und Komposition

Um die Komposition besser behandeln zu können, führen wir zunächst eine übersichtlichere Notation ein, welche die schrittweise Berechnung des Kompositionsergebnisses erleichtert.

Definition 3.1 Seien $k_1, k_2, n_2 \in \mathbb{N}$ und R' die Menge, die aus $RHS(F_2, C_2, k_2, n_2)$ entsteht, indem alle Rekursionsvariablen durch Elemente aus $RHS(F_1, C_1, k_1, 0)$ und alle Kontextvariablen durch Elemente aus $RHS(F_2, C_2, k_2, n_2)$ ersetzt werden. Weiterhin sei $i \in [k_1]$, $j \in [n_2]$, $k \in [k_1]$, $c_1^{(k)} \in C_1$, $t_1, \dots, t_k \in RHS(F_1, C_1, k_1, 0)$, $n \in [n_2]$, $f_2^{(1+n)} \in F_2$ und $t'_1, \dots, t'_n \in RHS(F_2, C_2, k_2, n_2)$. Schließlich sei $k' \in [k_2]$, $c_2^{(k')} \in C_2$ und $t''_1, \dots, t''_{k'} \in R'$ oder $t''_1, \dots, t''_{k'} \in RHS(F_2, C_2, k_2, n_2)$. Die Transformationsfunktion $\mathcal{C}[\![.]\!] : R' \to RHS(F_1 \times F_2, C_2, k_1, n_2)$ sei durch die nachfolgend angegebenen Transformationsregeln definiert.

$$
\begin{array}{lllll}
(C1) & \mathcal{C}[\![y_j]\!] & = & y_j & \\
(C2) & \mathcal{C}[\![(c_2\ t''_1 \ldots t''_{k'})]\!] & = & (c_2\ \mathcal{C}[\![t''_1]\!] \ldots \mathcal{C}[\![t''_{k'}]\!]) & \\
(C3) & \mathcal{C}[\![f_2\ (c_1\ t_1 \ldots t_k)\ t'_1 \ldots t'_n]\!] & = & \mathcal{C}[\![rhs(f_2, c_1)[x_i/t_i][y_j/t'_j]]\!] & \\
(C4) & \mathcal{C}[\![f_2\ (f_1\ x_i)\ t'_1 \ldots t'_n]\!] & = & \overline{f_1 f_2}\ x_i\ \mathcal{C}[\![t'_1]\!] \ldots \mathcal{C}[\![t'_n]\!] & \square
\end{array}
$$

Den Zusammenhang zwischen dieser Notation und Transformation 2.22 zeigt folgendes Lemma.

Lemma 3.2 Seien $M_1 \in TOP$ und $M_2 \in MAC$, aus denen entsprechend Punkt 1. der Transformation 2.22 der macro tree transducer M'_2 entstehe. Für alle $f \in F^{(1)}_{M_1}$, $c \in C^i{}_{M_1}$ und $g \in F^{(1+n)}_{M_2}$ mit $n \in I\!N$ gilt:

$$nf(\Rightarrow_{M'_2}, g\ (rhs(f,c))\ y_1 \ldots y_n) = \mathcal{C}[\![g\ (rhs(f,c))\ y_1 \ldots y_n]\!].$$

Beweis: Seien $k_1, k_2, n_2 \in I\!N$ und R' die Menge, die aus $RHS(F_2, C_2, k_2, n_2)$ entsteht, indem alle Rekursionsvariablen durch Elemente aus $RHS(F_1, C_1, k_1, 0)$ und alle Kontextvariablen durch Elemente aus $RHS(F_2, C_2, k_2, n_2)$ ersetzt werden. Wir beweisen die allgemeinere Aussage $nf(\Rightarrow_{M'_2}, r') = \mathcal{C}[\![r']\!]$ für alle $r' \in R'$ durch strukturelle Induktion über den Aufbau der Menge R'.

- $r' = y_j$ mit $j \in [n_2]$: Da Kontextvariablen gemäß der Definition von R' durch ein $r \in RHS(F_2, C_2, k_2, n_2)$ ersetzt werden, ist eine Fallunterscheidung entsprechend des Aufbaus der Menge $RHS(F_2, C_2, k_2, n_2)$ nötig:
  - $r = y_j$ mit $j \in [n_2]$: Kontextvariablen sind nullstellige Ausgabesymbole von M'_2, weswegen $nf(\Rightarrow_{M'_2}, y_j) = y_j$ gilt. Regel $(C1)$ berechnet ebenfalls die Identität.
  - $r = (c\, t_1 \ldots t_k)$ mit $k \in [k_2]$, $c^{(k)} \in C_2$ und $t_1, \ldots, t_k \in RHS(F_2, C_2, k_2, n_2)$: Die Nachfolger von Konstruktoren werden gemäß Punkt 2. von Definition 2.7 der Reduktionsrelation von links nach rechts reduziert. Die dabei entstehenden Normalformen sind wegen $RHS(F_2, C_2, k_2, n_2) \subseteq R'$ gleich den Ergebnissen der Funktion $\mathcal{C}[\![.]\!]$. Damit entspricht dieser Fall der Regel $(C2)$.
  - $r = (f_2\ x_i\ t'_1 \ldots t'_n)$ mit $n \in [n_2]$, $f_2^{(1+n)} \in F_2$, $i \in [k_2]$ und $t'_1, \ldots, t'_n \in R'$: Dieser Fall kommt beim Start der Berechnung mit dem Ausdruck aus dem hier zu beweisenden Lemma nicht vor, ansonsten wäre noch eine weitere Regel erforderlich.
- $r' = (c\, t_1 \ldots t_k)$ mit $k \in [k_2]$, $c^{(k)} \in C_2$ und $t_1, \ldots, t_k \in R'$: Die Nachfolger von Konstruktoren werden gemäß Punkt 2. von Definition 2.7 der Reduktionsrelation von links nach rechts reduziert. Da diese Bäume in R' sind, gleichen die dabei entstehenden Normalformen den Ergebnissen der Funktion $\mathcal{C}[\![.]\!]$. Damit entspricht dieser Fall der Regel $(C2)$.
- $r' = (f_2\ x_i\ t'_1 \ldots t'_n)$ mit $n \in [n_2]$, $f_2^{(1+n)} \in F_2$, $i \in [k_2]$ und $t'_1, \ldots, t'_n \in R'$: Bei Funktionsaufrufen wird nach Definition von R' die Rekursionsvariable durch ein $r \in RHS(F_1, C_1, k_1, 0)$ ersetzt, daher ist eine Fallunterscheidung bezüglich des Aufbaus von $RHS(F_1, C_1, k_1, 0)$ erforderlich:
  - $r = (c\, t_1 \ldots t_k)$ mit $k \in [k_1]$, $c^{(k)} \in C_1$ und $t_1, \ldots, t_k \in RHS(F_1, C_1, k_1, 0)$: Funktionsaufrufe auf einen Konstruktor werden nach Punkt 1. von Definition

2.7 der Reduktionsrelation durch ihre rechte Seite ersetzt, wobei die Rekursionsvariablen und Kontextparameter durch die aktuellen Bäume ersetzt werden. Der dadurch entstehende Ausdruck ist in R', weshalb dessen Normalform gleich dem Ergebnis der Funktion $\mathcal{C}[\![.]\!]$ ist. Damit entspricht dieser Fall der Regel ($C3$).

- $r = (f_1\ x_i)$ mit $f_1 \in F_1$ und $i \in [k_1]$: Die Funktion f_1 wird entsprechend Punkt 1. von Transformation 2.22 in den Regeln von M_2' als Konstruktor behandelt. Damit erfolgt die Reduktion gemäß Punkt 1. von Definition 2.7 der Reduktionsrelation, wobei die Rekursionsvariablen identisch repliziert werden. Die für die Kontextvariablen eingesetzten Bäume werden noch weiter reduziert. Deren Normalformen sind aber gleich den Ergebnissen der Funktion $\mathcal{C}[\![.]\!]$, weil diese Bäume in R' sind. Damit entspricht dieser Fall der Regel ($C4$).

Damit haben wir bewiesen, daß die Aussage für alle möglichen Fälle gilt. □

Nun zeigen wir, daß Deforestation und Komposition bei der hier behandelten Kompositionsklasse äquivalent sind.

Satz 3.3 Für alle $(p, e) \in (TOP; MAC)$ gilt: $Def(p, e) = Com(p, e)$.
(abgesehen von den unterschiedlichen Namenskonventionen für Funktionen und der Erzeugung überflüssiger Funktionen bei der Komposition)

Beweis: Seien $M_1 = (F^{(1)}_{M_1}, C^i_{M_1}, C^o_{M_1}, R_{M_1}) \in TOP$ und $M_2 = (F_{M_2}, C^i_{M_2}, C^o_{M_2}, R_{M_2}) \in MAC$ zwei tree transducer, die Bestandteile eines Programmes $p \in P$ sind. Die Funktionen $f_1 f_2$ und $\overline{f_1 f_2}$ mit $f_1 \in F_{M_1}$ und $f_2 \in F_{M_2}$ sehen wir im folgenden als gleich an. Wir betrachten den Kompositionsausdruck $e = g\ (f\ z_0)\ z_1 \ldots z_n$ mit $f \in F_{M_1}$, $g \in F^{(1+n)}_{M_2}$ und $n \geq 0$. Es sei $(p_d', e_d') = Def(p, e)$ und $(p_c', e_c') = Com(p, e)$. Zuerst zeigen wir die Gleichheit von e_d' und e_c'. Nach Punkt 3. von Transformation 2.22 ergibt sich bei der Komposition unmittelbar der Ergebnisausdruck $e_c' = \overline{fg}\ z_0\ z_1 \ldots z_n$. Deforestation muß mit dem Kompositionsausdruck gestartet werden, um den Ergebnisausdruck e_d' zu erhalten:

$$\begin{aligned} e_d' &= \mathcal{D}[\![e]\!] \\ &= \mathcal{D}[\![g\ (f\ z_0)\ z_1 \ldots z_n]\!] \\ &= fg\ z_0\ \mathcal{D}[\![z_1]\!] \ldots \mathcal{D}[\![z_n]\!] \\ &= fg\ z_0\ z_1 \ldots z_n \end{aligned}$$

Damit ist die Gleichheit der Ergebnisausdrücke für beide Verfahren bewiesen. Jetzt zeigen wir die Gleichheit der neuen Programme. Das bei der Komposition entstehende Programm p_c' enthält laut Punkt 2. aus Transformation 2.22 unter Berücksichtigung von Lemma 3.2 für alle $f_1 \in F_{M_1}$, $f_2 \in F^{(1+n_2)}_{M_2}$ mit $n_2 \geq 0$ und $c^{(k)} \in C^i_{M_1}$ mit $k \geq 0$ die Funktionsgleichung

$$\overline{f_1 f_2}\ (c\ x_1 \ldots x_k)\ y_1 \ldots y_{n_2} = \mathcal{C}[\![f_2\ (rhs(f_1, c))\ y_1 \ldots y_{n_2}]\!].$$

Bei Deforestation entsteht das neue Programm p_d', indem alle in den Regeln ($D4$) und ($D5$) aus Transformation 2.28 erzeugten Funktionsgleichungen aufgesammelt werden. Hierbei werden im Gegensatz zur Komposition allerdings nur tatsächlich benötigte Gleichungen generiert. Weil Regel ($D4$), wie wir weiter unten zeigen werden, nicht verwendet wird, haben diese Gleichungen laut Regel ($D5$) die Form

$$f_1 f_2\ (c\ x_1 \ldots x_k)\ y_1 \ldots y_{n_2} = \mathcal{D}[\![f_2\ (rhs(f_1, c))\ y_1 \ldots y_{n_2}]\!].$$

Die Gleichheit dieser Funktionsgleichungen ist gegeben, wenn die Transformationsfunktionen $\mathcal{C}$ und $\mathcal{D}$ den selben Ergebnisausdruck liefern. Dazu vergleichen wir im folgenden die Regeln beider Funktionen.

- Weil die einzigen bei der Komposition zu transformierenden Variablen aus der Menge der Kontextvariablen stammen, sind die Regeln $(D1)$ und $(C1)$ gleich.
- Die Regeln $(D2)$ und $(C2)$ sind gleich.
- Die Regeln $(D3)$ und $(C3)$ sind gleich.
- Die Regel $(D4)$ wird für die Transformation des betrachteten Kompositionsausdrucks e mittels Deforestation nicht benötigt: Gestartet wird das Verfahren mit einer Anwendung von Regel $(D5)$, danach werden die Ausdrucksvariablen mit Regel $(D1)$ behandelt. Für die Berechnung der rechten Seiten von neuen Funktionen in $(D5)$ mittels $\mathcal{D}[\![f_2\ (rhs(f_1, c))\ y_1 \ldots y_{n_2}]\!]$ werden nur die Regeln $(D3)$ und $(D5)$ gebraucht, weil die rechten Seiten eines top-down tree transducers aus Konstruktoren und Funktionsaufrufen bestehen. Regel $(D3)$ schließlich erfordert nach ihrer Anwendung die Transformation der rechten Seite einer Funktionsgleichung des macro tree transducers M_2, wobei die darin vorkommenden Variablen substituiert werden. Dazu sind nur die Regeln $(D1)$ für Kontextvariablen, $(D2)$ für Konstruktoren und $(D3)$ bzw. $(D5)$ für Funktionsaufrufe erforderlich, da das Rekursionsargument dieses Funktionsaufrufs durch die vorangegangene Substitution ein Teilterm einer rechten Seite einer Regel des top-down tree transducers M_1 ist und damit keine Kontextvariable sein kann.
- Die Regeln $(D5)$ und $(C4)$ erzeugen den gleichen Ausdruck, weil M_1 ein top-down tree transducer ist und demnach keine Kontextparameter hat, weshalb $n_1 = 0$ für $(D5)$ gilt. Weil bei der Komposition gemäß Punkt 2. von Transformation 2.22 für alle möglichen neuen Funktionen Gleichungen erzeugt werden, ist dies in der Regel $(C4)$ nicht mehr nötig. Im Gegensatz dazu erzeugt Deforestation in Regel $(D5)$ die Gleichungen für die dort im Ergebnisausdruck verwendete neue Funktion. Dieser Unterschied hat auf den Ergebnisausdruck jedoch keinen Einfluß.

Die beiden Transformationsfunktionen sind also gleich, weshalb abgesehen von den überflüssigen Funktionen bei der Komposition die gleichen Programme entstehen. Damit ist der Satz bewiesen. □

Korollar 3.4 Für alle $(p, e) \in (TOP; MAC)$ gilt: $\mathcal{E}(Def(p, e)) = \mathcal{E}(Com(p, e))$. □

Deshalb müssen wir nur noch für eines der beiden Verfahren untersuchen, wie sich die Effizienz des Transformationsergebnisses gegenüber dem ursprünglichen Ausdruck und Programm verhält. Für das andere Verfahren gelten die Resultate dann ebenso. Wir entscheiden uns für die Untersuchung der Komposition, weil die explizite Erzeugung des neuen Programms besser zu überschauen ist.

Die nachfolgenden Effizienzuntersuchungen beruhen gemäß Definition 2.13 auf dem Vergleich der Anzahl von Reduktionsschritten. Bei macro tree transducern wird deren Berechnung dadurch erschwert, daß Funktionsaufrufe auch in den Kontextparametern auftreten können. Falls Regeln vorhanden sind, welche deleting in den Kontextparametern sind, werden diese Aufrufe bei der Reduktion möglicherweise ignoriert. Wenn an einem Vorkommen des Eingabebaums eine Regel, die einen verschachtelten Funktionsaufruf enthält, angewendet wird, kann zu diesem Zeitpunkt noch keine Aussage darüber getroffen werden, ob die Funktionsaufrufe im Kontext tatsächlich durchgeführt werden. Dies ergibt sich erst im weiteren Verlauf der Reduktion in Abhängigkeit von der Gestalt des darunterliegenden Teils des Eingabebaums.

Beispiel 3.5 Gegeben sei ein Programm, welches aus einem auf monadischen Bäumen arbeitenden tree transducer $M_{id/exp}$ besteht. Die (g, α_0)-Regel ist deleting im Kontextparameter, die (g, α_1)-Regel aber nicht.

$$
\begin{array}{lcl}
f\ (\gamma\ x_1)\ y_1 & = & \gamma(f\ x_1\ (f\ x_1\ y_1)) \\
f\ \alpha_1\ y_1 & = & y_1 \\
f\ \alpha_0\ y_1 & = & \alpha
\end{array}
$$

Das Verhalten von $M_{id/exp}$ hängt vom untersten Symbol des Eingabebaums ab. Für Bäume, die mit α_0 abschließen, berechnet $M_{id/exp}$ die Identität, während er bei mit α_1 endenden Bäumen die Zahl der γ-Symbole exponentiell vervielfacht.

Nun betrachten wir die Eingabebäume $\gamma^n\alpha_1$ und $\gamma^n\alpha_0$. Der Kontextparameter sei dabei jeweils mit α belegt. Die Abbildung 3.1 zeigt diese Eingabebäume und die durch deren Übersetzung entstehenden Ausgabebäume.

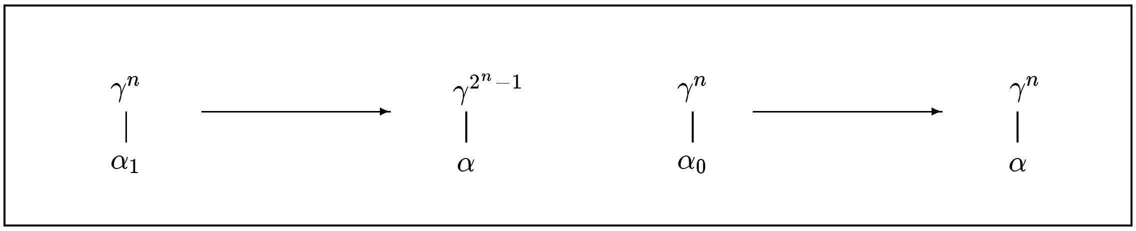

Abbildung 3.1: Auswirkungen der deleting-Eigenschaft im Kontextparameter

Wenn ein mit γ beschriftetes Symbol durch f besucht wird, steht noch nicht fest, ob der verschachtelte Funktionsaufruf durchgeführt wird. Dies entscheidet sich jeweils erst, wenn das unterste Symbol erreicht wird. □

Eine Ausnahme bilden macro tree transducer mit der nondeleting-Eigenschaft in den Kontextparametern, weil diese keine Regeln, welche Kontextparameter löschen, besitzen. Hier ist unabhängig vom Aussehen des Eingabebaums garantiert, daß bei Anwendung einer Regel alle Funktionsaufrufe im Kontext aufgrund der Auswertung mittels call-by-need genau einmal durchgeführt werden. Dies erleichtert die Berechnung der Anzahl der Reduktionsschritte erheblich, weil diese nur vom Aussehen der verwendeten Regeln abhängt. Deswegen behandeln wir die Kompositionsklassen $(TOP; ndc\text{-}MAC)$ und $(TOP; dc\text{-}MAC)$ getrennt. Dabei ist zu beachten, daß die Kompositionsklasse $(TOP; TOP)$ eine Teilklasse von $(TOP; ndc\text{-}MAC)$ ist, weil top-down tree transducer keine Kontextparameter besitzen.

3.3 Die Kompositionsklasse $(TOP; ndc\text{-}MAC)$

3.3.1 Notationen und Hilfsaussagen

Zur Beschreibung des Aussehens der rechten Regelseiten zwecks Berechnung der Anzahl der Reduktionsschritte brauchen wir folgende Notation.

Definition 3.6 Sei $i \in [k]$. Für die *Häufigkeit des Auftretens von Aufrufen der Funktion f mit dem Rekursionsargument x_i in $rhs(f',c)$* schreiben wir $n(f,i,f',c)$ und definieren sie als

$$n(f,i,f',c) = card\{o \mid o \in occ(rhs(f',c)), \\ label(o, rhs(f',c)) = f,\ label(o.1, rhs(f',c)) = x_i\}. \qquad \square$$

Das folgende Lemma formuliert für macro tree transducer mit der nondeleting-Eigenschaft in den Kontextparametern eine nur auf das eben definierte Aussehen der rechten Regelseiten zurückgreifende rekursive Berechnungsvorschrift für die Zahl der Besuche eines Vorkommens durch eine Funktion.

Lemma 3.7 Sei $M = (F, C^i, C^o, R) \in ndc\text{-}MAC$, $f_0, f' \in F$, $t \in T_{C^i}$, $o \in occ(t)$ und $label(o,t) = c^{(k)}$ mit $k > 0$. Dann gilt für alle $i \in [k]$

$$m_{f'}^{f_0,t}(o.i) = \sum_{f \in F} \left(m_f^{f_0,t}(o) \cdot n(f', i, f, c) \right).$$

Beweis: Sei $M = (F, C^i, C^o, R) \in ndc\text{-}MAC$, $t \in T_{C^i}$, $o \in occ(t)$ und $label(o,t) = c^{(k)}$ mit $k > 0$. Weil M nondeleting in den Kontextparametern ist, wird bei Anwendung einer Regel keiner der in der rechten Seite dieser Regel vorkommenden Funktionsaufrufe ignoriert. Wenn auf das Vorkommen o die Funktion $f \in F^{(1+n)}$ mit $n \in I\!N$ angewendet wird, so muß die (f,c)-Regel herangezogen werden. Jeder Funktionsaufruf der Form $(f' x_i\ t_1 \ldots t_{n'})$ in dieser Regel mit $f' \in F^{(1+n')}$, $i \in [k]$, $n' \in I\!N$ und $t_1, \ldots, t_{n'} \in RHS(F, C^o, k, n)$ hat wegen der Auswertung mittels call-by-need genau einen Besuch des i-ten Nachfolgers von c durch f' zur Folge. Da o mehrfach von f besucht werden kann, muß die Anzahl dieser Besuche mit der Häufigkeit der gerade diskutierten Funktionsaufrufe in der (f,c)-Regel multipliziert werden. Wenn man diese Rechnung für alle $f \in F$ durchführt, so ergibt sich die Gesamtanzahl der Besuche durch Summierung der Ergebnisse. $\square$

Nun wenden wir uns der Untersuchung der Komposition zu. Im folgenden sei $M_1 = (F_{M_1}^{(1)}, C_{M_1}^i, C_{M_1}^o, R_{M_1}) \in TOP$ und $M_2 = (F_{M_2}, C_{M_2}^i, C_{M_2}^o, R_{M_2}) \in ndc\text{-}MAC$. Wir betrachten den Kompositionsausdruck $g_0\ (f_0\ z_0)\ z_1 \ldots z_{n_2}$ mit $f_0^{(1)} \in F_{M_1}$, $g_0^{(1+n_2)} \in F_{M_2}$ und $n_2 \geq 0$. Daraus entstehe durch Komposition gemäß Transformation 2.22 der neue macro tree transducer $M = (F_M, C_M^i, C_M^o, R_M)$ und der neue Ausdruck $(\overline{f_0 g_0}\ z_0\ z_1 \ldots z_{n_2})$ mit $\overline{f_0 g_0}^{(1+n_2)} \in F_M$. Weiterhin seien $t_e \in T_{C_{M_1}^i}$ und $t_1, \ldots, t_{n_2} \in T_{C_{M_2}^o}$ die Bäume, die wir für z_0 bzw. $z_1, \ldots, z_{n_2}$ einsetzen, und $t_z = nf(\Rightarrow_{M_1}, (f_0\ t_e))$ das durch M_1 erzeugte Zwischenergebnis.

Zunächst zeigen wir, daß bei der Komposition die nondeleting-Eigenschaft in den Kontextparametern vom zweiten tree transducer auf den resultierenden tree transducer übertragen wird.

Lemma 3.8 Für den durch die Komposition entstehenden macro tree transducer gilt: $M \in ndc\text{-}MAC$.

Beweis: Bei der Komposition entsteht für alle $f \in F_{M_1}^{(1)}$, $g \in F_{M_2}^{(1+n)}$ und $c \in {C^i}_{M_1}^{(k)}$ die rechte Seite der $(\overline{fg}, c)$-Regel durch Berechnung von $nf(\Rightarrow_{M_2'}, g\ (rhs(f,c))\ y_1 \ldots y_n)$. Mit Notation 3.1 ist wegen Lemma 3.2 also

$$rhs(\overline{fg}, c) = \mathcal{C}[\![g\ (rhs(f,c))\ y_1 \ldots y_n]\!].$$

Wir sehen, daß diese Berechnung mit allen Kontextvariablen aus Y_n initialisiert wird. Im folgenden zeigen wir anhand der Regeln für die Transformationsfunktion $\mathcal{C}$, daß keine dieser Variablen während der Berechnung verloren geht.

- Die Regel $(C1)$ beschreibt eine identische Abbildung für Kontextvariablen.
- $(C2)$ transformiert alle Teilbäume eines Konstruktors, dabei geht also nichts verloren.
- In der Regel $(C3)$ werden unter anderem in der rechten Seite einer Regel von M_2 alle Kontextvariablen durch Bäume ersetzt, welche die Kontextvariablen der Initialisierung enthalten. Wegen $M_2 \in ndc\text{-}MAC$ sind in dieser rechten Seite alle Kontextvariablen vorhanden, so daß keiner der Bäume (und damit keine Kontextvariable der Initialisierung) verloren geht.
- Auch in $(C4)$ werden alle Bäume, die Kontextvariablen enthalten können, weiterverwendet.

Damit sind alle Kontextvariablen, mit denen die Berechnung gestartet wurde, im Ergebnis enthalten. Die rechte Seite der $(\overline{fg}, c)$-Regel ist also nondeleting in den Kontextparametern. Weil dies für alle Regeln von M gilt, ist $M \in ndc\text{-}MAC$. □

Das eben bewiesene Lemma benötigen wir, um den folgenden wichtigen Zusammenhang zwischen der Besuchsanzahl eines Vorkommens im Eingabebaum durch eine Funktion des Kompositionsergebnisses und der Besuchsanzahl dessen Bildes im Zwischenergebnis durch eine Funktion des zweiten tree transducers zu zeigen.

