Evaluation strategies for term rewriting systems

by

Masaki Nakamura

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Professor Kokichi Futatsugi

School of Information Science
Japan Advanced Institute of Science and Technology

March 2002

Abstract

Term rewriting systems are widely used in computer science as a model of computation to relate syntax and semantics. In order to implement term rewriting system we need to use a strategy since there are many reduction sequences from a term in general. A strategy chooses one from such sequences. It is a function that takes a term to be rewritten and returns a term obtained by rewriting from the input term. There are two well-known strategies: innermost strategies (or eager evaluation) and outermost strategies (or lazy evaluation). Innermost strategies can be implemented much more efficiently than outermost ones, while outermost strategies often have a better termination behavior than innermost ones. The evaluation strategy (the E-strategy), which is adopted by the family of OBJ algebraic specification languages, is one of the compromises between them. The E-strategy is more flexible than other fixed order of evaluation because each function symbol can have its own local strategy.

In this thesis we investigate methods to define suitable local strategies for a given term rewriting system. In recent year, context-sensitive rewriting has been proposed by Lucas and studied actively by many researchers. By comparing the E-strategy with context-sensitive rewriting we can obtain some useful properties for the E-strategy by use of properties of context-sensitive rewriting: termination, confluence and so on. We especially focus a shape of evaluated terms because an evaluated term is not always in normal form in the E-strategy. We define a notion of μ-correctness for the E-strategy. An E-strategy is μ-correct if each evaluated term is always in normal form of context-sensitive rewriting, called μ-normal form. We analyze which arguments can be evaluated lazily with keeping μ-correctness. From our correctness analysis we may obtain an E-strategy with better termination behavior. We also define a notion of strictness for arguments of function symbols. An argument is strict if eager evaluation of such argument does not change the termination behavior of reduction of the E-strategy. We analyze behavior of variables in rules of a term rewriting system to find strict arguments. From our strictness analysis of we may obtain an E-strategy with more efficiency in reduction steps. Finally we investigate an extension of the E-strategy, called the on-demand E-strategy. It is known that true lazy evaluation cannot be defined by the ordinary E-strategy. Thus we need a new function for realizing lazy evaluation by the E-strategy. In the on-demand E-strategy we can appoint an argument to be not evaluated until so forced. Evaluation may be forced when the arguments are involved in matching. We formalize the on-demand E-strategy by a pair of an E-strategy map and an on-demand map, which stand for orders of reduction and matching respectively. For the on-demand E-strategy order of matching is important because a target term may be changed while doing on-demand matching. We show some examples that we can treat well owing to an on-demand map. In conclusion we show how to apply our results in this thesis to verification systems, such as the CafeOBJ system.

Acknowledgments

I would like to thank Professor Kokichi Futatsugi for his guidance and encouragement. I wish to express my thanks to Research Assistant Kazuhiro Ogata, Research Assistant Noriki Amano, Associate Professor Takuo Watanabe and the members of Language Design Laboratory for the valuable discussion. I also would like to thank Professor Takuya Katayama, Professor Atsushi Ohori, Professor Toshiki Sakabe for their helpful advises on this work. I wish to express my gratitude for the helpful advice and valuable discussion received from Professor Yoshihito Toyama who had been my advisor during the first half of life in JAIST. I would like to acknowledge all people who help me with this thesis.

Contents

Chapter 1

Introduction

1.1 Term rewriting

Equational reasoning plays an important role in various areas in computer science, such as automated theorem proving, algebraic specifications, functional programming languages and so on. An equational system is a set of equations. We are interested in knowing whether an equation is a logical consequence from the given set. For a set E and terms s and t, an equation $s = t$ is true in E if and only if the equation can be derived from E by the following inference rules:

$$\frac{\text{true}}{t = t}\ (reflexivity) \quad \frac{s = t}{t = s}\ (symmetry) \quad \frac{s = t \quad t = u}{s = u}\ (transitivity)$$

$$\frac{s = t}{f(\ldots, s, \ldots) = f(\ldots, t, \ldots)}\ (congruence) \quad \frac{s = t}{s\theta = t\theta}\ (substitutivity)$$

where f is a function symbol in a signature under consideration and θ is a substitution which instantiate variables in terms. For example, the equation $s(0) + 0 = 0 + s(0)$ is true in the following set E:

$$E = \begin{cases} x + 0 = x \\ x + s(y) = s(x + y) \end{cases}$$

because we can write the following derivation tree:

$$\dfrac{\dfrac{x + 0 = x}{s(0) + 0 = s(0)}\ (subst) \quad \dfrac{\dfrac{\dfrac{\dfrac{x + 0 = x}{0 + 0 = 0}\ (subst)}{0 = 0 + 0}\ (symm)}{s(0) = s(0 + 0)}\ (cong) \quad \dfrac{\dfrac{x + s(y) = s(x + y)}{0 + s(0) = s(0 + 0)}\ (subst)}{s(0 + 0) = 0 + s(0)}\ (symm)}{s(0) = 0 + s(0)}\ (trans)}{s(0) + 0 = 0 + s(0)}\ (trans)$$

It is easy to see that for a given equation we can construct an infinite tree from the equation, especially by the symmetry and transitivity rules, such as

$$\dfrac{\dfrac{\vdots}{s = t}}{\dfrac{t = s}{s = t}} \qquad \dfrac{\dfrac{\dfrac{\vdots}{s = v} \quad \dfrac{\vdots}{v = u}}{s = u} \quad \dfrac{\dfrac{\vdots}{u = w} \quad \dfrac{\vdots}{w = t}}{u = t}}{s = t}$$

Hence, it is difficult to obtain an automatic computation to solve a given equation by directly applying these inference rules.