Lemma 3.9 Sei $f \in F_{M_1}$, $g \in F_{M_2}$, $\overline{fg} \in F_M$ und $o_e \in occ(t_e)$. Dann gilt:

$$m_{\overline{fg}}^{\overline{f_0 g_0}, t_e}(o_e) = \sum_{o_z \in bild_f^{f_0, t_e}(o_e)} m_g^{g_0, t_z}(o_z)$$

Beweis: Wir beweisen das Lemma durch Induktion über alle Pfade in t_e:

Induktionsanfang ($o_e = \varepsilon$):

Für die linke Seite der Gleichung gilt:

$$m_{\overline{fg}}^{\overline{f_0 g_0}, t_e}(\varepsilon) = \begin{cases} 1 & \text{falls } \overline{f_0 g_0} = \overline{fg} \\ 0 & \text{falls } \overline{f_0 g_0} \neq \overline{fg} \end{cases}$$

Die rechte Seite berechnen wir in mehreren Schritten:

$$bild_f^{f_0, t_e}(\varepsilon) = \begin{cases} \{\varepsilon\} & \text{falls } f_0 = f \\ \emptyset & \text{falls } f_0 \neq f \end{cases}$$

$$m_g^{g_0, t_z}(\varepsilon) = \begin{cases} 1 & \text{falls } g_0 = g \\ 0 & \text{falls } g_0 \neq g \end{cases}$$

$$\sum_{o_z \in bild_f^{f_0, t_e}(\varepsilon)} m_g^{g_0, t_z}(o_z) = \begin{cases} 1 & \text{falls } f_0 = f \text{ und } g_0 = g \\ 0 & \text{falls } f_0 \neq f \text{ oder } g_0 \neq g \end{cases}$$

Weil $\overline{f_0 g_0} = \overline{fg}$ genau dann gilt, wenn $f_0 = f$ und $g_0 = g$, sind die Ausdrücke auf der linken und rechten Seite gleich.

Induktionsschritt: Nun sei $label(o_e, t_e) = c^{(k)}$, $k > 0$, $c \in C^i_{M_1}$, $i \in [k]$, und als Induktionsvoraussetzung gelte für o_e die zu beweisende Beziehung. Wir gehen von o_e zum i-ten Nachfolger. Da wegen Lemma 3.8 mit M_2 auch M nondeleting in den Kontextparametern ist, ergibt sich mit Hilfe von Lemma 3.7 folgender rekursiver Ansatz.

$$m^{\overline{f_0 g_0}, t_e}_{\overline{f'g'}}(o_e.i) = \sum_{f \in F_{M_1}, g \in F_{M_2}} \left(m^{\overline{f_0 g_0}, t_e}_{\overline{fg}}(o_e) \cdot n(\overline{f'g'}, i, \overline{fg}, c) \right)$$

Unter Verwendung der Induktionsvoraussetzung ergibt sich

$$= \sum_{f \in F_{M_1}, g \in F_{M_2}} \left(\left(\sum_{o_z \in bild^{f_0, t_e}_{f}(o_e)} m^{g_0, t_z}_{g}(o_z) \right) \cdot n(\overline{f'g'}, i, \overline{fg}, c) \right)$$

Weil die rechte Seite der $(\overline{fg}, c)$-Regel durch Übersetzung der (f, c)-Regel mit g entsteht, wird ein Aufruf der Funktion $\overline{f'g'}$ auf den i-ten Nachfolger von c genau dann erzeugt, wenn dabei ein mit $(f' \; x_i)$ beschriftetes Vorkommen durch g' besucht wird. Dies rechtfertigt folgende Umformung:

$$= \sum_{f \in F_{M_1}, g \in F_{M_2}} \left(\left(\sum_{o_z \in bild^{f_0, t_e}_{f}(o_e)} m^{g_0, t_z}_{g}(o_z) \right) \cdot \left(\sum_{\substack{subtree(o, rhs(f,c)) \\ = (f' \; x_i)}} m^{g, rhs(f,c)}_{g'}(o) \right) \right)$$

Durch Ausmultiplizieren der beiden Summen erhält man

$$= \sum_{f \in F_{M_1}, g \in F_{M_2}} \; \sum_{\substack{o_z \in bild^{f_0, t_e}_{f}(o_e) \\ subtree(o, rhs(f,c)) = (f' \; x_i)}} \left(m^{g_0, t_z}_{g}(o_z) \cdot m^{g, rhs(f,c)}_{g'}(o) \right)$$

Die nun folgende Ersetzung von $rhs(f, c)$ durch $subtree(o_z, t_z)$ ist möglich, weil sich in t_z auf dem Pfad von o_z nach $o_z.o$ genau die Konstruktoren befinden, wie in $rhs(f, c)$ auf dem Pfad von der Wurzel zum Vorkommen o. Dies liegt daran, daß o_z durch die Übersetzung von o_e mit f entstanden ist. Außerdem kann die Summe über alle $g \in F_{M_2}$ nach innen gezogen werden, weil die Quantifizierung der inneren Summe nicht von g abhängt:

$$= \sum_{f \in F_{M_1}} \; \sum_{\substack{o_z \in bild^{f_0, t_e}_{f}(o_e) \\ subtree(o, rhs(f,c)) = (f' \; x_i)}} \; \sum_{g \in F_{M_2}} \left(m^{g_0, t_z}_{g}(o_z) \cdot m^{g, subtree(o_z, t_z)}_{g'}(o) \right)$$

Nun kann der Pfad $o_z.o$ zu einem neuen Vorkommen o'_z zusammengezogen werden. Die Gesamtzahl der Besuche dieses Vorkommens durch g' ergibt sich durch entsprechende Summation über alle $g \in F_{M_2}$:

$$= \sum_{f \in F_{M_1}} \; \sum_{\substack{o_z \in bild^{f_0, t_e}_{f}(o_e) \\ subtree(o, rhs(f,c)) = (f' \; x_i)}} m^{g_0, t_z}_{g'}(\underbrace{o_z.o}_{o'_z})$$

Der in der rechten Seite der (f, c)-Regel im Vorkommen o stehende Funktionsaufruf $(f' \; x_i)$ veranlaßt einen Besuch von $o_e.i$ im Eingabebaum durch f'. Das dabei in t_z entstehende

Bildvorkommen ist $o_z.o$, also o'_z. Betrachtet man die (f,c)-Regeln für alle $f \in F_{M_1}$, so entsteht auf diese Weise die komplette Bildmenge $bild_{f'}^{f_0,t_e}(o_e.i)$.

$$= \sum_{o'_z \in bild_{f'}^{f_0,t_e}(o_e.i)} m_{g'}^{g_0,t_z}(o'_z)$$

Weil aus der Gültigkeit der zu beweisenden Aussage für ein Vorkommen $o_e \in occ(t_e)$ auf die Gültigkeit für dessen Nachfolger geschlossen werden konnte, gilt die Aussage für alle $o_e \in occ(t_e)$. □

Aus Lemma 3.9 läßt sich ein ähnlicher Zusammenhang für die Gesamtanzahl der Besuche eines Vorkommens im Eingabebaum bzw. Zwischenergebnis herleiten.

Lemma 3.10 Sei $o_e \in occ(t_e)$. Dann gilt:

$$m^{\overline{f_0 g_0},t_e}(o_e) = \sum_{o_z \in bild^{f_0,t_e}(o_e)} m^{g_0,t_z}(o_z)$$

Beweis:

$$\begin{aligned}
m^{\overline{f_0 g_0},t_e}(o_e) &= \sum_{f \in F_{M_1}, g \in F_{M_2}} m_{\overline{fg}}^{\overline{f_0 g_0},t_e}(o_e) && \text{(Definition 2.15)} \\
&= \sum_{f \in F_{M_1}, g \in F_{M_2}} \left(\sum_{o_z \in bild_f^{f_0,t_e}(o_e)} m_g^{g_0,t_z}(o_z) \right) && \text{(Lemma 3.9)} \\
&= \sum_{f \in F_{M_1}} \sum_{o_z \in bild_f^{f_0,t_e}(o_e)} \left(\sum_{g \in F_{M_2}} m_g^{g_0,t_z}(o_z) \right) && \text{(Summe nach innen)} \\
&= \sum_{f \in F_{M_1}} \left(\sum_{o_z \in bild_f^{f_0,t_e}(o_e)} m^{g_0,t_z}(o_z) \right) && \text{(Definition 2.15)} \\
&= \sum_{o_z \in bild^{f_0,t_e}(o_e)} m^{g_0,t_z}(o_z) && \text{(Definition 2.16, Lemma 2.17)}
\end{aligned}$$
□

Bei der Komposition von M_1 und M_2 werden im Falle des Vorhandenseins der deleting-Eigenschaft beim zweiten tree transducer wegen der call-by-need-Strategie nur die Teile des Zwischenergebnisses erzeugt, die auch tatsächlich benötigt werden. Um diesen Fall behandeln zu können, modifizieren wir unsere Notationen für die Besuchsanzahl, die Bildmengen und die Anzahl der Reduktionsschritte.

Definition 3.11 Sei $f \in F_{M_1}$ und $o_e \in occ(t_e)$. Wir bezeichnen mit $m_f^{f_0 \circ g_0,t_e}(o_e)$ die Anzahl der Besuche des Vorkommens o_e durch die Funktion f, deren dabei entstehenden Bilder in t_z von einem $g \in F_{M_2}$ besucht werden. Für die Menge der Bilder des Vorkommens o_e bezüglich f, die in t_z von einem $g \in F_{M_2}$ besucht werden, schreiben wir $bild_f^{f_0 \circ g_0,t_e}(o_e)$.

Die Gesamtanzahl $m^{f_0 \circ g_0,t_e}(o_e)$ der Besuche des Vorkommens o_e, deren dabei entstehenden Bilder in t_z von einem $g \in F_{M_2}$ besucht werden, und die Menge $bild^{f_0 \circ g_0,t_e}(o_e)$ der Bilder des Vorkommens o_e, die in t_z von einem $g \in F_{M_2}$ besucht werden, definieren wir analog zu den Definitionen 2.15 und 2.16.

Die Anzahl der Reduktionsschritte, die zur Erzeugung des von g_0 konsumierten Zwischenergebnisses benötigt werden, bezeichnen wir mit $r^{f_0 \circ g_0, t_e}$ und definieren sie analog zu Lemma 2.20 mit Hilfe der Besuchsanzahl als

$$r^{f_0 \circ g_0, t_e} = \sum_{o_e \in occ(t_e)} m^{f_0 \circ g_0, t_e}(o_e)$$

□

Nun verfügen wir über alle Notationen und Hilfsaussagen, die wir zum Beweis der nachfolgenden Resultate brauchen.

3.3.2 Die Kompositionsklasse $(TOP; lin\text{-}ndc\text{-}MAC)$

$M_1 = (F_{M_1}^{(1)}, C_{M_1}^i, C_{M_1}^o, R_{M_1}) \in TOP$ und $M_2 = (F_{M_2}, C_{M_2}^i, C_{M_2}^o, R_{M_2}) \in lin\text{-}ndc\text{-}MAC$ seien im folgenden zwei beliebige tree transducer mit den genannten Eigenschaften. Wir betrachten den Kompositionsausdruck $g_0\ (f_0\ z_0)\ z_1 \ldots z_{n_2}$ mit $f_0 \in F_{M_1}$, $g_0 \in F_{M_2}^{(1+n_2)}$ und $n_2 \geq 0$, aus dem nach Transformation 2.25 der neue macro tree transducer $M = (F_M, C_M^i, C_M^o, R_M)$ und der neue Ausdruck $(\overline{f_0 g_0}\ z_0\ z_1 \ldots z_{n_2})$ mit $\overline{f_0 g_0}^{(1+n_2)} \in F_M$ entstehen. Weiterhin sei $t_e \in T_{C_{M_1}^i}$ und $t_z = nf(\Rightarrow_{M_1}, (f_0\ t_e))$.

Lemma 3.12 $r^{\overline{f_0 g_0}, t_e} = r^{f_0 \circ g_0, t_e}$

Beweis: Die Anzahl der Reduktionsschritte für das Kompositionsergebnis berechnen wir mit Hilfe von Lemma 2.20:

$$r^{\overline{f_0 g_0}, t_e} = \sum_{o_e \in occ(t_e)} m^{\overline{f_0 g_0}, t_e}(o_e)$$

Mit Lemma 3.10 erhalten wir folgende Umformung.

$$= \sum_{o_e \in occ(t_e)} \left(\sum_{o_z \in bild^{f_0, t_e}(o_e)} m^{g_0, t_z}(o_z) \right)$$

Weil $M_2 \in lin\text{-}MAC$ ist, wird jedes Vorkommen in t_z höchstens einmal besucht. Da wegen der call-by-need-Strategie die Teile des Zwischenergebnisses, welche ein löschender tree transducer ignoriert, überhaupt nicht erzeugt werden, nehmen wir folgende Umformung mit Hilfe von Definition 3.11 vor.

$$= \sum_{o_e \in occ(t_e)} \left(\sum_{o_z \in bild^{f_0 \circ g_0, t_e}(o_e)} 1 \right) = \sum_{o_e \in occ(t_e)} card(bild^{f_0 \circ g_0, t_e}(o_e))$$

Da entsprechend der Definition 2.14 jeder Besuch eines Vorkommens im Eingabebaum ein Bildvorkommen erzeugt, und diese wegen $M_1 \in TOP$ und Lemma 2.17 für gleiche Eingabevorkommen alle verschieden sind, können wir die Anzahl der Bildvorkommen durch die Zahl der Besuche im Eingabebaum ersetzen.

$$= \sum_{o_e \in occ(t_e)} m^{f_0 \circ g_0, t_e}(o_e)$$

Durch Anwendung von Definition 3.11 erhalten wir schließlich

$$= r^{f_0 \circ g_0, t_e}$$

□

Satz 3.13 Für alle $(p, e) \in (TOP; lin\text{-}ndc\text{-}MAC)$ gilt: $\mathcal{E}(Com(p, e)) > \mathcal{E}(p, e)$.

Beweis: Das Kompositionsergebnis benötigt laut Lemma 3.12 $r^{f_0 \circ g_0, t_e}$ Schritte, die ursprünglichen tree transducer zusammen aber $r^{f_0 \circ g_0, t_e} + r^{g_0, t_z}$ Schritte. Wegen $r^{g_0, t_z} \geq 1$ gilt folglich $r^{\overline{f_0 g_0}, t_e} < r^{f_0 \circ g_0, t_e} + r^{g_0, t_z}$, womit der Satz bewiesen ist. □

Es werden also durch die Komposition (und damit auch durch Deforestation) so viele Reduktionsschritte eingespart, wie der macro tree transducer zur Verarbeitung des Zwischenergebnisses benötigt.

Beispiel 3.14 Gegeben sei folgendes Programm, welches aus zwei tree transducern M_1 und M_2 besteht, wobei $F_{M_1} = \{f_1^{(1)}, f_2^{(1)}\}$ und $F_{M_2} = \{g_1^{(2)}, g_2^{(2)}\}$.

$$\begin{array}{lcllcl}
f_1\ (\gamma\ x_1) & = & \sigma\ (f_2\ x_1)\ (f_2\ x_1) & g_1\ (\sigma\ x_1\ x_2)\ y_1 & = & \gamma\ (g_2\ x_1\ y_1) \\
f_2\ (\gamma\ x_1) & = & f_1\ x_1 & g_2\ (\sigma\ x_1\ x_2)\ y_1 & = & \gamma\ (g_2\ x_1\ (g_1\ x_2\ y_1)) \\
f_1\ \alpha & = & \alpha & g_1\ \alpha\ y_1 & = & \gamma\ y_1 \\
f_2\ \alpha & = & \alpha & g_2\ \alpha\ y_1 & = & \gamma\ y_1
\end{array}$$

Die (f_1, γ)-Regel ist nichtlinear im Rekursionsargument und die (f_2, γ)-Regel nonproducing, deshalb gilt $M_1 \in nlin\text{-}nprod\text{-}TOP$. Weiterhin ist $M_2 \in dr\text{-}lin\text{-}ndc\text{-}MAC$. M_1 erzeugt aus einem monadischen Eingabebaum, der n γ-Symbole enthält, einen aus σ-Symbolen bestehenden Binärbaum mit $n/2$ Verzweigungsniveaus, es wird also nur bei jedem zweiten Eingabesymbol γ etwas produziert. M_2 durchläuft einen solchen Binärbaum und erzeugt dabei γ-Symbole als Ausgabe, wobei nicht alle rechten Teilbäume der σ-Symbole berücksichtigt werden, weil die (g_1, σ)-Regel deleting im Rekursionsargument ist. Nun betrachten wir den Kompositionsausdruck $g_1\ (f_1\ z_0)\ z_1$. Daraus entsteht nach Transformation 2.22 der neue Ausdruck $\overline{f_1 g_1}\ z_0\ z_1$ und das folgende Programm, welches aus einem macro tree transducer M besteht.

$$\begin{array}{lcllcl}
\overline{f_1 g_1}\ (\gamma\ x_1)\ y_1 & = & \gamma\ (\overline{f_2 g_2}\ x_1\ y_1) & \overline{f_1 g_1}\ \alpha\ y_1 & = & \gamma\ y_1 \\
\overline{f_1 g_2}\ (\gamma\ x_1)\ y_1 & = & \gamma\ (\overline{f_2 g_2}\ x_1\ (\overline{f_2 g_1}\ x_1\ y_1)) & \overline{f_1 g_2}\ \alpha\ y_1 & = & \gamma\ y_1 \\
\overline{f_2 g_1}\ (\gamma\ x_1)\ y_1 & = & \overline{f_1 g_1}\ x_1\ y_1 & \overline{f_2 g_1}\ \alpha\ y_1 & = & \gamma\ y_1 \\
\overline{f_2 g_2}\ (\gamma\ x_1)\ y_1 & = & \overline{f_1 g_2}\ x_1\ y_1 & \overline{f_2 g_2}\ \alpha\ y_1 & = & \gamma\ y_1
\end{array}$$

Um den vorausgesagten Einsparungseffekt zu zeigen, belegen wir z_0 mit einem konkreten Baum $t_e \in T_{\{\gamma,\alpha\}}$, nämlich $\gamma^6\alpha$. Die Abbildung 3.2 zeigt den Eingabebaum t_e, das Zwischenergebnis t_z und das Endergebnis sowie die Zahl der für deren Erzeugung benötigten Reduktionsschritte. Außerdem sind die Reduktionsschritte eingetragen, die M zur direkten Transformation von t_e in das Endergebnis benötigt.

In dieser Abbildung sind im Zwischenergebnis t_z zwei Teilbäume t'_z und t''_z markiert, welche von M_1 durch Übersetzung der Teilbäume t'_e und t''_e des Eingabebaums erzeugt, von M_2 aber nicht berücksichtigt werden. Daher benötigt M_1 wegen der Auswertung mittels call-by-need nur $29 - 14 - 2 = 13$ Schritte. M benötigt ebenfalls 13 Reduktionsschritte. Es werden also tatsächlich die 7 Schritte des zweiten tree transducers eingespart, welche zur Verarbeitung des Zwischenergebnisses nötig sind. □

3.3.3 Die Kompositionsklasse ($prod\text{-}TOP; ndc\text{-}MAC$)

$M_1 = (F_{M_1}^{(1)}, C_{M_1}^i, C_{M_1}^o, R_{M_1}) \in prod\text{-}TOP$ und $M_2 = (F_{M_2}, C_{M_2}^i, C_{M_2}^o, R_{M_2}) \in ndc\text{-}MAC$ seien im folgenden zwei beliebige tree transducer mit den genannten Eigenschaften. Wir betrachten den Kompositionsausdruck $g_0\ (f_0\ z_0)\ z_1 \ldots z_{n_2}$ mit $f_0 \in F_{M_1}$, $g_0 \in F_{M_2}^{(1+n_2)}$ und $n_2 \geq 0$, aus dem nach Transformation 2.25 der neue macro tree transducer $M =$

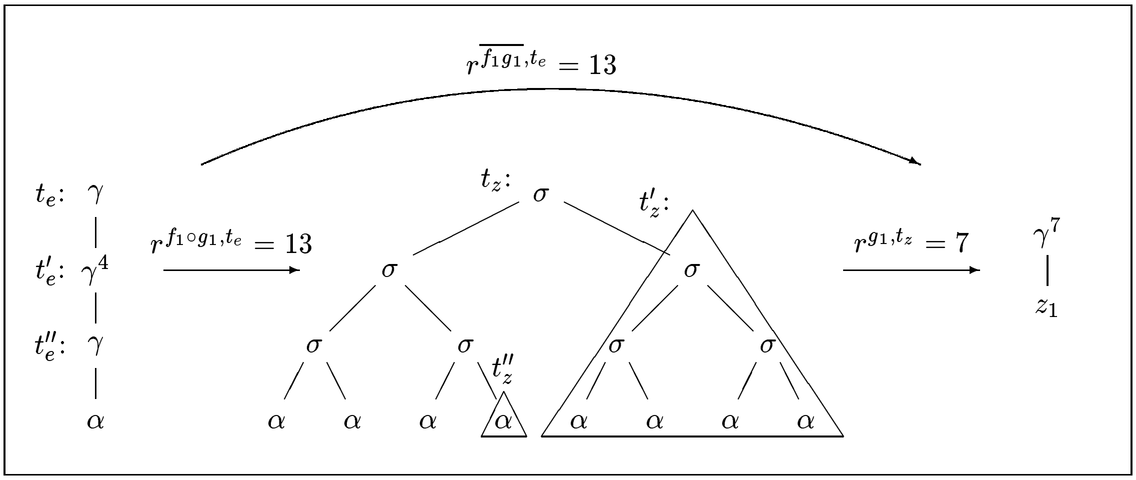

Abbildung 3.2: Reduktionsschritte für ein Beispiel aus $(nprod\text{-}TOP; lin\text{-}ndc\text{-}MAC)$

(F_M, C^i_M, C^o_M, R_M) und der neue Ausdruck $(\overline{f_0 g_0}\ z_0\ z_1 \ldots z_{n_2})$ mit $\overline{f_0 g_0}^{(1+n_2)} \in F_M$ entstehen. Weiterhin sei $t_e \in T_{C^i_{M_1}}$ und $t_z = nf(\Rightarrow_{M_1}, (f_0\ t_e))$.

Lemma 3.15 $\quad r^{\overline{f_0 g_0}, t_e} \leq r^{g_0, t_z}$

Beweis: Die Anzahl der Reduktionsschritte für das Kompositionsergebnis berechnen wir wieder mit Hilfe von Lemma 2.20

$$r^{\overline{f_0 g_0}, t_e} = \sum_{o_e \in occ(t_e)} m^{\overline{f_0 g_0}, t_e}(o_e)$$

und erhalten mit Lemma 3.10 folgende Umformung.

$$= \sum_{o_e \in occ(t_e)} \left(\sum_{o_z \in bild^{f_0, t_e}(o_e)} m^{g_0, t_z}(o_z) \right)$$

Wegen $M_1 \in prod\text{-}TOP$ und Lemma 2.18 ist die Vereinigung von Bildmengen verschiedener Vorkommen des Eingabebaums ohne den Verlust mehrfach vorkommender Elemente möglich.

$$= \sum_{o_z \in \bigcup\limits_{o_e \in occ(t_e)} bild^{f_0, t_e}(o_e)} m^{g_0, t_z}(o_z)$$

Die Vereinigung aller Bildmengen schöpft die Menge $occ(t_z)$ entweder komplett aus, oder es gibt in $occ(t_z)$ auch Vorkommen, die in überhaupt keiner Bildmenge enthalten sind. Deswegen nehmen wir folgende Abschätzung vor.

$$\leq \sum_{o_z \in occ(t_z)} m^{g_0, t_z}(o_z)$$

Durch nochmalige Anwendung von Lemma 2.20 erhalten wir schließlich

$$= r^{g_0, t_z}$$

□

Satz 3.16 Für alle $(p, e) \in (prod\text{-}TOP; ndc\text{-}MAC)$ gilt: $\mathcal{E}(Com(p, e)) > \mathcal{E}(p, e)$.

Beweis: Das Kompositionsergebnis benötigt laut Lemma 3.15 weniger als r^{g_0, t_z} Schritte, die ursprünglichen tree transducer zusammen aber $r^{f_0 \circ g_0, t_e} + r^{g_0, t_z}$ Schritte. Wegen $r^{f_0 \circ g_0, t_e} \geq 1$ gilt folglich $r^{\overline{f_0 g_0}, t_e} < r^{f_0 \circ g_0, t_e} + r^{g_0, t_z}$, womit der Satz bewiesen ist. □

Es werden also durch die Komposition (und damit auch durch Deforestation) so viele Reduktionsschritte eingespart, wie der top-down tree transducer zur Erzeugung des Zwischenergebnisses benötigt, zuzüglich der Schritte, die zur Reduktion der Symbole im Zwischenergebnis durch den macro tree transducer erforderlich sind, deren Pfad in keiner Bildmenge enthalten ist, weil sich in der erzeugenden rechten Seite darüber noch weitere Konstruktoren befanden.

Beispiel 3.17 Gegeben sei ein Programm, welches aus zwei auf monadischen Bäumen arbeitenden tree transducern M_{double} und M_{exp2} besteht, wobei $F_{M_{double}} = \{f^{(1)}\}$ und $F_{M_{exp2}} = \{g^{(2)}\}$.

$$\begin{array}{lcl} f\ (\gamma\ x_1) & = & \gamma\ (\gamma\ (f\ x_1)) \\ f\ \alpha & = & \alpha \\ & & \\ g\ (\gamma\ x_1)\ y_1 & = & \gamma(g\ x_1\ (g\ x_1\ y_1)) \\ g\ \alpha\ y_1 & = & \gamma\ y_1 \end{array}$$

Alle Regeln von M_{double} sind producing, deshalb gilt $M_{double} \in prod\text{-}TOP$. Weiterhin ist $M_{exp2} \in nlin\text{-}ndc\text{-}MAC$, weil die (g, γ)-Regel nichtlinear im Rekursionsargument ist und alle Regeln von M_{exp2} nondeleting in den Kontextparametern sind. M_{double} verdoppelt die Anzahl der γ-Symbole des monadischen Eingabebaumes. M_{exp2} erzeugt aus einem Baum, der n γ-Symbole enthält, einen Baum mit $2^{n+1} - 1$ γ-Symbolen, der Zuwachs ist also exponentiell. Nun betrachten wir den Kompositionsausdruck $g\ (f\ z_0)\ z_1$. Daraus entsteht nach Transformation 2.22 der neue Ausdruck $\overline{fg}\ z_0\ z_1$ und das folgende Programm, welches aus einem macro tree transducer M_{exp4} besteht.

$$\begin{array}{lcl} \overline{fg}\ (\gamma\ x_1)\ y_1 & = & \gamma\ (\gamma\ (\overline{fg}\ x_1\ (\overline{fg}\ x_1\ (\gamma\ (\overline{fg}\ x_1\ (\overline{fg}\ x_1\ y_1)))))) \\ \overline{fg}\ \alpha\ y_1 & = & \gamma\ y_1 \end{array}$$

Um den vorausgesagten Einsparungseffekt zu zeigen, belegen wir z_0 mit einem Baum $t_e \in T_{\{\gamma,\alpha\}}$, nämlich $\gamma^n\alpha$. Die Abbildung 3.3 zeigt den Eingabebaum t_e, das Zwischenergebnis $t_z = \gamma^{2n}\alpha$ und das Endergebnis $\gamma^{2 \cdot 4^n - 1}\alpha$ sowie die Zahl der für deren Erzeugung benötigten Reduktionsschritte. Außerdem sind die Reduktionsschritte eingetragen, die M_{exp4} zur direkten Transformation von t_e in das Endergebnis benötigt.

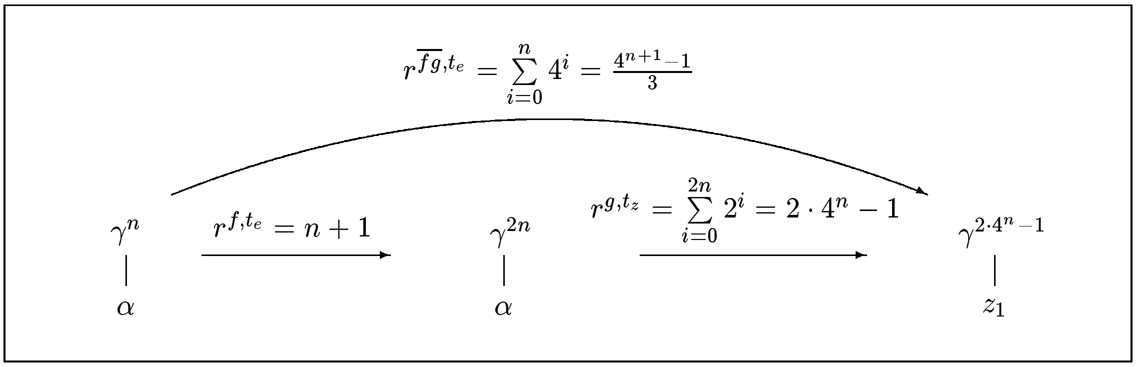

Abbildung 3.3: Reduktionsschritte für ein Beispiel aus $(prod\text{-}TOP; ndc\text{-}MAC)$

Von den Reduktionsschritten, die M_{exp2} zur Weiterverarbeitung des Zwischenergebnisses braucht, werden die für jedes zweite γ-Symbol benötigten Schritte eingespart, weil deren Pfade keine Bildvorkommen sind. Diese Einsparung berechnen wir wie folgt:

$$\sum_{i=1}^{n} 2^{2i-1} = \frac{1}{2}\sum_{i=1}^{n} 4^i = \frac{1}{2}\left(\frac{4^{n+1}-1}{3} - 1\right) = \frac{1}{2}\left(\frac{4^{n+1}-4}{3}\right) = 2\left(\frac{4^n-1}{3}\right) = \frac{2}{3}\left(4^n - 1\right)$$

Diese Zahl subtrahieren wir von den ursprünglichen $2 \cdot 4^n - 1$ Schritten:

$$(2 \cdot 4^n - 1) - \frac{2}{3}\left(4^n - 1\right) = \frac{6}{3}4^n - \frac{2}{3}4^n - 1 + \frac{2}{3} = \frac{4}{3}4^n - \frac{1}{3} = \frac{4^{n+1}-1}{3}$$

Da die $n+1$ Schritte, die M_{double} zur Erzeugung des Zwischenergebnisses benötigt, ebenfalls eingespart werden, haben wir damit die Reduktionsschritte des bei der Komposition entstandenen macro tree transducers berechnet. Ein Vergleich mit dem Wert aus Abbildung 3.3 bestätigt dies. □

3.3.4 Die Kompositionsklasse $(nprod\text{-}TOP; nlin\text{-}ndc\text{-}MAC)$

Beobachtung 3.18 Bei der Komposition von top-down tree transducern, welche nonproducing sind, mit nichtlinearen macro tree transducern können Effizienzverluste auftreten. □

Ein Beispiel für diesen Fall haben wir schon in der Einleitung gesehen. Dabei trat die Nichtlinearität im Rekursionsargument auf. Das folgende Beispiel verdeutlicht, daß Nichtlinearität auch in den Kontextparametern problematisch sein kann.

Beispiel 3.19 Gegeben sei ein Programm, welches zwei tree transducer M_1 und M_2 enthält, wobei $F_{M_1} = \{f^{(1)}\}$ und $F_{M_2} = \{g^{(2)}\}$:

$$\begin{array}{lcl}
f\ (\sigma\ x_1\ x_2) & = & \sigma\ (\delta\ (f\ x_1))\ (\delta\ (f\ x_2)) \\
f\ (\gamma\ x_1) & = & \boxed{f\ x_1} \\
f\ \alpha & = & \alpha \\
\\
g\ (\sigma\ x_1\ x_2)\ y_1 & = & g\ x_1\ (g\ x_2\ y_1) \\
g\ (\delta\ x_1)\ y_1 & = & g\ x_1\ (\sigma\ \underline{y_1}\ \underline{y_1}) \\
g\ \alpha\ y_1 & = & y_1
\end{array}$$

Die (f, γ)-Regel ist nonproducing, da die eingerahmte rechte Seite keinen Konstruktor enthält, also gilt $M_1 \in nprod\text{-}TOP$. Weiterhin ist $M_2 \in nlin\text{-}ndc\text{-}MAC$, weil alle Regeln nondeleting im Kontextparameter sind und die (g, δ)-Regel nichtlinear im Kontextparameter ist, was die Unterstreichungen der doppelt vorkommenden Kontextvariable zeigen. Für den Kompositionsausdruck $g\ (f\ z_0)\ z_1$ erzeugt Transformation 2.22 den neuen Ausdruck $\overline{fg}\ z_0\ z_1$ und folgendes neue Programm, welches einen macro tree transducer M mit $F_M = \{\overline{fg}^{(2)}\}$ enthält:

$$\begin{array}{lcl}
\overline{fg}\ (\sigma\ x_1\ x_2)\ y_1 & = & \overline{fg}\ x_1\ (\sigma\ (\overline{fg}\ x_2\ (\sigma\ y_1\ y_1))\ (\overline{fg}\ x_2\ (\sigma\ y_1\ y_1))) \\
\overline{fg}\ (\gamma\ x_1)\ y_1 & = & \overline{fg}\ x_1\ y_1 \\
\overline{fg}\ \alpha\ y_1 & = & y_1
\end{array}$$

Nun belegen wir z_0 mit einem konkreten Baum t_e. Die Abbildung 3.4 zeigt diesen Eingabebaum, das Zwischenergebnis t_z und das Endergebnis sowie die Zahl der für deren Erzeugung benötigten Reduktionsschritte. Außerdem sind die Reduktionsschritte eingetragen, die M zur direkten Transformation von t_e in das Endergebnis benötigt.

Man erkennt, daß ab $n = 3$ für die direkte Erzeugung des Endergebnisses mehr Reduktionsschritte benötigt werden, als für die Generierung und Verarbeitung des Zwischenergebnisses erforderlich sind. □

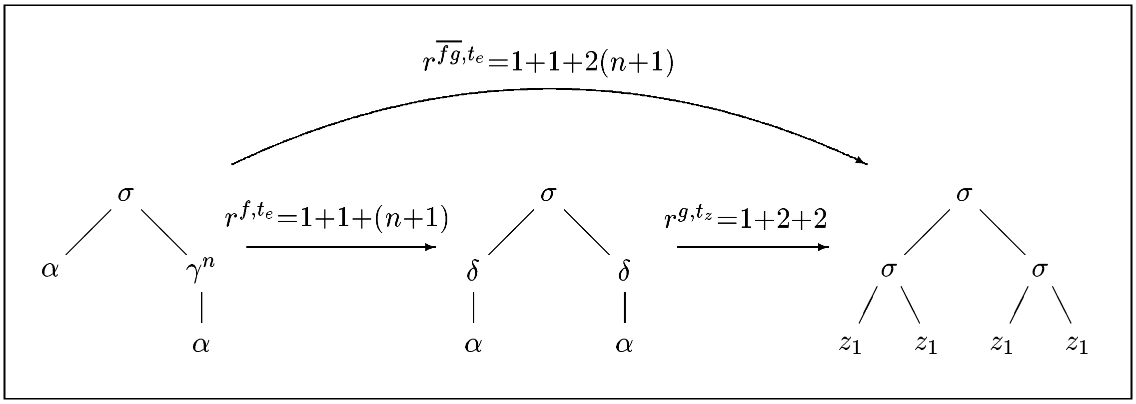

Abbildung 3.4: Reduktionsschritte für ein Beispiel mit Nichtlinearität im Kontext

3.4 Die Kompositionsklasse $(TOP; dc\text{-}MAC)$

In diesem Abschnitt lassen sich die Beweise des vorigen Abschnitts nicht durchführen, weil Lemma 3.7 nicht anwendbar ist. Wir werden anhand von Beispielen jedoch eine Übereinstimmung der Einspareffekte feststellen und die Vermutung äußern, daß die Resultate des vorigen Abschnitts auch für die hier behandelte Klasse gültig sind. Deshalb nehmen wir die gleiche Unterteilung der Klasse vor.

3.4.1 Die Kompositionsklasse $(TOP; lin\text{-}dc\text{-}MAC)$

Beispiel 3.20 Wir verwenden das Programm aus Beispiel 3.14, welches aus zwei tree transducern M_1 und M_2 besteht, wobei $F_{M_1} = \{f_1^{(1)}, f_2^{(1)}\}$ und $F_{M_2} = \{g_1^{(2)}, g_2^{(2)}\}$. Dieses erweitern wir um Regeln für ein zweites nullstelliges Symbol α_0, wobei die (g_1, α_0)-Regel und die (g_2, α_0)-Regel deleting im Kontextparameter sind.

$$
\begin{array}{lcllcl}
f_1\ (\gamma\ x_1) & = & \sigma\ (f_2\ x_1)\ (f_2\ x_1) & g_1\ (\sigma\ x_1\ x_2)\ y_1 & = & \gamma\ (g_2\ x_1\ y_1) \\
f_2\ (\gamma\ x_1) & = & f_1\ x_1 & g_2\ (\sigma\ x_1\ x_2)\ y_1 & = & \gamma\ (g_2\ x_1\ (g_1\ x_2\ y_1)) \\
f_1\ \alpha & = & \alpha & g_1\ \alpha\ y_1 & = & \gamma\ y_1 \\
f_2\ \alpha & = & \alpha & g_2\ \alpha\ y_1 & = & \gamma\ y_1 \\
f_1\ \alpha_0 & = & \alpha_0 & g_1\ \alpha_0\ y_1 & = & \alpha_0 \\
f_2\ \alpha_0 & = & \alpha_0 & g_2\ \alpha_0\ y_1 & = & \alpha_0
\end{array}
$$

Die (f_1, γ)-Regel ist nichtlinear im Rekursionsargument und die (f_2, γ)-Regel nonproducing, deshalb gilt $M_1 \in nlin\text{-}nprod\text{-}TOP$. Weiterhin ist $M_2 \in dr\text{-}lin\text{-}dc\text{-}MAC$. M_1 erzeugt wiederum aus einem monadischen Eingabebaum, der n γ-Symbole enthält, einen aus σ-Symbolen bestehenden Binärbaum mit $n/2$ Verzweigungsniveaus, es wird also nur bei jedem zweiten Eingabesymbol γ etwas produziert. M_2 durchläuft einen solchen Binärbaum und erzeugt dabei γ-Symbole als Ausgabe, wobei nicht alle rechten Teilbäume der σ-Symbole berücksichtigt werden, weil die (g_1, σ)-Regel deleting im Rekursionsargument ist. In diesem Beispiel können noch mehr solcher Bäume ignoriert werden, weil die (g_2, α_0)-Regel deleting im Rekursionsargument ist. Nun betrachten wir den Kompositionsausdruck $g_1\ (f_1\ z_0)\ z_1$. Daraus entsteht nach Transformation 2.22 der neue Ausdruck $\overline{f_1 g_1}\ z_0\ z_1$ und das folgende Programm, welches aus einem macro tree transducer M besteht.

$$
\begin{array}{lcl}
\overline{f_1g_1}\,(\gamma\, x_1)\, y_1 & = & \gamma\,(\overline{f_2g_2}\, x_1\, y_1) \\
\overline{f_1g_2}\,(\gamma\, x_1)\, y_1 & = & \gamma\,(\overline{f_2g_2}\, x_1\,(\overline{f_2g_1}\, x_1\, y_1)) \\
\overline{f_2g_1}\,(\gamma\, x_1)\, y_1 & = & \overline{f_1g_1}\, x_1\, y_1 \\
\overline{f_2g_2}\,(\gamma\, x_1)\, y_1 & = & \overline{f_1g_2}\, x_1\, y_1
\end{array}
$$

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
\overline{f_1g_1}\,\alpha\, y_1 & = & \gamma\, y_1 & \overline{f_1g_1}\,\alpha_0\, y_1 & = & \alpha_0 \\
\overline{f_1g_2}\,\alpha\, y_1 & = & \gamma\, y_1 & \overline{f_1g_2}\,\alpha_0\, y_1 & = & \alpha_0 \\
\overline{f_2g_1}\,\alpha\, y_1 & = & \gamma\, y_1 & \overline{f_2g_1}\,\alpha_0\, y_1 & = & \alpha_0 \\
\overline{f_2g_2}\,\alpha\, y_1 & = & \gamma\, y_1 & \overline{f_2g_2}\,\alpha_0\, y_1 & = & \alpha_0
\end{array}
$$

Um den Einsparungseffekt zu untersuchen, belegen wir z_0 mit dem konkreten Baum $t_e = \gamma^6\alpha_0$. Die Abbildung 3.5 zeigt diesen Eingabebaum t_e, das Zwischenergebnis t_z und das Endergebnis sowie die Zahl der für deren Erzeugung benötigten Reduktionsschritte. Außerdem sind die Reduktionsschritte eingetragen, die M zur direkten Transformation von t_e in das Endergebnis benötigt.

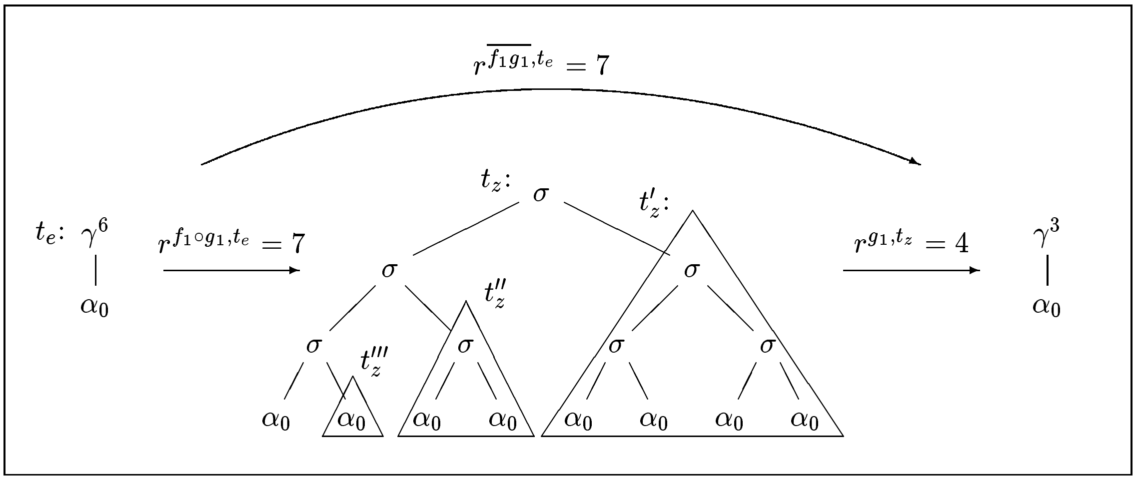

Abbildung 3.5: Reduktionsschritte für ein Beispiel aus $(nprod\text{-}TOP; lin\text{-}dc\text{-}MAC)$

In dieser Abbildung sind im Zwischenergebnis alle rechten Teilbäume der auf dem linkesten Ast stehenden σ-Symbole markiert, welche von M_2 nicht berücksichtigt werden. M_1 braucht deshalb zur Erzeugung des übrigen Teils wegen der Auswertung mittels call-by-need nur noch 7 Reduktionsschritte. Weil wegen des α_0-Symbols am Ende des Eingabebaums die Kontextparameter von M nicht verwendet werden, läuft er linear durch den Eingabebaum und braucht ebenfalls 7 Schritte. Damit werden die 4 Schritte, die M_2 zur Verarbeitung des Zwischenergebnis benötigte, eingespart. □

Diese Einsparung gleicht der, die im Abschnitt 3.3.2 für die Klasse $(TOP; lin\text{-}ndc\text{-}MAC)$ berechnet wurde. Wir vermuten deshalb, daß das dort angegebene Resultat auch für die hier behandelte Klasse gilt.

Vermutung 3.21 Für alle $(p, e) \in (TOP; lin\text{-}dc\text{-}MAC)$ gilt: $\mathcal{E}(Com(p, e)) > \mathcal{E}(p, e)$. □

3.4.2 Die Kompositionsklasse $(prod\text{-}TOP; dc\text{-}MAC)$

Beispiel 3.22 Wir verwenden das Programm aus Beispiel 3.17, welches aus zwei auf monadischen Bäumen arbeitenden tree transducern M_{double} und M_{exp2} bestand. Dieses erweitern wir um Regeln für ein zweites nullstelliges Symbol α_0, wobei die (g, α_0)-Regel deleting im Kontextparameter ist. Damit erhalten wir die tree transducer $M_{double'}$ und $M_{id/exp2}$, wobei $F_{M_{double'}} = \{f^{(1)}\}$ und $F_{M_{id/exp2}} = \{g^{(2)}\}$

$$\begin{aligned}
f\ (\gamma\ x_1) &= \gamma\ (\gamma\ (f\ x_1)) \\
f\ \alpha &= \alpha \\
f\ \alpha_0 &= \alpha_0 \\[4pt]
g\ (\gamma\ x_1)\ y_1 &= \gamma(g\ x_1\ (g\ x_1\ y_1)) \\
g\ \alpha\ y_1 &= \gamma\ y_1 \\
g\ \alpha_0\ y_1 &= \alpha_0
\end{aligned}$$

Alle Regeln von $M_{double'}$ sind producing, deshalb gilt $M_{double'} \in$ *prod-TOP*. Weiterhin ist $M_{id/exp2} \in$ *nlin-dc-MAC*, weil die (g, α_0)-Regel deleting im Kontextparameter und die (g, γ)-Regel nichtlinear im Rekursionsargument ist. Das Verhalten von $M_{id/exp2}$ weicht von dem aus Beispiel 3.17 ab. Für Bäume, die mit α_0 abschließen, berechnet $M_{id/exp2}$ die Identität, anstatt wie bei den mit α endenden Bäumen die Zahl der γ-Symbole exponentiell zu vervielfachen. Nun betrachten wir den Kompositionsausdruck $g\ (f\ z_0)\ z_1$. Daraus entsteht nach Transformation 2.22 der neue Ausdruck $\overline{fg}\ z_0\ z_1$ und das folgende Programm, welches aus einem macro tree transducer $M_{double/exp4}$ besteht.

$$\begin{aligned}
\overline{fg}\ (\gamma\ x_1)\ y_1 &= \gamma\ (\gamma\ (\overline{fg}\ x_1\ (\overline{fg}\ x_1\ (\gamma\ (\overline{fg}\ x_1\ (\overline{fg}\ x_1\ y_1)))))) \\
\overline{fg}\ \alpha\ y_1 &= \gamma\ y_1 \\
\overline{fg}\ \alpha_0\ y_1 &= \alpha_0
\end{aligned}$$

Nun belegen wir z_0 mit dem Baum $t_e = \gamma^n \alpha_0$. Die Abbildung 3.6 zeigt den Eingabebaum t_e, das Zwischenergebnis $t_z = \gamma^{2n}\alpha_0$ und das Endergebnis $\gamma^{2n}\alpha_0$ sowie die Zahl der für deren Erzeugung benötigten Reduktionsschritte. Außerdem sind die Reduktionsschritte eingetragen, die M zur direkten Transformation von t_e in das Endergebnis benötigt.

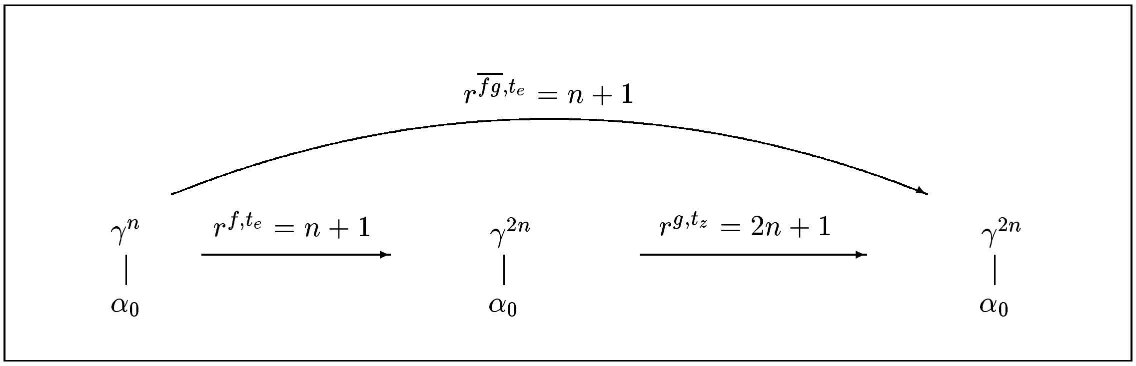

Abbildung 3.6: Reduktionsschritte für ein Beispiel aus (*prod-TOP*; *dc-MAC*)

Der bei der Komposition entstandene macro tree transducer benötigt nur noch $n+1$ Schritte. Dabei werden von den $2n+1$ Reduktionsschritten, die der ursprüngliche macro tree transducer zur Verarbeitung des Zwischenergebnisses braucht, die für jedes zweite γ-Symbol benötigten Schritte eingespart. Diese Einsparung beträgt n Schritte. Außerdem werden die $n+1$ Schritte, die der top-down tree transducer zur Erzeugung des Zwischenergebnisses benötigt, eingespart. □

Diese Einsparungen gleichen denen, die im Abschnitt 3.3.3 für die Klasse (*prod-TOP*; *ndc-MAC*) berechnet wurden. Wir vermuten deshalb, daß das dort angegebene Resultat auch für die hier behandelte Klasse gilt.

Vermutung 3.23 Für alle $(p, e) \in$ (*prod-TOP*; *dc-MAC*) gilt: $\mathcal{E}(Com(p, e)) > \mathcal{E}(p, e)$. □

3.4.3 Die Kompositionsklasse $(nprod\text{-}TOP; nlin\text{-}dc\text{-}MAC)$

Auch hier können wieder Effizienzverluste auftreten. Beispiele für diese Klasse erhält man z.B., indem man den linken Seiten der Regeln eines Beispiels aus der Klasse $(nprod\text{-}TOP; nlin\text{-}ndc\text{-}MAC)$ weitere Kontextparameter hinzufügt, ohne diese in den rechten Seiten zu verwenden. Damit ändert sich der Reduktionsvorgang nicht, so daß die dort möglichen Effizienzverluste weiterhin auftreten. Daher wurde in Beobachtung 3.18 die nondeleting-Eigenschaft in den Kontextparametern nicht explizit gefordert.

3.5 Zusammenfassung

Abschließend fassen wir die Resultate aus diesem Kapitel in Form einer Tabelle zusammen. Das Zeichen „:)“ bedeutet dabei, daß durch Deforestation und Komposition eine Effizienzsteigerung erreicht wird. Mit „:(“ verdeutlichen wir den möglichen Effizienzverlust.

$(y\text{-}TOP; x\text{-}MAC)$	*TOP* bzw. *ndc-MAC*		*dc-MAC*	
	lin	*nlin*	*lin*	*nlin*
prod-TOP	:)	$\leftrightarrow$ 3.16 :)	:)	$\leftrightarrow$ 3.23 :)
	$\updownarrow$ 3.13	3.18	$\updownarrow$ 3.21	3.18
nprod-TOP	:)	:(	:)	:(
	bewiesen		vermutet	

Abbildung 3.7: Zusammenfassung der Resultate für die Kompositionsklasse $(TOP; MAC)$

Kapitel 4

Vergleiche für die Kompositionsklasse $(MAC; TOP)$

4.1 Ein einführendes Beispiel

Wir wollen das Kapitel mit einem umfangreicheren Beispiel eröffnen, anhand dessen wir einige Effekte zeigen können, die für Deforestation und Komposition bei der hier behandelten Kompositionsklasse typisch sind.

Beispiel 4.1 Gegeben sei ein Programm, welches aus den beiden tree transducern M_1 und M_2 besteht, wobei $F_{M_1} = \{f^{(2)}\}$ und $F_{M_2} = \{g^{(1)}\}$. σ sei dabei ein n-stelliges Symbol mit $n \geq 1$.

$$\begin{array}{lcl}
f\ (\delta\ x_1)\ y_1 & = & \delta\ (f\ x_1\ (f\ x_1\ y_1)) \\
f\ (\gamma_1\ x_1)\ y_1 & = & \gamma_1\ (f\ x_1\ y_1) \\
f\ (\gamma_2\ x_1)\ y_1 & = & f\ x_1\ (\gamma_2\ y_1) \\
f\ (\gamma_0\ x_1)\ y_1 & = & f\ x_1\ y_1 \\
f\ \alpha\ y_1 & = & \delta\ y_1 \\
g\ (\delta\ x_1) & = & \sigma\ \overbrace{(g\ x_1) \ldots (g\ x_1)}^{n\text{-mal}} \\
g\ (\gamma_1\ x_1) & = & g\ x_1 \\
g\ (\gamma_2\ x_1) & = & g\ x_1 \\
g\ \alpha & = & \alpha
\end{array}$$

M_1 ist ein macro tree transducer. Die (f, δ)-Regel ist nichtlinear im Rekursionsargument und enthält einen verschachtelten Funktionsaufruf. Die (f, γ_1)-Regel und die (f, γ_2)-Regel sind producing, wobei sich bei der erstgenannten Regel das Ausgabesymbol außerhalb, bei der letztgenannten aber im Kontextparameter des Funktionsaufrufs befindet. Die (f, γ_0)-Regel ist nonproducing. Außerdem sind alle Regeln nondeleting im Kontextparameter.

M_2 ist ein top-down tree transducer. Die Eigenschaften der (f, σ)-Regel hängen vom Parameter n ab. Für $n = 1$ ist diese linear, ansonsten nichtlinear im Rekursionsargument. Weiterhin sind alle Regeln von M_2 nondeleting im Rekursionsargument.

Für den Kompositionsausdruck $g\ (f\ z_0\ z_1)$ erzeugt die Komposition der beiden tree transducer nach Transformation 2.25 den neuen Ausdruck $\overline{fg}\ z_0\ (g\ z_1)$ und das folgende neue Programm, welches einen macro tree transducer M mit $F_M = \{\overline{fg}^{(2)}\}$ enthält.

$$\begin{array}{lcl}
\overline{fg}\ (\delta\ x_1)\ y_1 & = & \sigma\ \overbrace{(\overline{fg}\ x_1\ (\overline{fg}\ x_1\ y_1))\dots(\overline{fg}\ x_1\ (\overline{fg}\ x_1\ y_1))}^{n\text{-mal}} \\
\overline{fg}\ (\gamma_1\ x_1)\ y_1 & = & \overline{fg}\ x_1\ y_1 \\
\overline{fg}\ (\gamma_2\ x_1)\ y_1 & = & \overline{fg}\ x_1\ y_1 \\
\overline{fg}\ (\gamma_0\ x_1)\ y_1 & = & \overline{fg}\ x_1\ y_1 \\
\overline{fg}\ \alpha\ y_1 & = & \sigma\ \underbrace{y_1\dots y_1}_{n\text{-mal}}
\end{array}$$

Deforestation erzeugt nach Transformation 2.28 den neuen Ausdruck $fg\ z_0\ z_1$ und das folgende neue Programm, welches in $P^+ - P$ ist, weil die rechte Seite der (fg,α)-Regel nicht dem Aufbau der Regeln von macro tree transducern genügt.

$$\begin{array}{lcl}
fg\ (\delta\ x_1)\ y_1 & = & \sigma\ \overbrace{(fg\ x_1\ (f\ x_1\ y_1))\dots(fg\ x_1\ (f\ x_1\ y_1))}^{n\text{-mal}} \\
fg\ (\gamma_1\ x_1)\ y_1 & = & fg\ x_1\ y_1 \\
fg\ (\gamma_2\ x_1)\ y_1 & = & fg\ x_1\ (\gamma_2\ y_1) \\
fg\ (\gamma_0\ x_1)\ y_1 & = & fg\ x_1\ y_1 \\
fg\ (\alpha\ y_1) & = & \sigma\ \underbrace{(g\ y_1)\dots(g\ y_1)}_{n\text{-mal}}
\end{array}$$

Nun belegen wir z_0 mit einem Eingabebaum, dessen Aussehen durch Parameter n_1, n_2 und $n_0 \in I\!N$ gesteuert wird. Damit können wir die verschiedenen Fälle bezüglich der producing-Eigenschaft des macro tree transducers in Kombination mit der Linearität/ Nichtlinearität des top-down tree transducers untersuchen. Die Abbildung 4.1 zeigt diesen Eingabebaum, das Zwischenergebnis und das Endergebnis sowie die zu deren Erzeugung notwendigen Reduktionsschritte. z_1 sei dabei mit einem beliebigen Baum $t \in T_{C^o_{M_1}}$ belegt.

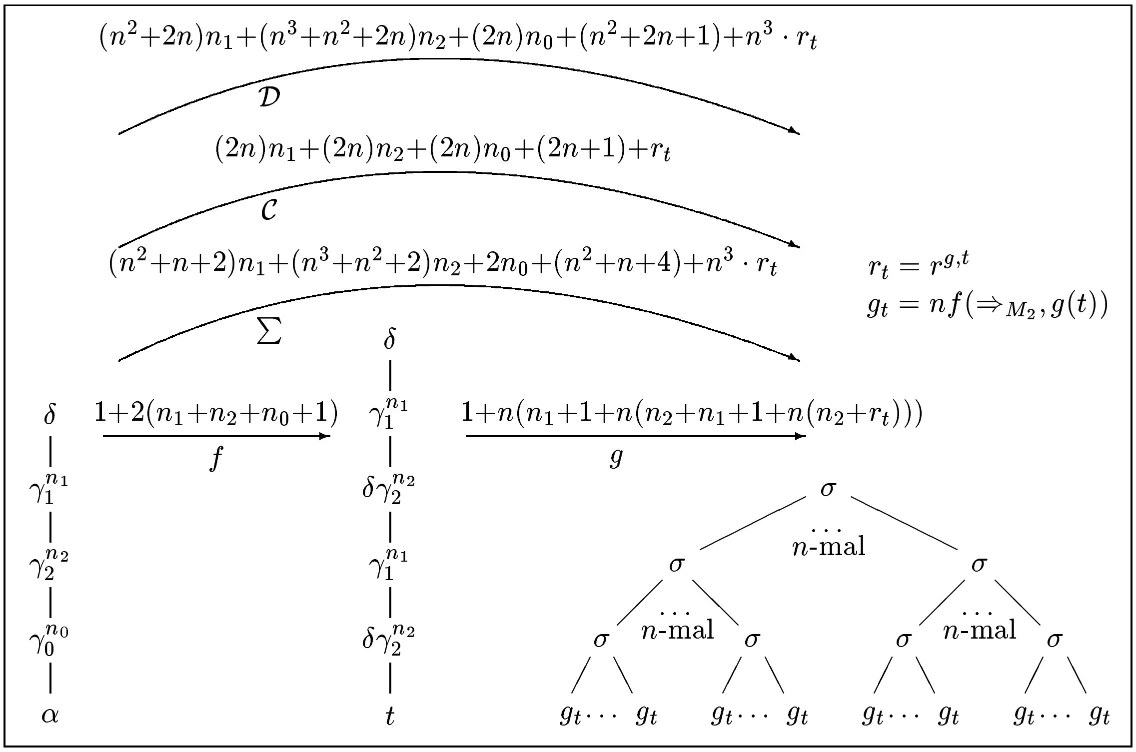

Abbildung 4.1: Bäume und Reduktionsschritte für ein Beispiel aus $(MAC;TOP)$

M_1 benötigt für jedes γ-Symbol zwei Reduktionsschritte, weil der verschachtelte Funktionsaufruf in der (f,δ)-Regel einen zweiten Durchlauf des Eingabebaumes verursacht. Nun betrachten wir die verschiedenen auftretenden Fälle.

- $n = 1$ ($M_2 \in$ *lin-TOP*)

  M_2 benötigt für jedes Symbol γ_1 und γ_2 des Eingabebaums zwei Reduktionsschritte, weil das Zwischenergebnis genau einmal durchlaufen wird und M_1 für ein Symbol des Eingabebaums zwei Symbole im Zwischenergebnis erzeugt. Damit brauchen M_1 und M_2 für γ_1 und γ_2 insgesamt jeweils 4 Schritte und für jedes γ_0 zwei Schritte.

  Durch Deforestation wird für jedes γ_1 ein Reduktionsschritt eingespart, weil die beim vom äußeren Funktionsaufruf der (f, δ)-Regel verursachten ersten Lauf von M_1 durch den Eingabebaum erzeugten Zwischenergebnisse vermieden werden. Der zweite Durchlauf wird jedoch unverändert durchgeführt, weil Deforestation den inneren Funktionsaufruf nicht erreicht. Für γ_2 wird sogar überhaupt nichts gespart, weil das im Kontextparamter der (f, γ_2)-Regel erzeugte Zwischenergebnis nicht entfernt wird. Damit sind für jedes γ_1-Symbol 3 Schritte und für jedes γ_2-Symbol 4 Schritte nötig. Bei γ_0 kann nichts eingespart werden, weil ja kein Zwischenergebnis erzeugt wird.

  Die Komposition entfernt sämtliche Zwischenergebnisse, so daß für γ_1 und γ_2 nur ein Schritt pro Durchlauf und Symbol benötigt wird, wegen der zwei Durchläufe also insgesamt je 2 Schritte. Für γ_0 kann wie bei Deforestation nichts eingespart werden.

- $n > 1$ ($M_2 \in$ *nlin-TOP*)

  - $n_0 = 0$ (M_1 wendet nur Regeln an, die producing sind.)

    Die (f, γ_0)-Regel ist dann überflüssig und kann weggelassen werden, wodurch M_1 producing wird. M_2 durchläuft bei jedem Besuch eines mit δ beschrifteten Vorkommens im Zwischenergebnis den Nachfolgerbaum n-mal. Damit sind für jedes Symbol γ_1 und γ_2 des Eingabebaums bei der Verarbeitung des Zwischenergebnisses $n + n^2$ bzw. $n^2 + n^3$ Reduktionsschritte nötig. Der Baum t muß n^3-mal reduziert werden.

    Nach Anwendung von Deforestation werden für jedes γ_1-Symbol $(n^2 + n + 2) - 1 + (-1 + n)$ Reduktionsschritte benötigt. Dabei wird auf der einen Seite zwar die Erzeugung des Zwischenergebnisses durch den äußeren Funktionsaufruf in der (f, δ)-Regel vermieden, wodurch pro Symbol ein Schritt eingespart wird. Andererseits wird der durch den inneren Aufruf verursachte Durchlauf vervielfacht, so daß statt einem nun n Schritte pro Symbol erforderlich sind. Dieser Effekt ist stärker als die Einsparung, so daß ab $n = 3$ ein Effizienzverlust auftritt. Für γ_2 werden $(n^3 + n^2 + 2n)$ statt der ursprünglichen $(n^3 + n^2 + 2)$ Reduktionsschritte benötigt. Dabei wird überhaupt kein Durchlauf gespart, weil das im Kontextparamter der (f, γ_2)-Regel erzeugte Zwischenergebnis nicht entfernt wird. Da die Vervielfachung auch hier auftritt, ist die Effizienz in jedem Fall geringer.

    Die Komposition entfernt sämtliche Zwischenergebnisse, so daß pro Nachfolger von σ nur zwei Durchläufe des Eingabebaums (je einen für den äußeren und inneren Funktionsaufruf) benötigt werden, also für γ_1 und γ_2 jeweils $2n$ Schritte. Außerdem muß t nur einmal übersetzt werden, weil dies bereits im Berechnungsausdruck geschieht. Wegen der call-by-need-Auswertungsstrategie wird bei mehrfacher Benutzung das Ergebnis der ersten Auswertung verwendet.

  - $n_0 > 0$ (M_1 wendet auch Regeln an, die nonproducing sind.)

    Hierbei kommt es zu einer weiteren Möglichkeit des Effizienzverlustes, weil die ursprünglichen tree transducer nur 2 Reduktionsschritte, die Ergebnisse von Deforestation und Komposition aber $2n$ Schritte pro γ_0-Symbol brauchen.

Wir erkennen also, daß es mehrere Möglichkeiten des Effizienzverlustes gibt, wenn der top-down tree transducer nichtlinear ist, wobei das Deforestation-Verfahren dabei erheblich anfälliger ist. □

Im weiteren Verlauf des Kapitels werden wir zeigen, daß die in diesem Beispiel aufgetretenen Phänomene allgemeingültigen Charakter haben.

4.2 Notationen

Da wir im Beispiel 4.1 gesehen haben, daß der Ort des Auftretens von Konstruktoren und Funktionsaufrufen bezüglich der Kontextparameter bei Deforestation und der Komposition zu unterschiedlichen Verhaltensweisen führen kann, unterteilen wir die rechte Seite einer Regel in drei Zonen.

Definition 4.2 Sei $r \in RHS(F, C, k, n)$. Wir unterteilen $occ(r)$ wie folgt:

- *Die Menge der Vorkommen im oberen (oder äußeren) Teil von* r, bezeichnet mit $occ_{top}(r)$, definieren wir als $occ_{top}(r) = \{o \mid o \in occ(r), label(o, r) \in C$, und für alle Teilpfade o_1 von o mit $o = o_1.o_2$ gilt $o_1 \in occ_{top}(r)\}$.
- *Die Menge der Vorkommen im mittleren Teil von* r, bezeichnet mit $occ_{mid}(r)$, definieren wir als $occ_{mid}(r) = \{o \mid o \in occ(r), label(o, r) \in F \cup Y_n$, und für alle Teilpfade o_1 von o mit $o = o_1.o_2$ gilt $o_1 \in occ_{top}(r)\} \cup \{o' \mid o' \in occ(r),\ o' = o.1,\ o \in occ_{mid}(r)\}$.
- *Die Menge der Vorkommen im unteren (oder inneren) Teil von* r, bezeichnet mit $occ_{bot}(r)$, definieren wir als $occ_{bot}(r) = occ(r) - occ_{top}(r) - occ_{mid}(r)$. □

Beispiel 4.3 Wir betrachten die rechte Seite $r = (\sigma(\sigma\ \alpha\ (f\ x_1\ \alpha\ (f\ x_1\ \alpha\ \alpha\ \alpha)\ y_2))\ y_3) \in RHS(\{f^{(4)}\}, \{\sigma^{(2)}, \alpha^{(0)}\}, 1, 3)$. Die Aufteilung dieser rechten Seite in die drei Zonen verdeutlicht Abbildung 4.2.

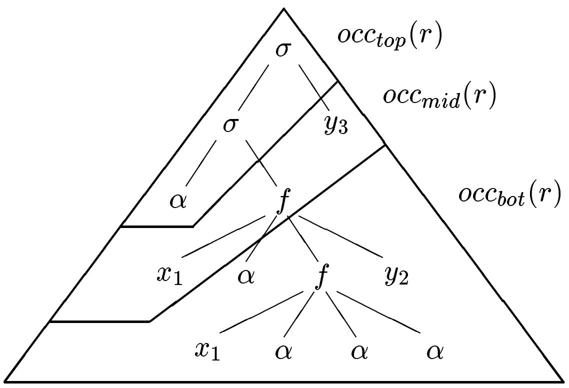

Abbildung 4.2: Zonenaufteilung einer rechten Seite

Der obere Teil von r besteht also aus miteinander verbundenen Kontruktoren, der Mittelteil aus sich direkt daran anschließenden Kontextvariablen und Funktionsaufrufen nebst deren Rekursionsargumenten, während der untere Teil den Rest von r umfaßt, wobei dies sowohl Konstruktoren, Kontextvariablen als auch weitere Funktionsaufrufe mit deren Rekursionsargumenten und Kontextparametern sein können. □

Entsprechend dieser Zonenaufteilung definieren wir nun die producing-Eigenschaft etwas präziser und führen eine Bezeichnung für das Nichtvorhandensein von verschachtelten Funktionsaufrufen (s. [Vog87]) ein.

Definition 4.4 Sei $M = (F, C^i, C^o, R)$ ein macro tree transducer. Weiterhin sei $f \in F^{(1+n)}$ mit $n \geq 0$ und $c \in C^{i(k)}$ mit $k \geq 0$. Sei $r = rhs(f, c)$. Die (f, c)-Regel heißt

- *außen producing*, wenn $occ_{top}(r) \neq \emptyset$.
- *innen producing*, wenn ein Vorkommen $o \in occ_{bot}(r)$ mit $label(o, r) = c'$ und $c' \in C^o$ existiert.
- *basic*, wenn kein $o \in occ_{bot}(r)$ mit $label(o, r) = f'$ und $f' \in F$ existiert. □

Eine Regel ist im Sinne von Definition 2.4 producing, wenn sie innen oder außen producing ist. Man beachte, daß für top-down tree transducer die producing-Eigenschaft und die Eigenschaft außen producing zusammenfallen.

In der folgenden Definition erweitern wir die Notation aus Definition 3.6, um die Zahlen des Auftretens von Funktionsaufrufen in den verschiedenen Regelzonen unterscheiden zu konnen.

Definition 4.5 Sei $i \in [k]$. Für die *Häufigkeit des Auftretens von Aufrufen der Funktion f mit dem Rekursionsargument x_i im mittleren* bzw. *inneren Teil von $rhs(f', c)$* schreiben wir $n_{mid}(f, i, f', c)$ bzw. $n_{bot}(f, i, f', c)$ und definieren sie als

$$\begin{aligned} n_{mid}(f, i, f', c) &= card\{o \mid o \in occ_{mid}(rhs(f', c)), \\ &\qquad label(o, rhs(f', c)) = f,\ label(o.1, rhs(f', c)) = x_i\} \text{ bzw.} \\ n_{bot}(f, i, f', c) &= card\{o \mid o \in occ_{bot}(rhs(f', c)), \\ &\qquad label(o, rhs(f', c)) = f,\ label(o.1, rhs(f', c)) = x_i\}. \end{aligned}$$ □

4.3 Die Kompositionsklasse (ndc-MAC; $1f$-ndr-TOP)

Wie schon im vorigen Kapitel betrachtet, erleichtert die nondeleting-Eigenschaft in den Kontextparametern bei macro tree transducern die Berechnung der Reduktionsschrittanzahl erheblich, weil diese dann nur vom Aussehen der verwendeten Regeln abhängt. Dies gilt auch für die durch Deforestation entstehenden Programm aus P^+. Daher werden wir die Klasse mit dieser Eigenschaft wieder gesondert behandeln. Allerdings kommt hier noch das Problem dazu, daß bei der Erzeugung der neuen rechten Seiten Kontextvariablen verlorengehen können, wenn der top-down tree transducer deleting im Rekursionsargument ist.

Beispiel 4.6 Wir betrachten den macro tree transducer M_1 mit der Gleichung $f\ \alpha\ y_1\ y_2 = \sigma\ y_1\ y_2$, welche nondeleting in den Kontextparametern ist, und den top-down tree transducer M_2 mit der Gleichung $g\ (\sigma\ x_1\ x_2) = x_1$, welche deleting im Rekursionsargument ist. Bei der Komposition von M_1 mit M_2 entsteht die Gleichung $\overline{fg}\ \alpha\ y_1\ y_2 = y_1$, welche deleting in den Kontextparametern ist. □

Ein weiteres Problem bei der Komposition ist die Vervielfachung der Kontextparameter entsprechend der Anzahl der Funktionen des top-down tree transducers. Dadurch geht die nondeleting-Eigenschaft in den Kontextparametern ebenfalls verloren, wie das folgende Beispiel zeigt.

Beispiel 4.7 Wir betrachten den macro tree transducer M_1 mit der Gleichung $f\ \alpha\ y_1 = y_1$ und einen top-down tree transducer M_2 mit zwei Funktionen g_1 und g_2. Bei der Komposition von M_1 mit M_2 entstehen die Gleichungen $\overline{fg_1}\ \alpha\ y_1\ y_2 = y_1$ und $\overline{fg_2}\ \alpha\ y_1\ y_2 = y_2$, welche beide deleting in den Kontextparametern sind. □

Deshalb beschränken wir uns auf die Kompositionsklasse $(ndc\text{-}MAC; 1f\text{-}ndr\text{-}TOP)$.

4.3.1 Hilfsaussagen

Im folgenden sei $p \in P$ ein Programm, welches die tree transducer $M_1 \in ndc\text{-}MAC$ und $M_2 \in 1f\text{-}ndr\text{-}TOP$ enthält, sowie $e = g\ (f_0\ z_0\ z_1 \ldots z_{n_1})$ ein Kompositionsausdruck mit $n_1 \in I\!N$, $f_0^{(1+n_1)} \in F_{M_1}$ und $F_{M_2} = \{g^{(1)}\}$.
Als erstes zeigen wir, daß bei der Komposition die nondeleting-Eigenschaft in den Kontextparametern vom ersten tree transducer auf den resultierenden tree transducer übertragen wird.

Lemma 4.8 Sei M der durch die Komposition von M_1 und M_2 nach Transformation 2.25 entstehende tree transducer. Dann ist $M \in ndc\text{-}MAC$.

Beweis: Sei $n \in I\!N$, $f^{(1+n)} \in F_{M_1}$, $k \in I\!N$ und $c^{(k)} \in C^i_{M_1}$. Gemäß Punkt 2. von Transformation 2.25 enthält R_M die Regel

$$\overline{fg}\ (c\ x_1 \ldots x_k)\ y_1 \ldots y_{h \cdot n} = nf(\Rightarrow_{M_2'}, g\ (rhs(f,c)))$$

Wegen $M_2 \in 1f\text{-}TOP$ gilt $h = 1$. Wir müssen also zeigen, daß $rhs(\overline{fg}, c)$ alle Variablen aus Y_n enthält. Weil $M_1 \in ndc\text{-}MAC$, sind in $rhs(f,c)$ alle Variablen aus Y_n vorhanden. Die Regelmenge von M_2' ist eine Erweiterung der Regelmenge von M_2. Da $M_2 \in ndr\text{-}TOP$ und die Erweiterungsregeln ebenfalls nondeleting im Rekursionsargument sind, ist $M_2' \in ndr\text{-}MAC$, weshalb bei der Übersetzung kein Teilbaum von $rhs(f,c)$ verlorengeht. Die Regeln von M_2' zur Übersetzung der Kontextvariablen berechnen wegen $M_2 \in 1f\text{-}TOP$ für alle Kontextvariablen die Identität. Damit enthält das Reduktionsergebnis, welches $rhs(\overline{fg}, c)$ darstellt, alle Variablen aus Y_n. □

Jetzt stellen wir einige Vorbetrachtungen zu Deforestation an. Um beweisen zu können, daß auch hier die nondeleting-Eigenschaft in den Kontextparametern erhalten bleibt, zeigen wir zunächst, daß durch die Transformationsfunktion $\mathcal{D}[\![.]\!]$ rechte Regelseiten von macro tree transducern identisch repliziert werden.

Lemma 4.9 Für alle $r \in RHS(F,C,k,n)$ gilt $\mathcal{D}[\![r]\!] = r$.

Beweis: Der Beweis erfolgt durch strukturelle Induktion über den Aufbau der Menge $RHS(F,C,k,n)$.

(i) $r = y_j$ mit $j \in [n]$: Die Regel $(D1)$ berechnet für Kontextvariablen die Identität.

(ii) $r = (c\ r_1 \ldots r_l)$ mit $l \in [k]$, $c^{(l)} \in C$ und $r_1, \ldots, r_l \in RHS(F,C,k,n)$: Die Regel $(D2)$ repliziert die Kontruktoren einer rechten Seite und nach Induktionsvoraussetzung auch deren Nachfolger identisch.

(iii) $r = (f\ x_i\ r_1 \ldots r_l)$ mit $l \in [n]$, $f^{(1+l)} \in F$, $i \in [k]$ und $r_1, \ldots, r_l \in RHS(F,C,k,n)$: Die Regel $(D4)$ repliziert einen Funktionsaufruf und nach Induktionsvoraussetzung auch deren Kontextparameter identisch. □

Damit können wir zeigen, daß im resultierenden Programm Regeln der ursprünglichen tree transducer M_1 und M_2 auftreten können.

Lemma 4.10 Sei $n \in I\!N$, $f^{(1+n)} \in F_{M_1}$ und $t'_1, \ldots, t'_{n_1} \in T_{C^o_{M_1}}$. Die in der Regel $(D4)$ bei der Berechnung von $\mathcal{D}[\![f\ v\ t'_1 \ldots t'_{n_1}]\!]$ bzw. $\mathcal{D}[\![g\ v]\!]$ generierten Funktionsgleichungen für f und g sind dieselben wie die im ursprünglichen Programm.

Beweis: Sei $c \in C^i_{M_1}$ und $c' \in C^i_{M_2}$. Die in Regel $(D4)$ erzeugten rechten Seiten der neuen (f, c)- bzw. (g, c')-Regeln entstehen durch $\mathcal{D}[\![rhs(f, c)]\!]$ bzw. $\mathcal{D}[\![rhs(g, c')]\!]$. Dies sind wegen Lemma 4.9 die ursprünglichen rechten Seiten. □

Nun können wir beweisen, daß die nondeleting-Eigenschaft in den Kontextparametern von M_1 auf das resultierende Programm übertragen wird.

Lemma 4.11 Sei $p' \in P^+$ das bei der Berechnung von $\mathcal{D}[\![e]\!]$ nach Transformation 2.28 entstehende Programm. Dann sind alle Regeln aus p' nondeleting in den Kontextparametern.

Beweis: Sei $n \in I\!N$, $f^{(1+n)} \in F_{M_1}$ und $c \in C^i_{M_1}$. Die Gleichungen von p' werden in den Regeln $(D4)$ und $(D5)$ erzeugt:

- Die in Regel $(D4)$ für f erzeugten Gleichungen von p' sind nondeleting in den Kontextparametern, weil dies wegen Lemma 4.10 die ursprünglichen Regeln für f aus p sind, welche nach Voraussetzung nondeleting in den Kontextparametern sind. Die für g erzeugten Regeln sind ebenfalls nondeleting in den Kontextparametern, weil dies die ursprünglichen Regeln eines top-down tree transducers sind, welcher keine Kontextparameter besitzt.
- Die in Regel $(D5)$ erzeugte rechte Seite der Gleichung der (fg, c)-Regel von p' entsteht durch $\mathcal{D}[\![g\ rhs(f, c)]\!]$. Weil nach Voraussetzung die (f, c)-Regel nondeleting in den Kontextparametern ist, enthält $rhs(f, c)$ alle Kontextvariablen aus Y_n. Wir zeigen, daß bei Anwendung der Regeln von $\mathcal{D}[\![.]\!]$ keine dieser Variablen verloren geht.
  - Regel $(D1)$ berechnet für Kontextvariablen die Identität.
  - Regel $(D2)$ behandelt alle Nachfolger eines Konstruktors, dabei geht also nichts verloren.
  - Regel $(D3)$ führt die Transformation $\mathcal{D}[\![g\ (c'\ t_1 \ldots t_k)]\!] = \mathcal{D}[\![rhs(g, c')[x_i/t_i]]\!]$ durch, wobei $k \in I\!N$, $c' \in C^o_{M_1}$ und $t_1, \ldots, t_k$ Teilbäume von einer rechten Regelseite von M_1 sind. Weil $M_2 \in ndr\text{-}TOP$, geht dabei kein solcher Baum, in welchem ja die Kontextvariablen enthalten sind, verloren.
  - Regel $(D4)$ transformiert alle Kontextparameter des Funktionsaufrufs.
  - Regel $(D5)$ verwendet alle vorhandenen Teilbäume weiter, so daß auch hier nichts verloren geht.

  Also sind in der neuen rechten Seite ebenfalls alle Kontextvariablen aus Y_n enthalten, weswegen diese Gleichung nondeleting in den Kontextparametern ist. □

Als nächstes untersuchen wir das Aussehen der Regeln von neu generierten Funktionen. Sei $f \in F_{M_1}$ und $c \in C^i_{M_1}$. Die Gleichung der neuen Funktion fg für das Eingabesymbol c wird in Regel $(D5)$ erzeugt, indem $rhs(f, c)$ durch die Funktion g aus M_2 übersetzt wird. Entsprechend der Zonenaufteilung dieser rechten Seite lassen sich dabei drei Fälle unterscheiden.

- Die Konstruktoren des äußeren Teils von $rhs(f, c)$ werden mit Hilfe von Regel $(D3)$ durch g übersetzt. Die Konstruktoren der dabei verwendeten rechten Regelseiten von g bilden den oberen Teil der (fg, c)-Regel, weil diese durch Anwendung von Regel $(D2)$ unverändert übernommen werden.
- Irgendwann stößt g bei der Übersetzung auf den Mittelteil. Dabei gibt es zwei Möglichkeiten.
  - Wenn g auf die Kontextvariable y_j angewendet wird, bleibt der Aufruf $(g\ y_j)$ nach Regel $(D4)$ in der neuen rechten Seite so stehen, und es werden gegebenenfalls die alten Regeln von g laut Lemma 4.10 im neuen Programm generiert.
  - Wenn g auf den Aufruf einer Funktion f' angewendet wird, dann generiert Regel $(D5)$ einen Aufruf der neuen Funktion $f'g$ und gegebenenfalls Regeln für diese neue Funktion, falls dies noch nicht geschehen war.
- Der Kontext von Funktionsaufrufen des Mittelteils, also der gesamte untere Teil, wird nach Lemma 4.9 identisch repliziert und so zum Kontext von neuen Funktionsaufrufen. Für die darin vorkommenden Funktionen werden nach Lemma 4.10 die alten Gleichungen übernommen.

Beispiel 4.12 Wir betrachten die rechte Seite $r = (\sigma(\sigma\ \alpha\ (f\ x_1\ \alpha\ (f\ x_1\ \alpha\ \alpha\ \alpha)\ y_2))\ y_3)$ aus Beispiel 4.3. Die Abbildung 4.3 zeigt die daraus entstehende rechte Seite einer neuen Regel, welche durch Berechnung von $\mathcal{D}[\![(g\ r)]\!]$ erzeugt wird. Für g gelten dabei die Regeln $g\ (\sigma\ x_1\ x_2) = \varrho\ (g\ x_1)\ (g\ x_2)$ und $g\ \alpha = \beta$.

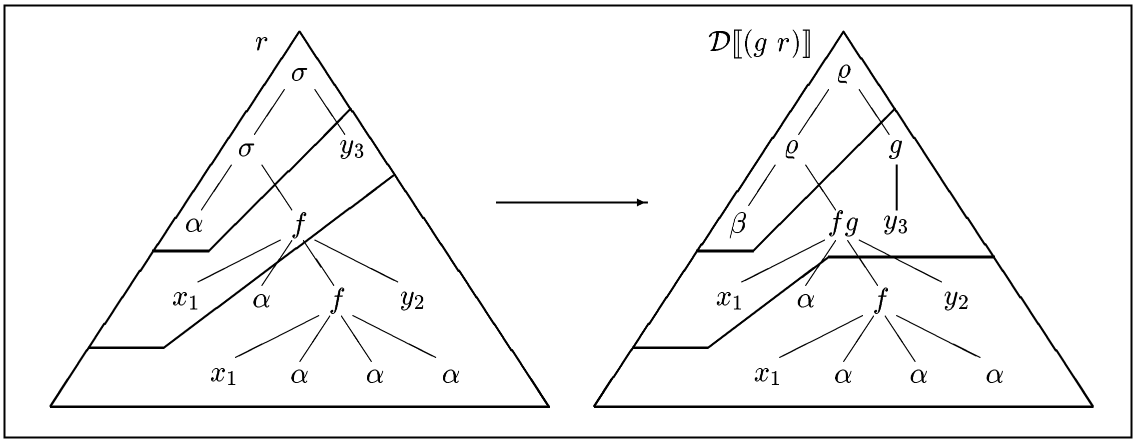

Abbildung 4.3: Entstehung der rechten Seiten bei Deforestation

Wir erkennen, daß der obere Teil der neuen rechten Seite aus Ausgabesymbolen von M_2 aufgebaut ist. Der Mittelteil besteht nun aus einem Aufruf der Funktion g auf eine Kontextvariable sowie einem Aufruf einer neuen Funktion nebst deren Rekursionsargument. Der untere Teil enthält die unveränderten Kontruktoren, Kontextvariablen und Funktionsaufrufe mit deren Rekursionsargument und Kontextparametern des unteren Teils der alten rechten Seite. □

Bei der Transformation des Kompositionsausdrucks mit Hilfe der Funktion $\mathcal{D}$ ergibt sich der Ergebnisausdruck $(f_0g\ z_0\ z_1 \ldots z_{n_1})$. Der Reduktionsvorgang beginnt also mit der Anwendung einer neuen Funktion. Der eben betrachtete Aufbau der rechten Seiten dieser Funktionen hat zur Folge, daß sich die Reduktionsschritte in drei Gruppen einteilen lassen:

(1) Es wird mit der Anwendung einer neuen Funktion im Eingabebaum begonnen, wonach die im Mittelteil der Regeln für neue Funktionen vorhandenen Aufrufe neuer Funktionen durchgeführt werden.

(2) Ein im Mittelteil einer neuen Regel vorkommender Aufruf der Funktion g auf einen Kontextparameter erzwingt die Auswertung von Aufrufen alter Funktionen, die neben Konstruktoren in dem für den Kontextparameter eingesetzten Baum enthalten sind. Diese Funktionen arbeiten ebenfalls auf dem Eingabebaum.

(3) Schließlich werden durch g die für die Kontextparameter eingesetzten Bäume verarbeitet. Darin sind neben Konstruktoren aus dem unteren Teil von neuen Regeln auch die von den alten Funktionen bei Reduktionsschritten der Gruppe (2) erzeugten Zwischenergebnisse enthalten.

Wir ordnen nun die Reduktionsschritte so um, daß zunächst alle Schritte der ersten Gruppe, dann die der zweiten und zuletzt die der dritten Gruppe ausgeführt werden. Bei der call-by-need-Auswertung wird diese Reihenfolge in der Regel nicht eingehalten, sondern es werden Reduktionsschritte verschiedener Gruppen gemischt. Da die nichtdeterministische Ableitungsrelation jedoch die gleichen Schritte durchführt und das gleiche Ergebnis liefert, ist die hier vorgenommene Sortierung zulässig. Wir erhalten also eine Aufteilung des Reduktionsvorganges in drei Phasen. Dies wollen wir an einem konkreten Beispiel veranschaulichen.

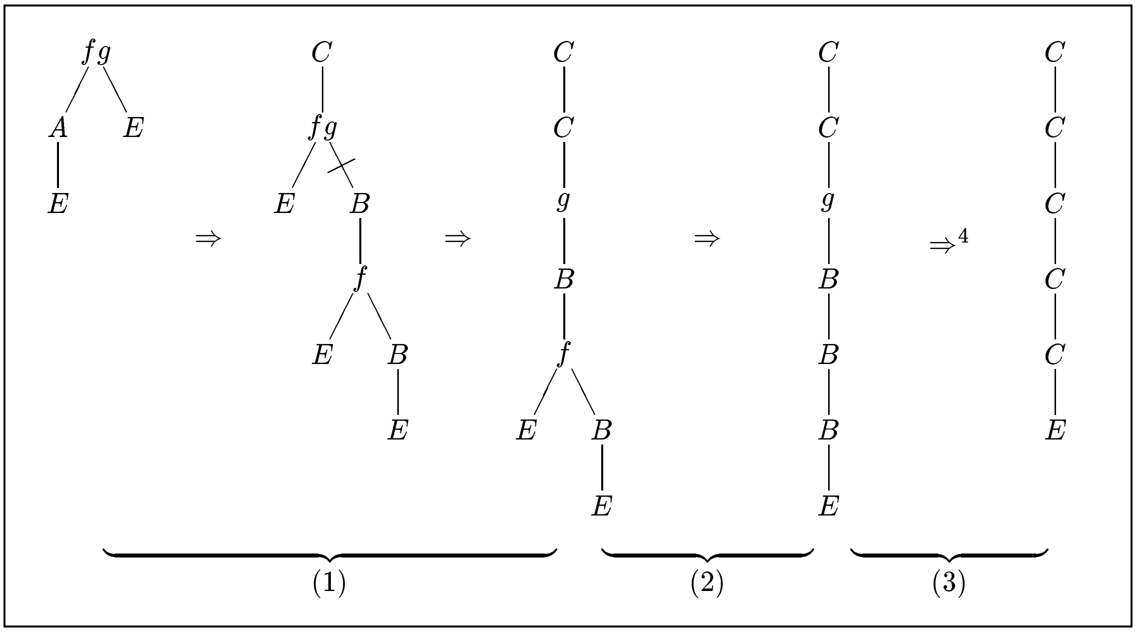

Abbildung 4.4: Beispiel für die Reduktionsphasen nach Deforestation

Beispiel 4.13 Gegeben sei ein Programm, welches aus zwei auf monadischen Bäumen arbeitenden tree transducern $M_1 \in ndc\text{-}MAC$ und $M_2 \in 1f\text{-}ndr\text{-}TOP$ besteht, wobei $F_{M_1} = \{f^{(2)}\}$ und $F_{M_2} = \{g^{(1)}\}$.

$$
\begin{array}{lcl}
f\ (A\ x_1)\ y_1 & = & B\ (f\ x_1\ (B\ (f\ x_1\ (B\ y_1)))) \\
f\ E\ y_1 & = & B\ y_1 \\[4pt]
g\ (B\ x_1) & = & C\ (g\ x_1) \\
g\ E & = & E
\end{array}
$$

Nun betrachten wir den Kompositionsausdruck $g\ (f\ z_0\ z_1)$. Deforestation erzeugt nach Transformation 2.28 den neuen Ausdruck $(fg\ z_0\ z_1)$ und ein neues Programm, das neben den ursprünglichen Regeln folgende neue Regeln enthält.

$$\begin{array}{lcl} fg\ (A\ x_1)\ y_1 & = & C\ (fg\ x_1\ (B\ (f\ x_1\ (B\ y_1)))) \\ fg\ E\ y_1 & = & C\ (g\ y_1) \end{array}$$

Die Ausdrucksvariablen z_0 und z_1 belegen wir mit den Bäumen AE und E. In Abbildung 4.4 ist der Reduktionsvorgang für $(fg\ (AE)\ E)$ nach Phasen geordnet dargestellt. □

Wir wollen versuchen, zu Vergleichszwecken die Zahl der Reduktionsschritte in Abhängigkeit von der Anzahl der Reduktionsschritte anzugeben, welche die beiden ursprünglichen tree transducer benötigen. Der dabei auftretende Reduktionsvorgang läßt sich auf ähnliche Weise in Phasen unterteilen:

(1a) Es wird mit einer Funktion von M_1 im Eingabebaum begonnen, und es werden die im Mittelteil der Regeln vorhandenen Aufrufe solcher Funktionen durchgeführt.

(1b) Die Kontextvariablen im Mittelteil erzwingen die Auswertung der in den dafür eingesetzten Bäumen neben Konstruktoren enthaltenen Aufrufe von Funktionen aus M_1, welche ebenfalls auf dem Eingabebaum arbeiten.

(2a) Die von den äußeren Konstruktoren und den Funktionsaufrufen im Mittelteil der Regeln von M_1 erzeugten Zwischenergebnisse werden von der Funktion g aus M_2 verarbeitet.

(2b) Zuletzt werden durch g die Teilbäume des Zwischenergebnisses verarbeitet, die für die Kontextparameter aus den Mittelteilen der Regeln von M_1 eingesetzt wurden. Darin sind auch die in Phase (1b) erzeugten Zwischenergebnisse enthalten.

Auch diese Phasen verdeutlichen wir an unserem Beispiel.

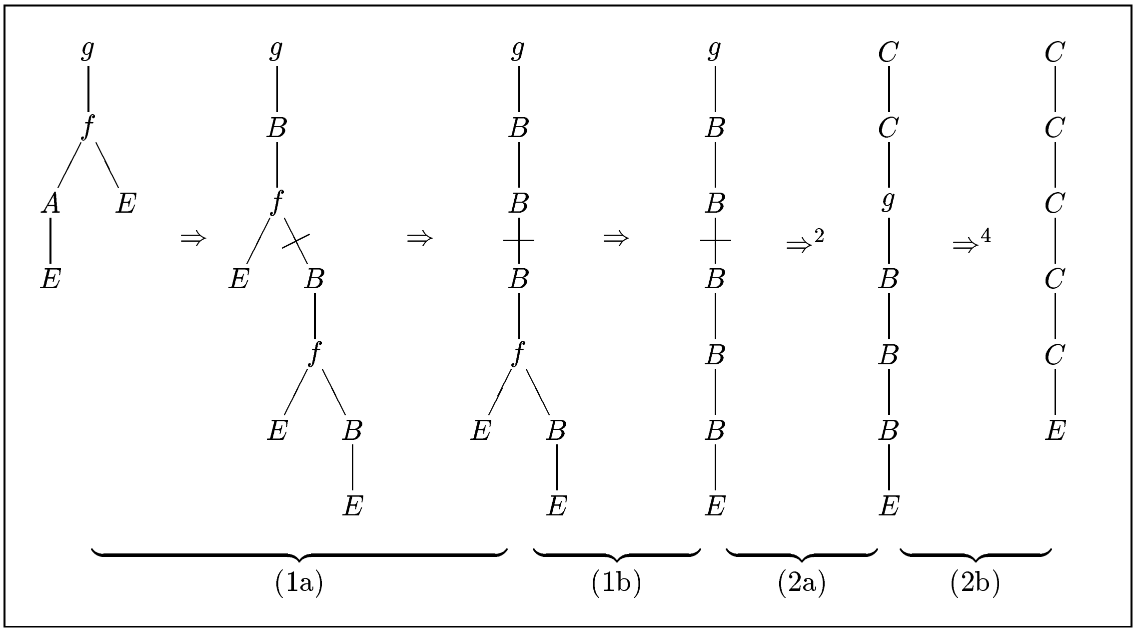

Abbildung 4.5: Beispiel für die Phasen bei zwei tree transducern

Beispiel 4.14 Gegeben sei das Programm und der Kompositionsausdruck aus Beispiel 4.13. Die Ausdrucksvariablen z_0 und z_1 belegen wir wiederum mit den Bäumen AE und E. In Abbildung 4.5 zeigen wir den Reduktionsvorgang durch die beiden tree transducer für $g\ (f\ (AE)\ E)$ nach Phasen geordnet.

Wenn wir die beiden Abbildungen aus diesem und dem vorigen Beispiel vergleichen, fällt zunächst auf, daß die Phasen (3) und (2b) identisch sind. Außerdem erkennen wir, daß die in Phase (2a) durchgeführte Übersetzung des oberen Teils des Zwischenergebnisses durch Deforestation bereits in Phase (1a) integriert wird, womit wir Phase (1) erhalten. Die Phasen (1b) und (2) ähneln sich weitgehend, da die durchgeführten Reduktionen gleich sind. Der Unterschied besteht wie eben erwähnt nur darin, daß nach Deforestation die Konstruktoren im oberen Teil schon übersetzt sind, bei der Reduktion durch die ursprünglichen tree transducder aber noch nicht. Dadurch kann es passieren, daß die in Phase (1b) durchgeführten Schritte in Phase (2) vervielfacht werden, wenn der zweite tree transducer nichtlinear ist. In diesem Fall wären die Phasen (1a) und (1) bezüglich ihrer Reduktionsschrittanzahl auch nicht mehr gleich. □

Um die verfrühte Bearbeitung der Kontextparameter in den Phasen (1a) und (1) zu verhindern, betrachten wir die Kontextvariablen im Mittelteil zunächst als nullstellige Ausgabesymbole. Diese Vorgehensweise demonstrieren wir noch einmal an unserem Beispiel.

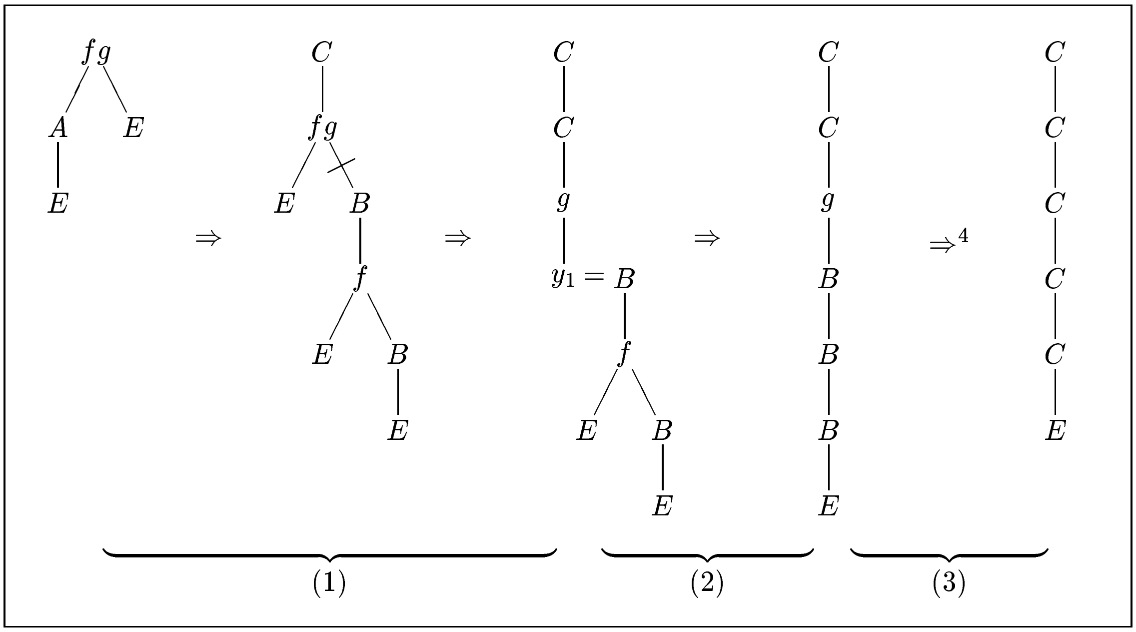

Abbildung 4.6: Beispiel für die Phasen mit Kontextvariablen nach Deforestation

Beispiel 4.15 Wir verwenden wieder das Programm mit den beiden tree transducern aus Beispiel 4.13. Den dort gezeigten Reduktionsvorgang nach Deforestation und den aus Beispiel 4.14 für die tree transducer führen wir nun mit verzögerter Ersetzung der Kontextvariablen durch. Dies ist in den Abbildungen 4.6 und 4.7 dargestellt. □

Wie in Abbildung 4.7 schon angedeutet, lassen sich die Vorkommen des Zwischenergebnisses nach der Phase, in welcher sie erzeugt werden, in Gruppen einteilen. Dabei bezeichnen wir die Menge der Vorkommen aus t_z, die in Phase (1a) erzeugt werden, mit $occ_{top}(t_z)$. Die nach Abschluß dieser Phase mit Kontextvariablen beschrifteten Grenzvorkommen, welche

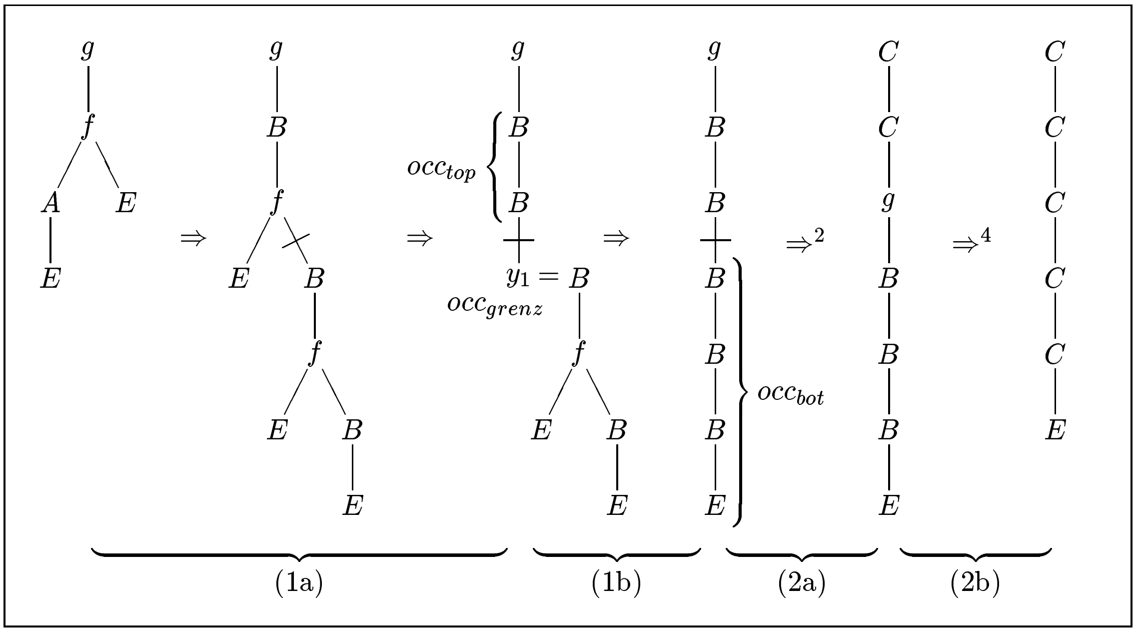

Abbildung 4.7: Beispiel für die Phasen mit Kontextvariablen bei zwei tree transducern

wir mit $occ_{grenz}(t_z)$ bezeichnen, gehören jedoch nicht zu dieser Menge, da sie am Ende von Phase (1b) noch ersetzt werden. Die Menge der in jener Phase entstehenden Vorkommen bezeichnen wir mit $occ_{bot}(t_z)$. Dabei gilt für die Grenzvorkommen die Beziehung $occ_{grenz}(t_z) = \{o \mid o \in occ_{bot}(t_z),\ o = o'.i,\ o' \in occ_{top}(t_z),\ i \in \mathbb{N}\}$. Zur Unterscheidung der Zahl der Reduktionsschritte in den einzelnen Phasen erweitern wir die Notationen aus Definition 2.15 und 2.19, indem wir sie mit einem Index versehen, der die Phase angibt. Die verschiedenen Phasen, die vorübergehende Behandlung der Kontextvariablen als Ausgabesymbole und die eben eingeführten Notationen sind noch einmal zusammenfassend in Abbildung 4.8 dargestellt.

Jetzt geben wir die Gesamtzahl der nach Deforestation benötigten Reduktionsschritte in Abhängigkeit von den durch die tree transducer benötigten Schritte an.

Lemma 4.16 Die Zahl der Reduktionsschritte nach Deforestation läßt sich mit folgender Formel berechnen, wobei im Teil (2) jeweils $label(o_e, t_e) = c^{(k)}$ mit $k \in \mathbb{N}$ gilt.

$$\begin{aligned} r^{f_0 g, t_e} &\overset{(1)}{=} \sum_{o_e \in occ(t_e)} \ \sum_{\substack{o_z \in bild^{f_0, t_e}(o_e) \\ \cap\, occ_{top}(t_z)}} m^{g, t_z}(o_z) \quad \overset{(3)}{+}\ r^{g, t_z}_{(2b)} \\ &\overset{(2)}{+} \sum_{o_e \in occ(t_e)} \ \sum_{f \in M_1} \ \sum_{\substack{f' \in F_{M_1} \\ i \in [k]}} \ \sum_{\substack{o_z \in bild^{f_0, t_e}_{f}(o_e) \\ \cap\, occ_{top}(t_z)}} m^{g, t_z}(o_z) \cdot n_{bot}(f', i, fg, c) \cdot r^{f', subtree(o_e.i, t_e)} \end{aligned}$$

Beweis: Bei der Berechnung der Reduktionsschritte, die nach Durchführung von Deforestation gebraucht werden, behandeln wir die Phasen getrennt. Wir beginnen mit der ersten Phase. Für die Zahl der insgesamt in Phase (1) benötigten Reduktionsschritte ergibt sich nach Definition 2.15 und sinngemäß nach Lemma 2.20 folgender Ansatz, da diese Schritte durch Besuche der Vorkommen im Eingabebaum durch neue Funktionen verursacht werden:

$$r^{f_0 g, t_e}_{(1)} = \sum_{o_e \in occ(t_e)} \ \sum_{fg \in F_{p'}} m^{f_0 g, t_e}_{fg}(o_e)$$

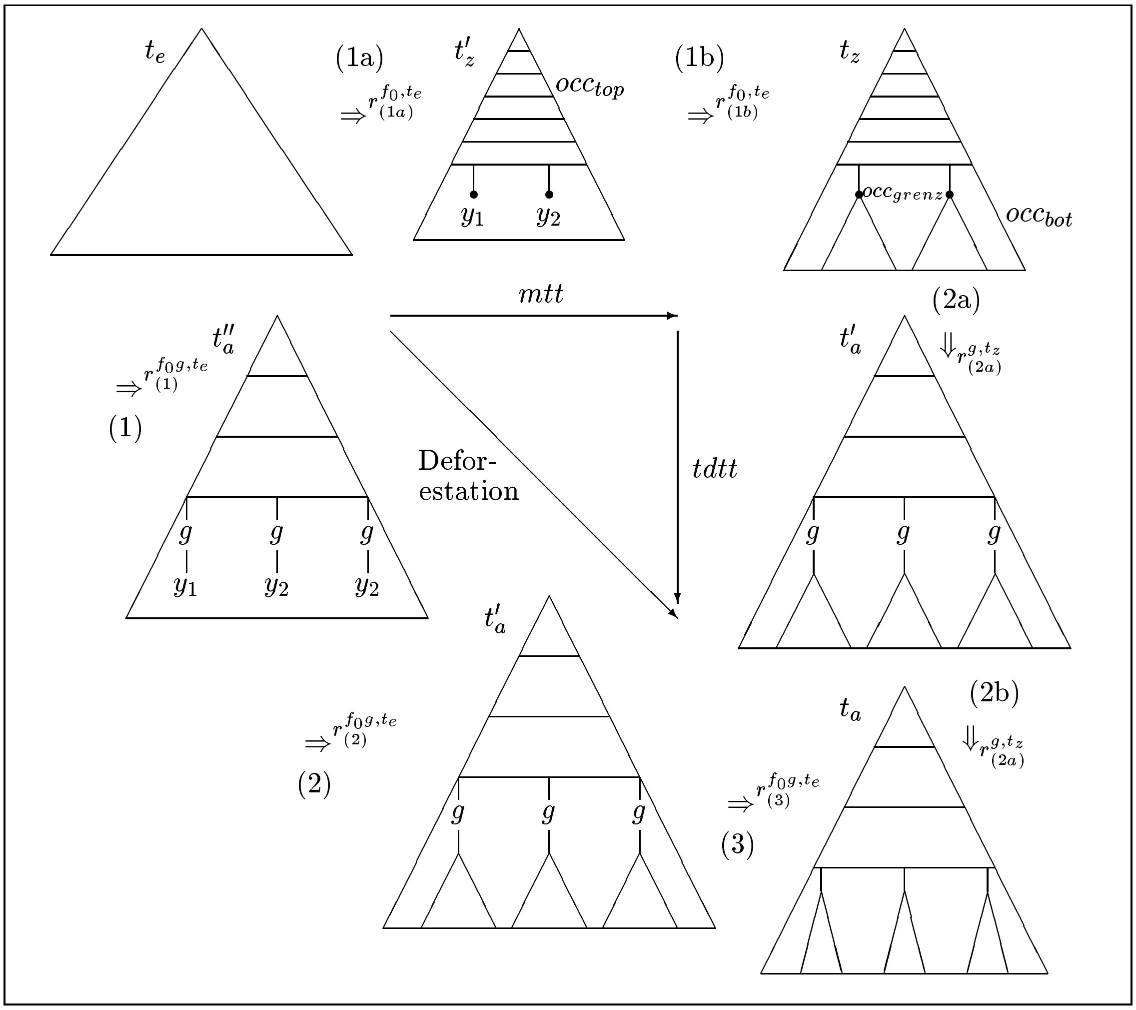

Abbildung 4.8: Phasen bei der Reduktion eines Kompositionsausdrucks durch zwei tree transducer bzw. bei der Reduktion des durch Deforestation entstandenen Ausdrucks

Weil in den Phasen (1a) und (1) die Kontextparameter keine Rolle spielen, können sie dort weggelassen werden, womit M_1 zu einem top-down tree transducer wird. Wir erhalten dadurch die Kompositionsklasse (TOP; $1f$-ndr-TOP), welche eine Teilklasse von (TOP; ndc-MAC) ist. Dies ermöglicht die Verwendung der Resultate aus Abschnitt 3.3 zur Untersuchung der ersten Phase. Aus Lemma 3.9 und Lemma 3.3 erhalten wir die folgende Beziehung für die Anzahl der Besuche eines Vorkommens, wobei die Menge der Bildvorkommen auf $occ_{top}(t_z)$ beschränkt werden muß, weil die Funktionen in den Kontextparametern später noch weitere Bildvorkommen in t_z erzeugen.

$$m_{fg}^{f_0g,t_e}(o_e) = \sum_{o_z \in \substack{bild_f^{f_0,t_e}(o_e) \\ \cap\, occ_{top}(t_z)}} m^{g,t_z}(o_z)$$

Diese Gleichung setzen wir in den Ansatz ein und erhalten unter Berücksichtigung der wegen $M_2 \in 1f$-MAC eineindeutigen Zuordnung von jedem $fg \in F_{p'}$ zu einem $f \in F_{M_1}$ die folgende Formel für die Phase (1).

$$r_{(1)}^{f_0g,t_e} = \sum_{o_e \in occ(t_e)} \sum_{f \in F_{M_1}} \sum_{o_z \in \substack{bild_f^{f_0,t_e}(o_e) \\ \cap\, occ_{top}(t_z)}} m^{g,t_z}(o_z)$$

Da wir M_1 in Phase (1a) als top-down tree transducer ansehen, darf wegen Lemma 2.17 die Definition 2.16 angewendet werden.

$$r_{(1)}^{f_0 g, t_e} = \sum_{o_e \in occ(t_e)} \sum_{\substack{o_z \in \, bild^{f_0, t_e}(o_e) \\ \cap\, occ_{top}(t_z)}} m^{g, t_z}(o_z)$$

Nun wenden wir uns Phase (2) zu. Wenn ein mit $c^{(k)}$ beschriftetes Vorkommen des Eingabebaums in Phase (1) durch eine neue Funktion besucht wird und im unteren Teil der dabei verwendeten Regel ein Aufruf der alten Funktion f' auf den i-ten Nachfolger mit $i \in [k]$ steht, dann wird der Teilbaum an diesem Nachfolger in Phase (2) durch die Funktion f' wegen der nondeleting-Eigenschaft in den Kontextparametern sowie der call-by-need-Strategie genau einmal reduziert. Durch Betrachtung all dieser Besuche und aller Funktionsaufrufe im unteren Teil der dabei verwendeten Regeln ergeben sich alle Reduktionsschritte, die in Phase (2) durchgeführt werden.

$$r_{(2)}^{f_0 g, t_e} = \sum_{o_e \in occ(t_e)} \sum_{fg \in F_{p'}} \sum_{\substack{f' \in F_{M_1} \\ i \in [k]}} m_{fg}^{f_0 g, t_e}(o_e) \cdot n_{bot}(f', i, fg, c) \cdot r^{f', subtree(o_e.i, t_e)}$$

Für die Zahl der Besuche aus Phase (1) können wir noch das dort verwendete Resultat einsetzen, wobei wir wieder jeder Funktion fg aus $F_{p'}$ genau eine Funktion f aus F_{M_1} zuordnen.

$$r_{(2)}^{f_0 g, t_e} = \sum_{o_e \in occ(t_e)} \sum_{f \in F_{M_1}} \sum_{\substack{f' \in F_{M_1} \\ i \in [k]}} \sum_{\substack{o_z \in \, bild_f^{f_0, t_e}(o_e) \\ \cap\, occ_{top}(t_z)}} m^{g, t_z}(o_z) \cdot n_{bot}(f', i, fg, c) \cdot r^{f', subtree(o_e.i, t_e)}$$

Zum Schluß betrachten wir Phase (3). Wenn ein mit c beschriftetes Vorkommen des Eingabebaums in Phase (1) durch eine neue Funktion besucht wird und im mittleren Teil der dabei verwendeten Regel ein Aufruf der alten Funktion g von M_2 auf den j-ten Kontextparameter steht, dann wird der dafür eingesetzte Baum in Phase (3) durch die Funktion g genau einmal reduziert. Dieser Baum ist ein Teilbaum des alten Zwischenergebnisses t_z. Wie in den vorherigen Betrachtungen bereits erwähnt, läßt sich dem Besuch aus Phase (1) ein Besuch eines Bildes in Phase (2a) eindeutig zuordnen. Ein Aufruf der Funktion g auf eine Kontextvariable in der neuen Regel entsteht, wenn bei deren Erzeugung g auf eine Kontextvariable in der zugehörigen alten Regel stößt. Die Konstruktoren aus deren oberen Teil erscheinen auch im Zwischenergebnis, so daß wir auf diese Weise den Pfad des Teilbaums von t_z erhalten, welcher für die Kontextvariable eingesetzt wird. Durch Betrachtung all dieser Besuche im Eingabebaum und aller solcher Funktionsaufrufe im mittleren Teil der dabei verwendeten Regeln ergeben sich alle Reduktionsschritte, die in Phase (3) durchgeführt werden.

$$r_{(3)}^{f_0 g, t_e} = \sum_{\substack{o_e \in \\ occ(t_e)}} \sum_{\substack{f^{(n)} \\ \in F_{M_1}}} \sum_{\substack{o_z \in \, bild_f^{f_0, t_e}(o_e) \\ \cap\, occ_{top}(t_z)}} \sum_{j \in [n]} \sum_{\substack{o \in occ(rhs(f,c)) \\ label(o, rhs(f,c)) \\ = y_j}} m^{g, t_z}(o_z) \cdot m^{g, rhs(f,c)}(o) \cdot r^{g, subtree(o_z.o, t_z)}$$

Der Pfad $o_z.o$ läßt sich nun zu einem neuen Pfad zusammenziehen, damit ergibt sich ein Grenzvorkommen $o_g \in occ_{grenz}(t_z)$. Die Quantifizierungen der Summen erzeugen überschneidungsfrei alle Grenzvorkommen des Zwischenergebnisses.

$$r_{(3)}^{f_0 g, t_e} = \sum_{o_g \in occ_{grenz}(t_z)} m^{g, t_z}(o_g) \cdot r^{g, subtree(o_g, t_z)}$$

Aus der Besuchsanzahl aller Grenzvorkommen multipliziert mit den Reduktionsschritten für die daran hängenden Teilbäume ergibt sich die Besuchsanzahl der Vorkommen des unteren Teils des Zwischenergebnisses, was genau die Reduktionsschritte von Phase (2b) sind.

$$r_{(3)}^{\overline{f_0 g}, t_e} = r_{(2b)}^{g, t_z}$$

Die Formel für die Gesamtzahl der nach Deforestation benötigten Reduktionsschritte ergibt sich durch Zusammenfügen der Resultate für die drei Phasen. □

Nach diesen Vorüberlegungen können wir die Effizienzvergleiche durchführen. Da sich bei $M_2 \in nlin\text{-}TOP$ die beiden Verfahren wesentlich anders als im linearen Fall verhalten, nehmen wir eine weitere Unterteilung in verschiedene Teilklassen vor.

4.3.2 Die Kompositionsklasse $(ndc\text{-}MAC; lin\text{-}1f\text{-}ndr\text{-}TOP)$

Im folgenden sei $p \in P$ ein Programm, welches $M_1 = (F_{M_1}, C^i_{M_1}, C^o_{M_1}, R_{M_1}) \in ndc\text{-}MAC$ und $M_2 = (\{g^{(1)}\}, C^i_{M_2}, C^o_{M_2}, R_{M_2}) \in lin\text{-}1f\text{-}ndr\text{-}TOP$ enthält. Wir betrachten den Kompositionsausdruck $e = g\ (f_0\ z_0\ z_1 \ldots z_{n_1})$ mit $f_0^{(1+n_1)} \in F_{M_1}$ und $n_1 \geq 0$. Weiterhin seien $t_e \in T_{C^i_{M_1}}$ und $t_1, \ldots, t_{n_1} \in T_{C^o_{M_1}}$ die Bäume, die wir für z_0 bzw. $z_1, \ldots, z_{n_1}$ einsetzen, und $t_z = nf(\Rightarrow_{M_1}, (f_0\ t_e\ t_1 \ldots t_{n_1}))$ das durch M_1 erzeugte Zwischenergebnis.

Zuerst untersuchen wir die Komposition. Aus dem Kompositionsausdruck entstehe nach Transformation 2.25 der neue macro tree transducer $M = (F_M, C^i_M, C^o_M, R_M)$ und der neue Ausdruck $(\overline{f_0 g}\ z_0\ (g\ z_1) \ldots (g\ z_{n_1}))$ mit $\overline{f_0 g}^{(1+n_1)} \in F_M$. Außerdem sei $f \in F_{M_1}$ und $\overline{fg} \in F_M$.

Wir versuchen, die nach der Komposition benötigten Reduktionsschritte in Abhängigkeit von den ursprünglichen Reduktionsschritten anzugeben, um sie vergleichen zu können. Wegen $M_2 \in lin\text{-}1f\text{-}ndr\text{-}TOP$ wird bei der Übersetzung der rechten Seiten von M_1 deren Struktur beibehalten. Deswegen stimmt die Besuchsanzahl eines Vorkommens im Eingabebaum durch eine Funktion des Kompositionsergebnisses mit der Besuchsanzahl dieses Vorkommens durch die zugehörige Funktion des ursprünglichen macro tree transducers überein.

Lemma 4.17 Für alle $o_e \in occ(t_e)$ gilt:

$$m_{\overline{fg}}^{\overline{f_0 g}, t_e}(o_e) = m_f^{f_0, t_e}(o_e)$$

Beweis: Wir beweisen das Lemma durch Induktion über alle Pfade in t_e:

Induktionsanfang ($o_e = \varepsilon$):

Für die linke Seite der Gleichung gilt:

$$m_{\overline{fg}}^{\overline{f_0 g}, t_e}(\varepsilon) = \begin{cases} 1 & \text{falls } \overline{f_0 g} = \overline{fg} \\ 0 & \text{falls } \overline{f_0 g} \neq \overline{fg} \end{cases}$$

Für die rechte Seite der Gleichung gilt:

$$m_f^{f_0, t_e}(\varepsilon) = \begin{cases} 1 & \text{falls } f_0 = f \\ 0 & \text{falls } f_0 \neq f \end{cases}$$

Weil $\overline{f_0g} = \overline{fg}$ genau dann gilt, wenn $f_0 = f$, sind die Ausdrücke auf der linken und rechten Seite gleich.

Induktionsschritt: Nun sei $label(o_e, t_e) = c^{(k)}$, $k > 0$, $c \in C^i_{M_1}$, $i \in [k]$, und als Induktionsvoraussetzung gelte für o_e die zu beweisende Beziehung. Wir gehen von o_e zum i-ten Nachfolger. Da wegen $M_1 \in ndc\text{-}MAC$, $M_2 \in 1f\text{-}ndr\text{-}TOP$ und daher wegen Lemma 4.8 auch M nondeleting in den Kontextparametern ist, ergibt sich mit Hilfe von Lemma 3.7 folgender rekursiver Ansatz.

$$m_{\overline{fg}}^{\overline{f_0g},t_e}(o_e.i) = \sum_{\overline{f'g}\in F_M} \left(m_{\overline{f'g}}^{\overline{f_0g},t_e}(o_e) \cdot n(\overline{fg}, i, \overline{f'g}, c) \right)$$

Wegen $M_2 \in 1f\text{-}TOP$ gehört zu jedem $f' \in F_{M_1}$ genau ein $\overline{f'g} \in F_M$, weshalb die Summenbildung über die Funktionen von M_1 möglich ist. Unter Verwendung der Induktionsvoraussetzung ergibt sich

$$= \sum_{f'\in F_{M_1}} \left(m_{f'}^{f_0,t_e}(o_e) \cdot n(\overline{fg}, i, \overline{f'g}, c) \right)$$

Weil die rechte Seite der $(\overline{f'g}, c)$-Regel durch Übersetzung der (f', c)-Regel mit g entsteht, wird ein Aufruf der Funktion $\overline{fg}$ auf den i-ten Nachfolger von c genau dann erzeugt, wenn während dieser Übersetzung ein Vorkommen, das mit f beschriftet und dessen erster Nachfolger x_i ist, durch g besucht wird. Dies rechtfertigt folgende Umformung:

$$= \sum_{f'\in F_{M_1}} \left(m_{f'}^{f_0,t_e}(o_e) \cdot \left(\sum_{\substack{o\in occ(rhs(f',c)) \\ label(o,rhs(f',c))=f \\ label(o.1,rhs(f',c))=x_i}} m_g^{g,rhs(f',c)}(o) \right) \right)$$

Wegen $M_2 \in lin\text{-}TOP$ wird bei der Übersetzung jedes Vorkommen genau einmal besucht.

$$= \sum_{f'\in F_{M_1}} \left(m_{f'}^{f_0,t_e}(o_e) \cdot \left(\sum_{\substack{o\in occ(rhs(f',c)) \\ label(o,rhs(f',c))=f \\ label(o.1,rhs(f',c))=x_i}} 1 \right) \right)$$

Definition 3.6 ermöglicht die folgende Umformung.

$$= \sum_{f'\in F_{M_1}} \left(m_{f'}^{f_0,t_e}(o_e) \cdot n(f, i, f', c) \right)$$

Daraus erhalten wir wegen $M_1 \in ndc\text{-}MAC$ mit Hilfe von Lemma 3.7 die Besuchsanzahl des Vorkommens $o_e.i$.

$$= m_f^{f_0,t_e}(o_e.i)$$

Weil aus der Gültigkeit der zu beweisenden Aussage für ein Vorkommen $o_e \in occ(t_e)$ auf die Gültigkeit für dessen Nachfolger geschlossen werden konnte, gilt die Aussage für alle $o_e \in occ(t_e)$. □

Aus Lemma 4.17 läßt sich ein ähnlicher Zusammenhang für die Gesamtanzahl der Besuche eines Vorkommens im Eingabebaum herleiten.

Lemma 4.18 Für alle $o_e \in occ(t_e)$ gilt:

$$m^{\overline{f_0 g}, t_e}(o_e) = m^{f_0, t_e}(o_e)$$

Beweis:

$$\begin{aligned} m^{\overline{f_0 g}, t_e}(o_e) &= \sum_{\overline{fg} \in F_M} m_{\overline{fg}}^{\overline{f_0 g}, t_e}(o_e) && \text{(Definition 2.15)} \\ &= \sum_{f \in F_{M_1}} m_f^{f_0, t_e}(o_e) && \text{(Lemma 4.17, zu jedem } f \text{ genau ein } \overline{fg}\text{)} \\ &= m^{f_0, t_e}(o_e) && \text{(Definition 2.15)} \end{aligned}$$

□

Aufgrund dieses Zusammenhangs können wir die Zahl der Reduktionsschritte, die das Kompositionsergebnis benötigt, mit Hilfe der Schritte ausdrücken, die die ursprünglichen tree transducer benötigen.

Lemma 4.19 $\quad r^{(\overline{f_0 g}\ t_e\ t_1 \ldots\ t_{n_1})} = r^{f_0, t_e} + \sum_{i=1}^{n_1} r^{g, t_i}$

Beweis:

$$\begin{aligned} r^{(\overline{f_0 g}\ t_e\ t_1 \ldots\ t_{n_1})} &= \sum_{o_e \in occ(t_e)} m^{\overline{f_0 g}, t_e}(o_e) + \sum_{i=1}^{n_1} r^{(g\ t_i)} && \text{(Lemma 2.20)} \\ &= \sum_{o_e \in occ(t_e)} m^{f_0, t_e}(o_e) + \sum_{i=1}^{n_1} r^{g, t_i} && \text{(Lemma 4.18, Definition 2.19)} \\ &= r^{f_0, t_e} + \sum_{i=1}^{n_1} r^{g, t_i} && \text{(Lemma 2.20)} \end{aligned}$$

□

Als nächstes untersuchen wir Deforestation. Aus dem Kompositionsausdruck wird dabei ein neues Programm p' und der neue Ausdruck ($f_0 g\ z_0\ z_1 \ldots z_{n_1}$) mit $f_0 g^{(1+n_1)} \in F_{p'}$ erzeugt.

Wir geben im folgenden die Zahl der Reduktionsschritte für den neuen Ausdruck an.

Lemma 4.20 $\quad r^{f_0 g, t_e} = r^{f_0, t_e} + r_{(2b)}^{g, t_z}$

Beweis: Wir formen die allgemeine Formel aus Lemma 4.16 für diesen Fall um. Dabei gilt im Teil für Phase (2) jeweils $c^{(k)} = label(o_e, t_e)$ mit $k \in I\!N$.

$$\begin{aligned} r^{f_0 g, t_e} &\overset{(1)}{=} \sum_{o_e \in occ(t_e)} \sum_{\substack{o_z \in bild^{f_0, t_e}(o_e) \\ \cap occ_{top}(t_z)}} m^{g, t_z}(o_z) \overset{(3)}{+} r_{(2b)}^{g, t_z} \\ &\overset{(2)}{+} \sum_{o_e \in occ(t_e)} \sum_{f \in F_{M_1}} \sum_{\substack{f' \in F_{M_1} \\ i \in [k]}} \sum_{\substack{o_z \in bild_f^{f_0, t_e}(o_e) \\ \cap occ_{top}(t_z)}} m^{g, t_z}(o_z) \cdot n_{bot}(f', i, fg, c) \cdot r^{f', subtree(o_e.i, t_e)} \end{aligned}$$

Wegen $M_2 \in$ *lin-ndr-TOP* wird jedes Vorkommen im Zwischenergebnis genau einmal besucht. Außerdem bleiben dadurch bei der Erzeugung von neuen Regeln die Funktionsaufrufe im unteren Regelteil unverändert erhalten.

$$\begin{aligned} r^{f_0 g, t_e} &\overset{(1)}{=} \sum_{o_e \in occ(t_e)} \sum_{\substack{o_z \in \, bild^{f_0, t_e}(o_e) \\ \cap occ_{top}(t_z)}} 1 \qquad \overset{(3)}{+} r_{(2b)}^{g, t_z} \\ &\overset{(2)}{+} \sum_{o_e \in occ(t_e)} \sum_{f \in F_{M_1}} \sum_{\substack{f' \in F_{M_1} \\ i \in [k]}} \sum_{\substack{o_z \in \, bild_f^{f_0, t_e}(o_e) \\ \cap occ_{top}(t_z)}} n_{bot}(f', i, f, c) \cdot r^{f', subtree(o_e.i, t_e)} \end{aligned}$$

Weil zwei verschiedene Besuche eines Eingabevorkommens auch zwei verschiedene Bildvorkommen erzeugen, wenn diese im oberen Teil des Zwischenergebnisses liegen, ist die Zahl der Bildvorkommen im oberen Teil gleich der Zahl der für die Erzeugung verantwortlichen Besuche im Eingabebaum aus Phase (1a).

$$\begin{aligned} r^{f_0 g, t_e} &\overset{(1)}{=} \sum_{o_e \in occ(t_e)} m_{(1a)}^{f_0, t_e}(o_e) \qquad \overset{(3)}{+} r_{(2b)}^{g, t_z} \\ &\overset{(2)}{+} \sum_{o_e \in occ(t_e)} \sum_{f \in F_{M_1}} \sum_{\substack{f' \in F_{M_1} \\ i \in [k]}} m_{f(1a)}^{f_0, t_e}(o_e) \cdot n_{bot}(f', i, f, c) \cdot r^{f', subtree(o_e.i, t_e)} \end{aligned}$$

Für den Formelteil (2) gilt, daß der Teilbaum am i-ten Nachfolger eines Vorkommens im Eingabebaum in Phase (1b) durch eine Funktion f' reduziert wird, wenn dieses Vorkommen in Phase (1a) durch eine Funktion besucht wird und im unteren Teil der dabei verwendeten Regel ein Funktionsaufruf von f' auf diesen Nachfolger steht. Durch Betrachtung all dieser Besuche und aller Funktionsaufrufe im unteren Teil der dabei verwendeten Regeln ergeben sich alle Reduktionsschritte, die in Phase (1b) durchgeführt werden.

$$\begin{aligned} r^{f_0 g, t_e} &\overset{(1)}{=} \sum_{o_e \in occ(t_e)} m_{(1a)}^{f_0, t_e}(o_e) \qquad \overset{(3)}{+} r_{(2b)}^{g, t_z} \\ &\overset{(2)}{+} \sum_{o_e \in occ(t_e)} m_{(1b)}^{f_0, t_e}(o_e) \end{aligned}$$

Die Summe der Besuche aus den Phasen (1a) und (1b) ist die Gesamtzahl der Besuche bei der Übersetzung des Eingabebaums durch M_1. Nach Anwendung von Lemma 2.20 erhalten wir schließlich

$$r^{f_0 g, t_e} = r^{f_0, t_e} + r_{(2b)}^{g, t_z}$$

Damit haben wir die zu beweisende Beziehung hergeleitet. □

Nachdem es uns nun gelungen ist, die nach Komposition bzw. Deforestation benötigten Reduktionsschritte in Abhängigkeit von den vorher benötigten Schritten anzugeben, können wir zum Vergleich übergehen. Zuerst treffen wir eine Aussage zu der durch die Komposition erreichten Effizienzsteigerung.

Satz 4.21 Für alle $(p, e) \in (ndc\text{-}MAC; lin\text{-}1f\text{-}ndr\text{-}TOP)$ gilt: $\mathcal{E}(Com(p, e)) \geq \mathcal{E}(p, e)$.

Beweis: Die ursprünglichen tree transducer benötigen zusammen $r^{f_0, t_e} + r^{g, t_z}$ Schritte, das Kompositionsergebnis laut Lemma 4.19 aber $r^{f_0, t_e} + \sum_{i=1}^{n_1} r^{g, t_i}$ Schritte. Wegen $M_2 \in$ *ndr-TOP* ist $\sum_{i=1}^{n_1} r^{g, t_i} \leq r^{g, t_z}$. Folglich gilt $r^{(\overline{f_0 g}\ t_e\ t_1 \ldots\ t_{n_1})} \leq r^{f_0, t_e} + r^{g, t_z}$, womit der Satz bewiesen ist. □

Es werden also diejenigen Reduktionsschritte eingespart, die zur Weiterverarbeitung von Konstruktoren im Zwischenergebnis, die nicht aus den Kontext-Eingabebäumen stammen, erforderlich sind.

Wir spezifizieren die $\geq$-Beziehung durch folgende Fallunterscheidung bezüglich der Regeln von M_1 noch etwas genauer.

- $\mathcal{E}(Com(p,e)) > \mathcal{E}(p,e)$, falls alle Regeln producing sind, dann enthält das Zwischenergebnis außer den ursprünglichen Bäumen immer mindestens einen Konstruktor.
- $\mathcal{E}(Com(p,e)) \geq \mathcal{E}(p,e)$, falls einige Regeln producing sind, denn dann enthält das Zwischenergebnisses möglicherweise, aber nicht immer einen zusätzlichen Konstruktor.
- $\mathcal{E}(Com(p,e)) = \mathcal{E}(p,e)$, falls keine Regel producing ist, weil dann keine Zwischenergebnisse erzeugt werden.

Jetzt wenden wir uns der durch Deforestation erzielten Effizienzsteigerung zu.

Satz 4.22 Für alle $(p,e) \in (ndc\text{-}MAC; lin\text{-}1f\text{-}ndr\text{-}TOP)$ gilt: $\mathcal{E}(Def(p,e)) \geq \mathcal{E}(p,e)$.

Beweis: Deforestation benötigt laut Lemma 4.20 $r^{f_0,t_e} + r^{g,t_z}_{(2b)}$ Reduktionsschritte, die beiden ursprünglichen tree transducer zusammen aber $r^{f_0,t_e} + r^{g,t_z}$ Schritte. Wegen $r^{g,t_z}_{(2b)} \leq r^{g,t_z}$ ist der Satz bewiesen. □

Es werden also die Reduktionsschritte eingespart, die während der Verarbeitung des oberen Teils des Zwischenergebnisses durchgeführt werden.

Wir spezifizieren die $\geq$-Beziehung durch folgende Fallunterscheidung bezüglich der Regeln von M_1 noch etwas genauer.

- $\mathcal{E}(Def(p,e)) > \mathcal{E}(p,e)$, falls alle Regeln außen producing sind, dann enthält der obere Teil des Zwischenergebnisses immer mindestens einen Konstruktor.
- $\mathcal{E}(Def(p,e)) \geq \mathcal{E}(p,e)$, falls einige Regeln außen producing sind, denn dann enthält der obere Teil des Zwischenergebnisses möglicherweise, aber nicht immer einen Konstruktor.
- $\mathcal{E}(Def(p,e)) = \mathcal{E}(p,e)$, falls keine Regel außen producing ist, dann ist der obere Teil des Zwischenergebnisses leer.

Zum Schluß vergleichen wir Komposition und Deforestation.

Satz 4.23 Für alle $(p,e) \in (ndc\text{-}MAC; lin\text{-}1f\text{-}ndr\text{-}TOP)$ gilt:
$\mathcal{E}(Com(p,e)) \geq \mathcal{E}(Def(p,e))$.

Beweis: Die Komposition benötigt nach Lemma 4.19 $r^{f_0,t_e} + \sum_{i=1}^{n_1} r^{g,t_i}$ Reduktionsschritte, Deforestation laut Lemma 4.20 aber $r^{f_0,t_e} + r^{g,t_z}_{(2b)}$ Schritte. Weil die für die Kontextparameter eingesetzten Bäume im unteren Teil des Zwischenergebnisses auftreten, gilt $\sum_{i=1}^{n_1} r^{g,t_i} \leq r^{g,t_z}_{(2b)}$, womit der Satz bewiesen ist. □

Die Komposition spart also gegenüber Deforestation die Schritte ein, die zur Verarbeitung der Konstruktoren im unteren Teil des Zwischenergebnisses benötigt werden, welche nicht aus den Kontext-Eingabebäumen stammen.

Wir spezifizieren die $\geq$-Beziehung durch folgende Fallunterscheidung bezüglich der Regeln von M_1 noch etwas genauer.

- $\mathcal{E}(Com(p,e)) > \mathcal{E}(Def(p,e))$ ist nicht möglich, da es immer einen Eingabebaum gibt, für den die Gleichheit gilt. Dies kann z.B. ein Baum sein, der nur aus einem nullstelligen Symbol besteht. Die dabei angewandte Regel kann keinen Funktionsaufruf enthalten, weswegen im unteren Teil des Zwischenergebnisses keine zusätzlichen Konstruktoren produziert werden.
- $\mathcal{E}(Com(p,e)) \geq \mathcal{E}(Def(p,e))$, falls einige Regeln innen producing sind oder einige der Regeln von Funktionen, die in Kontextparametern auftreten, außen producing sind.
- $\mathcal{E}(Com(p,e)) = \mathcal{E}(Def(p,e))$, wenn $occ_{bot}(t_z)$ nur die Bäume, die zu Beginn der Reduktion für die Kontextparameter eingesetzt werden, enthält. Das ist der Fall, wenn keine Regel innen producing ist und
  - entweder alle Regeln des macro tree transducers basic sind, denn dann stehen in den Kontextparametern von Funktionen aus dem Mittelteil einer Regel nur Kontextvariablen
  - oder keine Regel außen producing ist, denn dann erzeugen die Funktionsaufrufe in den Kontextparametern keine Zwischenergebnisse.

Das folgende Korollar faßt die Resultate der Sätze 4.21, 4.22 und 4.23 zusammen.

Korollar 4.24 Für alle $(p,e) \in (ndc\text{-}MAC; lin\text{-}1f\text{-}ndr\text{-}TOP)$ gilt:
$\mathcal{E}(Com(p,e)) \geq \mathcal{E}(Def(p,e)) \geq \mathcal{E}(p,e)$. □

In Abhängigkeit von den Eigenschaften der Regeln des macro tree transducers M_1 können wir hier sechs Fälle unterscheiden. Diese erhalten wir durch Überlagerung der bei den drei Sätzen aufgetretenen Fälle. Wir werden für jeden Fall ein Beispiel angeben, welches durch entsprechende Modifikation von M_1 aus Beispiel 4.1 auf Seite 43 entsteht. Wegen $M_2 \in lin\text{-}TOP$ ist dabei stets $n = 1$. Für z_1 benutzen wir den Baum $t = \alpha$, womit sich $r_\alpha = 1$ ergibt.

- $\mathcal{E}(Com(p,e)) \geq \mathcal{E}(Def(p,e)) > \mathcal{E}(p,e)$, falls alle Regeln außen producing sind und einige Regeln innen producing oder nicht basic sind.

  Beispiel 4.25 Damit alle Regeln außen producing sind, setzen wir die Parameter $n_2 = n_0 = 0$ und lassen die Regeln für γ_2 und γ_0 weg. Damit erhalten wir folgendes Programm.

$$\begin{array}{lcl@{\qquad}lcl} f\ (\delta\ x_1)\ y_1 & = & \delta\ (f\ x_1\ (f\ x_1\ y_1)) & g\ (\delta\ x_1) & = & \sigma\ (g\ x_1) \\ f\ (\gamma_1\ x_1)\ y_1 & = & \gamma_1\ (f\ x_1\ y_1) & g\ (\gamma_1\ x_1) & = & g\ x_1 \\ f\ \alpha\ y_1 & = & \delta\ y_1 & g\ \alpha & = & \alpha \end{array}$$

  Damit werden für den Baum $\delta\gamma_1^{n_1}\alpha$ nach Komposition $2n_1 + 4$ Schritte, nach Deforestation $3n_1 + 5$ Schritte und ohne diese Optimierungen $4n_1 + 7$ Schritte gebraucht, weshalb von einem Verfahren zum nächsten die Zahl der Schritte echt größer wird.

Betrachtet man jedoch den Eingabebaum $\gamma_1^{n_1}\alpha$, so werden nach Komposition $n_1 + 2$ Schritte, nach Deforestation ebenfalls $n_1 + 2$ Schritte und ohne diese Optimierungen $2n_1 + 3$ Schritte gebraucht. In diesem Fall ist die Zahl der Schritte nach Komposition und Deforestation gleich und echt kleiner als bei den ursprünglichen tree transducern. □

- $\mathcal{E}(Com(p,e)) = \mathcal{E}(Def(p,e)) > \mathcal{E}(p,e)$, falls alle Regeln außen producing und basic und nicht innen producing sind.

Beispiel 4.26 Damit alle Regeln außen producing sind, setzen wir die Parameter $n_2 = n_0 = 0$ und lassen die Regeln für γ_2 und γ_0 weg. Außerdem streichen wir die (f,δ)-Regel, weil diese nicht basic ist. Damit erhalten wir folgendes Programm.

$$\begin{array}{lcllcl} f\ (\gamma_1\ x_1)\ y_1 & = & \gamma_1\ (f\ x_1\ y_1) & g\ (\delta\ x_1) & = & \sigma\ (g\ x_1) \\ f\ \alpha\ y_1 & = & \delta\ y_1 & g\ (\gamma_1\ x_1) & = & g\ x_1 \\ & & & g\ \alpha & = & \alpha \end{array}$$

Hier werden für den Baum $\gamma_1^{n_1}\alpha$ nach Komposition $n_1{+}2$ Schritte, nach Deforestation ebenfalls $n_1 + 2$ Schritte und ohne diese Optimierungen $2n_1 + 3$ Schritte gebraucht, weshalb für alle $n_1 \in I\!N$ und damit für alle möglichen Eingabebäume die Zahl der Schritte nach Komposition und Deforestation gleich und echt kleiner als bei den ursprünglichen tree transducern ist. □

- $\mathcal{E}(Com(p,e)) \geq \mathcal{E}(Def(p,e)) \geq \mathcal{E}(p,e)$, falls einige Regeln außen producing und einige Regeln innen producing sind oder einige Regeln von Funktionen, die in Kontextparametern bei nicht-basic-Regeln auftreten, außen producing sind.

Beispiel 4.27 Wir lassen die (f,δ)-Regel weg, weil diese nicht basic ist und einige Regeln von f außen producing sind. Außerdem streichen wir in der (f,α)-Regel das δ-Symbol. Damit erhalten wir folgendes Programm.

$$\begin{array}{lcllcl} f\ (\gamma_1\ x_1)\ y_1 & = & \gamma_1\ (f\ x_1\ y_1) & g\ (\gamma_1\ x_1) & = & g\ x_1 \\ f\ (\gamma_2\ x_1)\ y_1 & = & f\ x_1\ (\gamma_2\ y_1) & g\ (\gamma_2\ x_1) & = & g\ x_1 \\ f\ (\gamma_0\ x_1)\ y_1 & = & f\ x_1\ y_1 & g\ \alpha & = & \alpha \\ f\ \alpha\ y_1 & = & y_1 & & & \end{array}$$

Der Eingabebaum sei auf beliebige Weise aus n_1 γ_1-Symbolen, n_2 γ_2-Symbolen und n_0 γ_0-Symbolen aufgebaut und schließe mit α ab. Damit werden nach Komposition $n_1{+}n_2{+}n_0{+}2$ Schritte, nach Deforestation $n_1{+}2n_2{+}n_0{+}2$ Schritte und ohne diese Optimierungen $2n_1 + 2n_2 + n_0 + 2$ Schritte gebraucht. Je nachdem, ob die Parameter n_1 und n_2 größer oder gleich 0 sind, sind die Zahlen der Reduktionsschritte zwischen zwei Verfahren kleiner oder gleich. □

- $\mathcal{E}(Com(p,e)) = \mathcal{E}(Def(p,e)) \geq \mathcal{E}(p,e)$, falls einige Regeln außen producing und alle Regeln basic sind und keine Regel innen producing ist.

Beispiel 4.28 Damit keine Regel innen producing ist, setzen wir den Parameter $n_2 = 0$ und lassen die Regel für γ_2 weg. Außerdem streichen wir die (f,δ)-Regel, weil diese nicht basic ist. Schließlich entfernen wir in der (f,α)-Regel das δ-Symbol. Damit erhalten wir folgendes Programm.

$$\begin{array}{lcllcl} f\ (\gamma_1\ x_1)\ y_1 & = & \gamma_1\ (f\ x_1\ y_1) & g\ (\delta\ x_1) & = & \sigma\ (g\ x_1) \\ f\ (\gamma_0\ x_1)\ y_1 & = & f\ x_1\ y_1 & g\ (\gamma_1\ x_1) & = & g\ x_1 \\ f\ \alpha\ y_1 & = & y_1 & g\ \alpha & = & \alpha \end{array}$$

Der Eingabebaum sei auf beliebige Weise aus n_1 γ_1-Symbolen und n_0 γ_0-Symbolen aufgebaut und schließe mit α ab. Damit werden nach Komposition n_1+n_0+2 Schritte, nach Deforestation ebenfalls n_1+n_0+2 Schritte und ohne diese Optimierungen $2n_1+n_0+2$ Schritte gebraucht. Je nachdem, ob der Parameter n_1 größer oder gleich 0 ist, sind die Zahlen der Reduktionsschritte zwischen einem der Optimierungsverfahren und den ursprünglichen tree transducern kleiner oder gleich. $\square$

- $\mathcal{E}(Com(p,e)) \geq \mathcal{E}(Def(p,e)) = \mathcal{E}(p,e)$, falls keine Regel außen producing ist und einige Regeln innen producing sind.

Beispiel 4.29 Damit keine Regel außen producing ist, setzen wir den Parameter $n_1 = 0$ und lassen die Regel für γ_1 weg. Außerdem streichen wir die (f,δ)-Regel, weil diese ebenfalls außen producing ist. Schließlich müssen wir in der (f,α)-Regel das δ-Symbol streichen. Damit erhalten wir folgendes Programm.

$$\begin{array}{lcl@{\qquad}lcl} f\ (\gamma_2\ x_1)\ y_1 & = & f\ x_1\ (\gamma_2\ y_1) & g\ (\gamma_2\ x_1) & = & g\ x_1 \\ f\ (\gamma_0\ x_1)\ y_1 & = & f\ x_1\ y_1 & g\ \alpha & = & \alpha \\ f\ \alpha\ y_1 & = & y_1 & & & \end{array}$$

Der Eingabebaum sei auf beliebige Weise aus n_2 γ_2-Symbolen und n_0 γ_0-Symbolen aufgebaut und schließe mit α ab. Damit werden nach Komposition $n_2 + n_0 + 2$ Schritte, nach Deforestation $2n_2+n_0+2$ Schritte und ohne diese Verfahren ebenfalls $2n_2+n_0+2$ Schritte gebraucht. Je nachdem, ob der Parameter n_2 größer oder gleich 0 ist, sind die Zahlen der Reduktionsschritte zwischen Komposition und Deforestation kleiner oder gleich. $\square$

- $\mathcal{E}(Com(p,e)) = \mathcal{E}(Def(p,e)) = \mathcal{E}(p,e)$, falls keine Regel producing ist.

Beispiel 4.30 Damit keine Regel producing ist, setzen wir die Parameter $n_1 = n_2 = 0$ und lassen die Regeln für γ_1, γ_2 und δ weg. Außerdem müssen wir in der (f,α)-Regel das δ-Symbol streichen. Damit erhalten wir folgendes Programm.

$$\begin{array}{lcl@{\qquad}lcl} f\ (\gamma_0\ x_1)\ y_1 & = & f\ x_1\ y_1 & g\ \alpha & = & \alpha \\ f\ \alpha\ y_1 & = & y_1 & & & \end{array}$$

Für den Baum $\gamma_0^{n_0}\alpha$ werden nach Komposition, Deforestation und ohne diese Transformationen n_0+2 Schritte gebraucht, weshalb bei allen möglichen Eingabebäumen die Zahl der Reduktionsschritte in allen drei Fällen gleich ist, da nirgends Zwischenergebnisse erzeugt werden. $\square$

Wir können für die Kompositionsklasse $(ndc\text{-}MAC; lin\text{-}1f\text{-}ndr\text{-}TOP)$ also zusammenfassend feststellen, daß die Durchführung der Komposition empfehlenswert und der Anwendung von Deforestation vorzuziehen ist.

4.3.3 Die Kompositionsklasse $(prod\text{-}ndc\text{-}MAC; nlin\text{-}1f\text{-}ndr\text{-}TOP)$

Im folgenden sei $p \in P$ ein Programm, welches $M_1 = (F_{M_1}, C^i_{M_1}, C^o_{M_1}, R_{M_1}) \in prod\text{-}ndc\text{-}MAC$ und $M_2 = (\{g^{(1)}\}, C^i_{M_2}, C^o_{M_2}, R_{M_2}) \in nlin\text{-}1f\text{-}ndr\text{-}TOP$ enthält. Wir betrachten den Kompositionsausdruck $e = g\ (f_0\ z_0\ z_1 \ldots z_{n_1})$ mit $f_0^{(1+n_1)} \in F_{M_1}$ und $n_1 \geq 0$. Weiterhin seien $t_e \in T_{C^i_{M_1}}$ und $t_1, \ldots, t_{n_1} \in T_{C^o_{M_1}}$ die Bäume, die wir für z_0 bzw. $z_1, \ldots, z_{n_1}$ einsetzen, und $t_z = nf(\Rightarrow_{M_1}, (f_0\ t_e\ t_1 \ldots t_{n_1}))$ das durch M_1 erzeugte Zwischenergebnis.

Zuerst untersuchen wir die Komposition. Aus dem Kompositionsausdruck entstehe nach Transformation 2.25 der neue macro tree transducer $M = (F_M, C^i_M, C^o_M, R_M)$ und der neue Ausdruck $(\overline{f_0 g}\ z_0\ (g\ z_1) \ldots (g\ z_{n_1}))$ mit $\overline{f_0 g}^{(1+n_1)} \in F_M$. Außerdem sei $f \in F_{M_1}$ und $\overline{fg} \in F_M$.

Wir versuchen, die nach der Komposition benötigten Reduktionsschritte in Abhängigkeit von den ursprünglichen Reduktionsschritten anzugeben, um sie vergleichen zu können. Weil $M_1 \in$ *prod-ndc-MAC*, läßt sich jedem Besuch ein anderes Vorkommen im Zwischenergebnis zuordnen. Dadurch ist es möglich, die Zahl der Besuche eines Vorkommens im Eingabebaum durch das Kompositionsergebnis mit Hilfe der Besuchsanzahl der zugehörigen Bilder im Zwischenergebnis durch den top-down tree transducer auszudrücken.

Lemma 4.31 Für alle $o_e \in occ(t_e)$ gilt:

$$m_{\overline{fg}}^{\overline{f_0 g}, t_e}(o_e) \leq \sum_{o_z \in bild_f^{f_0, t_e}(o_e)} m^{g, t_z}(o_z)$$

Beweis: Wir beweisen das Lemma durch Induktion über alle Pfade in t_e:

Induktionsanfang ($o_e = \varepsilon$):

Für die linke Seite der Gleichung gilt:

$$m_{\overline{fg}}^{\overline{f_0 g}, t_e}(\varepsilon) = \begin{cases} 1 & \text{falls } \overline{f_0 g} = \overline{fg} \\ 0 & \text{falls } \overline{f_0 g} \neq \overline{fg} \end{cases}$$

Die rechte Seite berechnen wir in mehreren Schritten:

$$bild_f^{f_0, t_e}(\varepsilon) = \begin{cases} \{\varepsilon\} & \text{falls } f_0 = f \\ \emptyset & \text{falls } f_0 \neq f \end{cases}$$

$$m^{g, t_z}(\varepsilon) = 1$$

$$\sum_{o_z \in bild_f^{f_0, t_e}(\varepsilon)} m^{g, t_z}(o_z) = \begin{cases} 1 & \text{falls } f_0 = f \\ 0 & \text{falls } f_0 \neq f \end{cases}$$

Weil $\overline{f_0 g} = \overline{fg}$ genau dann gilt, wenn $f_0 = f$, sind die Ausdrücke auf der linken und rechten Seite gleich, weswegen die $\leq$-Beziehung gilt.

Induktionsschritt: Nun sei $label(o_e, t_e) = c^{(k)}$, $k > 0$, $c \in C_{M_1}^i$, $i \in [k]$, und als Induktionsvoraussetzung gelte für o_e die zu beweisende Beziehung. Wir gehen von o_e zum i-ten Nachfolger. Da wegen $M_1 \in$ *ndc-MAC*, $M_2 \in 1f$-*ndr-TOP* und daher wegen Lemma 4.8 auch M nondeleting in den Kontextparametern ist, ergibt sich mit Hilfe von Lemma 3.7 folgender rekursiver Ansatz.

$$m_{\overline{f'g}}^{\overline{f_0 g}, t_e}(o_e.i) = \sum_{\overline{fg} \in F_M} \left(m_{\overline{fg}}^{\overline{f_0 g}, t_e}(o_e) \ \cdot\ n(\overline{f'g}, i, \overline{fg}, c) \right)$$

Wegen $M_2 \in 1f$-*TOP* gehört zu jedem $f \in F_{M_1}$ genau ein $\overline{fg} \in F_M$, weshalb die Summenbildung über die Funktionen von M_1 möglich ist. Unter Verwendung der Induktionsvoraussetzung ergibt sich

$$\leq \sum_{f \in F_{M_1}} \left(\left(\sum_{o_z \in bild_f^{f_0, t_e}(o_e)} m^{g, t_z}(o_z) \right) \cdot n(\overline{f'g}, i, \overline{fg}, c) \right)$$

Weil die rechte Seite der $(\overline{fg}, c)$-Regel durch Übersetzung der (f, c)-Regel mit g entsteht, wird ein Aufruf der Funktion $\overline{f'g}$ auf den i-ten Nachfolger von c genau dann erzeugt, wenn während dieser Übersetzung ein Vorkommen besucht wird, das mit f' beschriftetet und dessen erster Nachfolger x_i ist. Dies rechtfertigt folgende Umformung:

$$= \sum_{f \in F_{M_1}} \left(\left(\sum_{o_z \in bild_f^{f_0,te}(o_e)} m^{g,t_z}(o_z) \right) \cdot \left(\sum_{\substack{o \in occ(rhs(f,c)) \\ label(o,rhs(f,c))=f' \\ label(o.1,rhs(f,c))=x_i}} m^{g,rhs(f,c)}(o) \right) \right)$$

Durch Ausmultiplizieren der beiden Summen erhält man

$$= \sum_{f \in F_{M_1}} \sum_{\substack{o_z \in bild_f^{f_0,te}(o_e) \\ o \in occ(rhs(f,c)) \\ label(o,rhs(f,c))=f' \\ label(o.1,rhs(f,c))=x_i}} \left(m^{g,t_z}(o_z) \cdot m^{g,rhs(f,c)}(o) \right)$$

Im nächsten Schritt ersetzen wir $rhs(f, c)$ in $m^{g,rhs(f,c)}(o)$ durch $subtree(o_z, t_z)$. Das ist möglich, weil o_z durch Übersetzung von o_e mit f entstanden ist, wobei die (f, c)-Regel angewendet wurde. Dadurch treten alle Konstruktoren, die sich in $rhs(f, c)$ auf dem Pfad von der Wurzel zum Vorkommen o befinden, in t_z auf dem Pfad von o_z zu einem Vorkommen $o_z.o'$ in der gleichen Reihenfolge auf. o' ist dabei ein Vorkommen in $subtree(o_z, t_z)$, welches bei der Übersetzung des i-ten Nachfolgers von c durch f' entsteht, weshalb $o' \in bild_{f'}^{f,subtree(o_e,t_e)}(i)$ gilt.

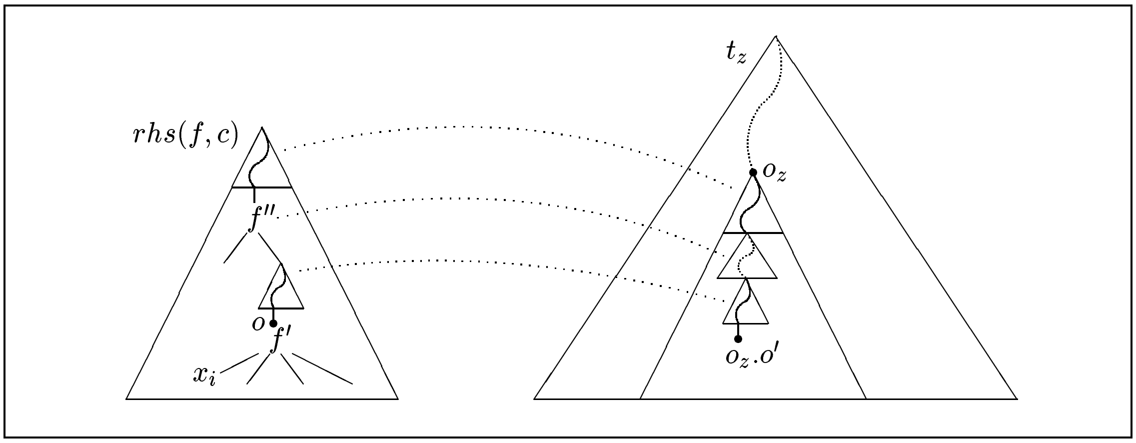

Abbildung 4.9: Auftreten der Konstruktoren einer rechten Seite im Zwischenergebnis

Wenn sich der Funktionsaufruf f' im unteren Teil der rechten Seite befindet, dann liegen auf dem Pfad zum Vorkommen o außer Konstruktoren noch Funktionssymbole, in deren Kontextparametern sich f' befindet. Wegen $M_1 \in$ *prod-MAC* erzeugen diese Aufrufe im Zwischenergebnis weitere Konstruktoren, die zwischen den Konstruktoren aus der rechten Seite eingeschoben werden. Dieser Fall ist in Abbildung 4.9 dargestellt. Dadurch kann wegen $M_2 \in$ *nlin-MAC* die Zahl der Besuche von o' in $subtree(o_z, t_z)$ höher sein als die Besuchsanzahl von o in $rhs(f, c)$, weshalb wir folgende Abschätzung vornehmen.

$$\leq \sum_{f \in F_{M_1}} \sum_{\substack{o_z \in bild_f^{f_0,te}(o_e) \\ o' \in bild_{f'}^{f,subtree(o_e,te)}(i)}} \left(m^{g,t_z}(o_z) \cdot m^{g,subtree(o_z,t_z)}(o') \right)$$

Daraus erhalten wir die Besuchsanzahl des Vorkommens $o_z.o'$.

$$= \sum_{f \in F_{M_1}} \sum_{\substack{o_z \in bild_f^{f_0,t_e}(o_e) \\ o' \in bild_{f'}^{f,subtree(o_e,t_e)}(i)}} m^{g,t_z}(\underbrace{o_z.o'}_{o'_z})$$

Nun kann der Pfad $o_z.o'$ zu einem neuen Vorkommen o'_z zusammengezogen werden. Der in der rechten Seite der (f,c)-Regel im Vorkommen o stehende Funktionsaufruf von f' auf x_i veranlaßt einen Besuch von $o_e.i$ im Eingabebaum durch f'. Das dabei in t_z entstehende Bildvorkommen ist $o_z.o'$, also o'_z. Betrachtet man die (f,c)-Regeln für alle $f \in F_{M_1}$, so entsteht auf diese Weise die komplette Bildmenge $bild_{f'}^{f_0,t_e}(o_e.i)$, weil die Bildmengen eines Vorkommens für verschiedene Funktionen wegen der producing-Eigenschaft disjunkt sind.

$$= \sum_{o'_z \in bild_{f'}^{f_0,t_e}(o_e.i)} m^{g,t_z}(o'_z)$$

Weil aus der Gültigkeit der zu beweisenden Aussage für ein Vorkommen $o_e \in occ(t_e)$ auf die Gültigkeit für dessen Nachfolger geschlossen werden konnte, gilt die Aussage für alle $o_e \in occ(t_e)$. □

Aus Lemma 4.31 läßt sich ein analoger Zusammenhang für die Gesamtanzahl der Besuche herleiten.

Lemma 4.32 Für alle $o_e \in occ(t_e)$ gilt:

$$m^{\overline{fog},t_e}(o_e) \leq \sum_{o_z \in bild^{f_0,t_e}(o_e)} m^{g,t_z}(o_z)$$

Beweis:

$$\begin{aligned}
m^{\overline{fog},t_e}(o_e) &= \sum_{\overline{fg} \in F_M} m_{\overline{fg}}^{\overline{fog},t_e}(o_e) && \text{(Definition 2.15)} \\
&\leq \sum_{f \in F_{M_1}} \left(\sum_{o_z \in bild_f^{f_0,t_e}(o_e)} m^{g,t_z}(o_z) \right) && \text{(Lemma 4.31)} \\
&= \sum_{o_z \in bild^{f_0,t_e}(o_e)} m^{g,t_z}(o_z) && \text{(Definition 2.16, Lemma 2.17)}
\end{aligned}$$

□

Nun können wir die Zahl der Reduktionsschritte, die das Kompositionsergebnis benötigt, angeben.

Lemma 4.33 $r^{(\overline{fog}\ t_e\ t_1 \ldots t_{n_1})} \leq r^{g,t_z}$

Beweis: Wir berechnen die Anzahl der Reduktionsschritte für das Kompositionsergebnis mit Hilfe von Lemma 2.20. Dabei müssen wir auch die Reduktionsschritte berücksichtigen, die zur Übersetzung der Bäume $t_1, \ldots, t_{n_1}$ nötig sind.

$$r^{(\overline{fog}\ t_e\ t_1 \ldots t_{n_1})} = \sum_{o_e \in occ(t_e)} m^{\overline{fog},t_e}(o_e) + \sum_{i=1}^{n_1} r^{(g\ t_i)}$$

Unter Verwendung von Lemma 4.32 und Definition 2.19 erhalten wir die Abschätzung

$$\leq \sum_{o_e \in occ(t_e)} \left(\sum_{o_z \in bild^{f_0,t_e}(o_e)} m^{g,t_z}(o_z) \right) + \sum_{i=1}^{n_1} r^{g,t_i}$$

Nun versuchen wir, die Bildmengen aller Vorkommen aus $occ(t_e)$ zu vereinigen. Da die Voraussetzung von Lemma 2.18 nicht erfüllt ist, kann ein Vorkommen $o_z \in occ(t_z)$ in mehreren dieser Bildmengen enthalten sein. Dies ist genau dann der Fall, wenn eine Funktion $f \in F_{M_1}$, deren Regel nicht außen producing ist, an einem mit dem Eingabesymbol $c \in C^i_{M_1}$ beschrifteten Vorkommen $o_e \in occ(t_e)$ angewendet wird. Der dann zwangsläufig im Mittelteil der (f,c)-Regel enthaltene Aufruf einer Funktion $f' \in F_{M_1}$ auf den i-ten Nachfolger von o_e erzeugt das gleiche Bild wie dieser Besuch. Ein im inneren Teil dieser Regel eventuell vorkommender weiterer Aufruf einer Funktion $f'' \in F_{M_1}$ erzeugt jedoch ein anderes Bildvorkommen $o''_z \in occ(t_z)$, weil sich zwischen o_z und o''_z noch Konstruktoren befinden. Sie entstehen durch Auswertung des darüberliegenden Funktionsaufrufs f'. Dabei wird auf jeden Fall eine Regel, welche im Mittelteil eine Kontextvariable enthält und zur Aktivierung des inneren Funktionsaufrufs benötigt wird, angewendet. Solche Regeln sind außen producing, denn ansonsten bestünden sie nur aus dieser Kontextvariablen und verstießen damit gegen die producing-Bedingung. Diese Situation zeigt Abbildung 4.10.

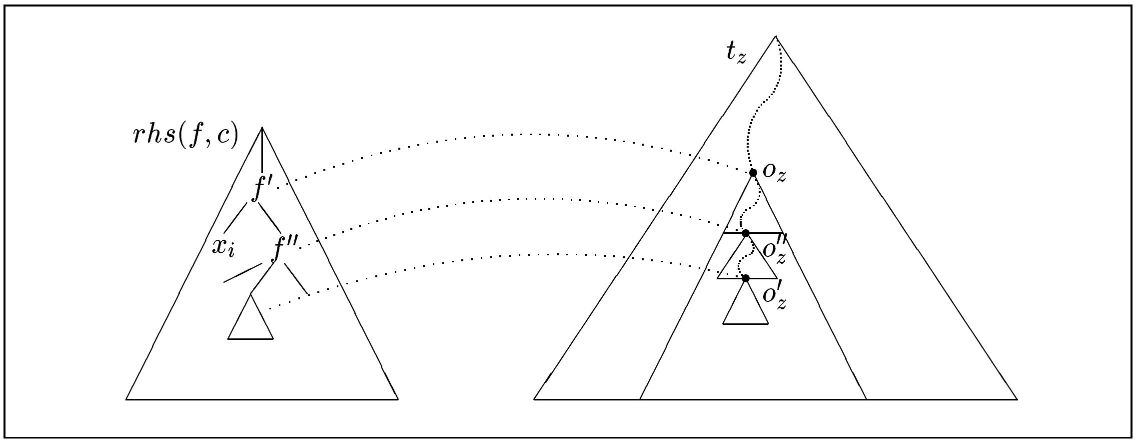

Abbildung 4.10: Bildvorkommen und Konstruktoren im Zwischenergebnis bei Anwendung einer Regel, die nicht außen producing, aber innen producing ist

Weil die (f,c)-Regel nicht außen producing ist, muß im inneren Teil von $rhs(f,c)$ mindestens ein Konstruktorsymbol vorhanden sein, damit die Regel producing wird. Wegen $M_1 \in ndc\text{-}MAC$ erscheint dieses Symbol im Zwischenergebnis. Der Pfad $o'_z \in occ(t_z)$ zu diesem Konstruktor ist in keiner Bildmenge enthalten, weil sich darüber wieder die Konstruktoren aus dem oberen Teil einer Regel befinden, die eine Kontextvariable im Mittelteil enthält. Darüber hinaus ist der Pfad o'_z eine Verlängerung von o_z. Auch dies ist in Abbildung 4.10 dargestellt. Wegen $M_2 \in nlin\text{-}ndr\text{-}TOP$ wird o'_z also mindestens so oft von g besucht wie o_z. Wenn wir nun anstelle der Besuche mehrfach vorkommender Elemente, die bei der Vereinigung aller Bildmengen verlorengehen, die Besuche dieser Vorkommen benutzen, kann die Summe der Besuche all dieser Vorkommen größer werden, auf keinen Fall jedoch kleiner. Die Vorkommen, welche unter Verwendung von t_1 bis t_{n_1} entstehen, sind ebenfalls in keiner Bildmenge enthalten, weil diese auch durch Regeln mit Kontextvariablen im Mittelteil erzeugt werden. Deren Besuchsanzahl kann gegenüber der Übersetzung im Vorfeld auch größer werden. Damit können wir folgende Abschätzung vornehmen.

$$\leq \sum_{\substack{o_z \in \bigcup\limits_{o_e \in occ(t_e)} bild^{f_0,t_e}(o_e)}} m^{g,t_z}(o_z) \quad + \sum_{\substack{o_z \notin \bigcup\limits_{o_e \in occ(t_e)} bild^{f_0,t_e}(o_e)}} m^{g,t_z}(o_z)$$

Die beiden Summen fassen wir nun zusammen.

$$= \sum_{o_z \in occ(t_z)} m^{g_0,t_z}(o_z)$$

Durch nochmalige Anwendung von Lemma 2.20 erhalten wir schließlich

$$= r^{g,t_z}$$

□

Nach diesen Vorüberlegungen können wir die durch die Komposition erzielte Effizienzsteigerung angeben.

Satz 4.34 Für alle $(p,e) \in (prod\text{-}ndc\text{-}MAC; nlin\text{-}1f\text{-}ndr\text{-}TOP)$ gilt:
$\mathcal{E}(Com(p,e)) > \mathcal{E}(p,e)$.

Beweis: Die ursprünglichen tree transducer benötigen zusammen $r^{f_0,t_e} + r^{g,t_z}$ Schritte, das Kompositionsergebnis laut Lemma 4.33 aber weniger als r^{g,t_z} Schritte. Wegen $r^{f_0,t_e} \geq 1$ gilt folglich $r^{(\overline{f_0g}\; t_e\; t_1 \ldots\, t_{n_1})} < r^{f_0,t_e} + r^{g,t_z}$, womit der Satz bewiesen ist. □

Es werden also die Reduktionsschritte eingespart, welche zur Erzeugung des Zwischenergebnisses nötig sind. Eine weitere Einsparung wird dadurch erzielt, daß die Bäume, die zu Beginn der Reduktion für die Kontextparameter eingesetzt werden, nur einmal reduziert werden. Eine dritte Einsparung betrifft alle Schritte, die zur Übersetzung der Vorkommen im Zwischenergebnis erforderlich sind, welche in keiner Bildmenge enthalten sind, deren Beschriftungen nicht aus den eben genannten Bäumen stammen sowie die nicht als Ersatz für die bei der Vereinigung verlorengegangenen Vorkommen benutzt wurden. Ein letzter möglicher Einspareffekt stammt von den Abschätzungen, die bei der Verwendung von Lemma 4.32 und bei der soeben erwähnten Ersetzung vorgenommen worden sind.

Jetzt vergleichen wir die Komposition und Deforestation.

Satz 4.35 Für alle $(p,e) \in (prod\text{-}ndc\text{-}MAC; nlin\text{-}1f\text{-}ndr\text{-}TOP)$ gilt:
$\mathcal{E}(Com(p,e)) \geq \mathcal{E}(Def(p,e))$.

Beweis: Für das Kompositionsergebnis gilt nach Lemma 2.20, 4.32 und Definition 2.19

$$r^{(\overline{f_0g}\; t_e\; t_1 \ldots\, t_{n_1})} \leq \sum_{o_e \in occ(t_e)} \left(\sum_{o_z \in bild^{f_0,t_e}(o_e)} m^{g,t_z}(o_z) \right) + \sum_{i=1}^{n_1} r^{g,t_i}$$

Nach Transformation 2.28 entsteht durch Deforestation aus dem Kompositionsausdruck der Ausdruck $(f_0g\; z_0\; z_1 \ldots z_{n_1})$, wobei ein neues Programm $p' \in P^+$ erzeugt wird. Es gilt $f_0g^{(1+n_1)} \in F_{p'}$. Wir vergleichen die eben angegebene Formel für $r^{\overline{f_0g},t_e}$ mit der Formel für r^{f_0g,t_e} aus Lemma 4.16. Dabei gilt im Teil für Phase (2) jeweils $c^{(k)} = label(o_e, t_e)$ mit

$k \in I\!N$.

$$
\begin{aligned}
r^{f_0 g, t_e} &\overset{(1)}{=} \sum_{o_e \in occ(t_e)} \left(\sum_{\substack{o_z \in bild^{f_0,t_e}(o_e) \\ \cap\, occ_{top}(t_z)}} m^{g,t_z}(o_z) \right) \overset{(3)}{+} r^{g,t_z}_{(2b)} \\
&\overset{(2)}{+} \sum_{o_e \in occ(t_e)} \sum_{f \in F_{M_1}} \sum_{\substack{f' \in F_{M_1} \\ i \in [k]}} \sum_{\substack{o_z \in bild_f^{f_0,t_e}(o_e) \\ \cap\, occ_{top}(t_z)}} m^{g,t_z}(o_z) \cdot n_{bot}(f', i, fg, c) \cdot r^{f', subtree(o_e.i, t_e)}
\end{aligned}
$$

Die Zahl der Reduktionsschritte aus den Phasen (1) und (3) bei Deforestation ist zusammen schon gleich oder größer als bei der Komposition, weil dort nur die Besuche der Bildvorkommen und die Reduktionsschritte für die am Beginn für die Kontextvariablen eingesetzten Bäume zählen, während bei Deforestation neben den Bildvorkommen aus $occ_{top}(t_z)$ und $occ_{bot}(t_z)$ sämtliche übrigen Vorkommen des unteren Teils betrachtet werden. Darin sind auch die Vorkommen enthalten, die aus den am Anfang eingesetzten Bäumen stammen und mehrfach besucht werden können. Außerdem kommt gegebenenfalls noch der Formelteil für die Phase (2) hinzu, der bei der Komposition überhaupt nicht dabei ist. □

Zum Schluß vergleichen wir $\mathcal{E}(Def(p,e))$ und $\mathcal{E}(p,e)$. Beim Vergleich der Schritte aus Lemma 4.16 mit $r^{f_0,t_e} + r^{g,t_z}$ stellen wir fest, daß durch Deforestation die Schritte aus Phase (1a) sowie die Schritte aus Phase (2a), die zur Reduktion von Vorkommen aus $occ(t_z)$ benötigt werden, die in keiner Bildmenge sind, eingespart werden. Andererseits können in Phase (2) Schritte aus Phase (1b) vervielfacht werden, wenn $n_{bot}(f', i, fg, c) > n_{bot}(f', i, f, c)$. Dadurch kommt es zu Effizienzverlusten, wenn dieser Effekt stärker als die Einsparung ist, wie das folgende Beispiel zeigt.

Beispiel 4.36 Wir verwenden eine modifizierte Version des Beispiels 4.1 von Seite 43. Damit M_1 producing wird, setzen wir $n_0 = 0$ und streichen die (f, γ_0)-Regel. Der Eingabebaum sei $\delta\gamma_1^{n_1}\gamma_2^{n_2}\alpha$, und z_1 belegen wir mit α.

Wir betrachten zunächst den Fall $n = 2$. Dann werden nach Deforestation $8n_1 + 16n_2 + 17$ Schritte benötigt, während die beiden tree transducer ursprünglich $8n_1 + 14n_2 + 18$ Schritte brauchten. Für die γ_1-Symbole heben sich der Einspareffekt und der Effizienzverlust genau auf, während für die γ_2-Symbole nichts gespart wird, weswegen es zu einem Effizienzverlust kommt, wenn $n_2 > 0$.

Nun untersuchen wir noch den Fall $n = 3$. Dann werden nach Deforestation $15n_1 + 42n_2 + 43$ Schritte benötigt, während die beiden tree transducer ursprünglich $14n_1 + 38n_2 + 43$ Schritte brauchten. Hier ist sogar schon für die γ_1-Symbole der Effizienzverlust größer als der Einspareffekt, weswegen es auf jeden Fall zu einem Effizienzverlust kommt. □

Der eben beschriebene Effizienzverlust kann nur auftreten, wenn Funktionsaufrufe im unteren Teil einer Regel von M_1 bei der Erzeugung der neuen Regeln vervielfacht werden. Wenn jedoch alle Regeln basic sind, ist dies nicht möglich. Dann gilt stets $\mathcal{E}(Def(p,e)) > \mathcal{E}(p,e)$, weil jeder Eingabebaum mit einem nullstelligen Symbol endet, für welches die Regel außen producing sein muß. Deshalb wird immer ein Effizienzgewinn erzielt, wie folgendes Beispiel belegt.

Beispiel 4.37 Wir modifizieren das Beispiel 4.1 so, daß M_1 alle Regeln basic und producing werden, indem wir $n_0 = 0$ setzen und die (f, δ)-Regel sowie die (f, γ_0)-Regel streichen. Der Eingabebaum sei $\gamma_1^{n_1}\gamma_2^{n_2}\alpha$, und z_1 belegen wir mit α.

Wir betrachten den Fall $n = 2$. Dann werden nach Deforestation $n_1 + 3n_2 + 3$ Schritte benötigt, während die beiden tree transducer ursprünglich $2n_1 + 3n_2 + 4$ Schritte brauchten. Weil der konstante Teil bei Deforestation immer kleiner als bei den tree transducern ist, wird unabhängig von der konkreten Gestalt des Eingabebaums mindestens ein Schritt gespart. □

Zusammenfassend können wir für die Kompositionsklasse $(prod\text{-}ndc\text{-}MAC; nlin\text{-}1f\text{-}ndr\text{-}TOP)$ also feststellen, daß die Komposition wegen des garantierten Effizienzgewinns durchgeführt werden sollte, während demgegenüber auf die Anwendung von Deforestation aufgrund des geringeren Effizienzgewinns und des möglichen Effizienzverlustes verzichtet werden sollte.

4.3.4 Die Kompositionsklasse $(nprod\text{-}ndc\text{-}MAC; nlin\text{-}1f\text{-}ndr\text{-}TOP)$

Da wir die Effizienzverluste bei der Kombination der nonproducing-Eigenschaft beim ersten und der Nichtlinearität beim zweiten tree transducer schon mehrfach beobachtet haben, geben wir hier nur noch zwei Beispiele an.

Beispiel 4.38 Das Beispiel aus der Einleitung für den Effizienzverlust im nichtlinearen Fall kann wegen $(nprod\text{-}TOP; nlin\text{-}1f\text{-}ndr\text{-}TOP) \subset (nprod\text{-}ndc\text{-}MAC; nlin\text{-}1f\text{-}ndr\text{-}TOP)$ dazu verwendet werden, um die möglichen Effizienzverluste bei der hier behandelten Klasse zu belegen. □

Beispiel 4.39 Ein weiteres Beispiel ergibt sich durch entsprechende Modifikation von Beispiel 2.25. Wir setzen $n_1 = n_2 = 0$ und lassen die (f, γ_1)-Regel sowie die (f, γ_2)-Regel weg. Der Eingabebaum sei $\delta\gamma_0^{n_0}\alpha$, und z_1 belegen wir mit α.

Wir betrachten den Fall $n = 2$. Dann werden nach Deforestation und Komposition jeweils $4n_0 + 6$ bzw. Schritte $4n_0 + 17$ benötigt, während die beiden tree transducer ursprünglich nur $2n_0 + 18$ Schritte brauchten. Damit kommt es bei beiden Verfahren zu einem Effizienzverlust, sobald der Eingabebaum mindestens ein γ_0-Symbol enthält. □

4.3.5 Ausblick auf die Klasse $(MAC; TOP) - (ndc\text{-}MAC; 1f\text{-}ndr\text{-}TOP)$

Wir wollen an dieser Stelle noch einmal das Beispiel 4.6 von Seite 47 aufgreifen. Dort enthalten die rechten Regelseiten der neu entstandenen Funktionen jeweils nur eine der beiden Kontextvariablen. Wenn darin jedoch beide Kontextvariablen enthalten wären, hieße das, daß infolge der Vervielfachung der Kontextparameter irgendwelche Berechnungen, die in den Kontextparametern einer alten Regeln standen, vervielfacht würden. Dies bedeutete einen Effizienzverlust. Da bei der Übersetzung jedoch nur genau eine der mehreren möglichen neuen Kontextvariablen erzeugt wird, wenn eine Funktion des top-down tree transducers auf eine Kontextvariable stößt, tritt dieser Effekt nicht auf. Der Verlust der nondeleting-Eigenschaft in den Kontextparametern hebt sozusagen diesen Linearitätsverlust genau auf. Deshalb vermuten wir, daß die Resultate für top-down tree transducer mit einer Funktion auch allgemein gelten.

Ftür die generelle Aufhebung der nondeleting-Eigenschaft in den Kontextparametern und im Rekursionsargument ist es jedoch erforderlich, neue Beweistechniken zu entwickeln, da die hier verwendeten Anstätze, welche an der Gestalt der Regeln von tree transducern orientiert waren, nicht mehr verwendbar sind. Dies ist jedoch nicht mehr Gegenstand dieser Arbeit.

Kapitel 5

Ausblick

Wir haben in dieser Arbeit quantitative Vergleiche von Deforestation und Kompositionstechniken für tree transducer durchgeführt, welche als spezielle funktionale Programme angesehen werden können. Dabei benutzten wir als Effizienzmaß die ungewichtete Anzahl der Reduktionsschritte. Weil die Kosten eines solchen Schrittes von der Speichernutzung abhängen, wäre eine Wichtung der Reduktionsschritte entsprechend der dabei erzeugten Kontruktoren hilfreich.

Bei unseren Untersuchungen haben wir uns vorrangig mit macro tree transducern beschäftigt, welche keine Kontextparameter löschen. Bei der Klasse $(MAC; TOP)$ war dabei noch zusätzlich die Beschränkung auf nichtlöschende top-down tree transducer mit einer Funktion erforderlich. In zukünftigen Arbeiten sollten wir versuchen, neue Beweistechniken zu finden, um diese Einschränkungen fallen lassen zu können.

Außerdem möchten wir weitere Verfahren zur Elimination von Zwischenergebnissen in unsere Vergleiche einbeziehen. Eine solche Möglichkeit ist z.B. *Short Cut Deforestation* ([GLPJ93]), welche eine vereinfachte Variante von Deforestation ist, die aufgrund des geringeren Analyse- und Transformationsaufwandes bereits zur effizienten Implementierung funktionaler Sprachen benutzt wird, z.B. für Haskell ([FHP97]).

Ein weitere Möglichkeit ist die Berücksichtigung von Kompositionsresultaten für *attributed tree transducer* ([Fül81]), welche eine Abstraktion von Attributgrammatiken sind ([Knu68]). Solche Kompositionsresultate sind beispielsweise in [Fül81], [Gan83], [Gie88] und [Küh97] zu finden. Für eine eingeschränkte Klasse von macro tree transducern lassen sich diese Methoden zur Komposition verwenden, wie in [Küh98] gezeigt wird. Für die angestrebten Vergleiche halten wir *macro attributed tree transducer* ([KV94]) für das geeignete integrative Konzept.

Literaturverzeichnis

[BD77] R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24:44–67, 1977.

[CF82] B. Courcelle and P. Franchi–Zannettacci. Attribute grammars and recursive program schemes. *Theoret. Comput. Sci.*, 17:163–191, 235–257, 1982.

[Eng80] J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R.V. Book, editor, *Formal language theory: perspectives and open problems*, pages 241–286. New York, Academic Press, 1980.

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. System Sci.*, 31:71–146, 1985.

[FHP97] J.H. Fasel, P. Hudak, and J. Peterson. A gentle introduction to Haskell – Version 1.4. Tutorial, Yale University and University of California, 1997.

[Fül81] Z. Fülöp. On attributed tree transducers. *Acta Cybernet.*, 5:261–279, 1981.

[FV98] Z. Fülöp and H. Vogler. *Syntax-directed semantics — Formal Models Based on Tree Transducers*, volume 56 of *Monographs in Theoretical Computer Science, An EATCS Series*. Springer-Verlag, 1998.

[Gan83] H. Ganzinger. Increasing modularity and language–independency in automatically generated compilers. *Science of Comput. Prog.*, 3:223–278, 1983.

[Gie88] R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Inform.*, 25:355–423, 1988.

[GLPJ93] A. Gill, J. Launchbury, and S.L. Peyton Jones. A short cut to deforestation. In *Functional Programming Languages and Computer Architecture*, pages 223–232, Kopenhagen, 1993. ACM.

[HJ92] G. W. Hamilton and S. B. Jones. Extending deforestation for first order functional programs. In *Functional Programming, Glasgow 1991: Proceedings of the 1991 Workshop, Portree, UK*, pages 134–145. Springer-Verlag, 1992.

[Hud89] Paul Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3):359–411, September 1989.

[Hug90] R. J. M. Hughes. Why functional programming matters. In D. A. Turner, editor, *Research Topics in Functional Programming*, pages 17–42. Addison-Welsey, 1990.

[Knu68] D.E. Knuth. Semantics of context–free languages. *Math. Systems Theory*, 2:127–145, 1968.

[Küh97] A. Kühnemann. *Berechnungsstärken von Teilklassen primitiv-rekursiver Programmschemata.* PhD thesis, Dresden University of Technology, 1997. Shaker Verlag, Aachen.

[Küh98] A. Kühnemann. Benefits of tree transducers for optimizing functional programs. In A. Middeldorp and T. Sato, editors, *FST & TCS'98, Chennai, India*, volume 1530 of *LNCS*, pages 146–157. Springer-Verlag, December 1998.

[Küh99] A. Kühnemann. Comparison of deforestation techniques for functional programs and for tree transducers. In A. Middeldorp and T. Sato, editors, *4th Fuji International Symposium on Functional and Logic Programming – FLOPS'99, Tsukuba, Japan*, LNCS. Springer-Verlag, 1999. to appear.

[KV94] A. Kühnemann and H. Vogler. Synthesized and inherited functions — a new computational model for syntax–directed semantics. *Acta Inform.*, 31:431–477, 1994.

[KV98] A. Kühnemann and H. Vogler. Declarative programming languages. Vorlesungsscript, Dresden University of Technology, Department of Computer Science, 1998.

[Rou70] W. C. Rounds. Mappings and grammars on trees. *Mathematical Systems Theory*, 4(3):257–287, 1970.

[SGJ94] M. H. Sørensen, R. Glück, and N.D. Jones. Towards unifying deforestation, supercompilation, partial evaluation, and GPC. In D. Sannella, editor, *European Symposium on Programming – ESOP'94, Edinburgh, UK*, volume 788 of *LNCS*, pages 485–500. Springer-Verlag, 1994.

[Vog87] H. Vogler. Basic tree transducers. *J. Comput. System Sci.*, 34:87–128, 1987.

[Wad90] P.L. Wadler. Deforestation: transforming programs to eliminate trees. *Theoretical Computer Science*, 73:231–248, 1990.