Term rewriting is one of the methods to obtain flexible computing of equational reasoning. In term rewriting, equations are regarded as directed rules. When we want to know whether an equation $s = t$ is true, reduce the both side terms s and t by the rules and check whether the results are identical with each other.

$$\begin{tikzcd} s \arrow[d] & & t \arrow[d] \\ s' \arrow[r, phantom, "\equiv ?"] & & t' \end{tikzcd}$$

A set R of rules is called a term rewriting system (a TRS). For example, the following R is the result of directing the equations in E.

$$R = \begin{cases} x + 0 \to x \\ x + s(y) \to s(x + y) \end{cases}$$

We can easily prove that the equation $s(0) + 0 = 0 + s(0)$ is true, since $s(0) + 0$ and $0 + s(0)$ are reduced into the same term $s(0)$ by the rules in R, such as $s(0) + 0 \to_R s(0)$ and $0 + s(0) \to_R s(0 + 0) \to_R s(0)$.

Rewriting is undecidable in general , i.e. it is not always that a term is rewritten into a unique term. For example, $(s(0) + 0) + s(0)$ is rewritten into the two terms: $(s(0) + 0) + s(0) \to_R s(0) + s(0)$ by applying the first rule to the first argument and $(s(0) + 0) + s(0) \to_R s((s(0) + 0) + 0)$ by applying the second rule to the whole term. So if we want to implement an equational reasoning by term rewriting, we need a function which takes a term t and returns a term s reduced from t, i.e. $t \to_R \cdots \to_R s$ (Fig. 1.1). In this thesis we call such a function F a strategy.

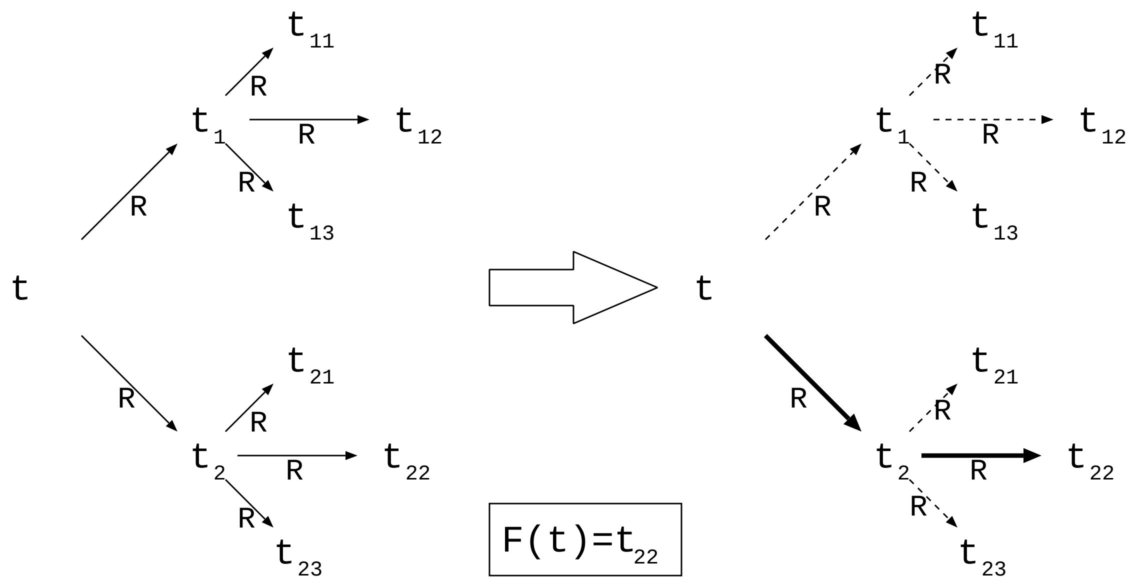

Figure 1.1: A strategy

Most of the functional languages evaluate a term by a fixed strategy, such as the leftmost innermost strategy (or eager evaluation) for the Standard ML[MTHM97], the functional strategy for Haskell[HWA+90] and so on. Those strategies have each merits and demerits. In recent year, strategies whose behaviour can be decided by a user have being watched with interest, such as the evaluation strategy for the OBJ family and Stratego for ELAN [Vis01, Vis99, Vis99a, Vis99b, Vis99c, Dol01]. By using such a strategy we can define a suitable strategy for every program or specification.

1.2 One-step strategies

A strategy may be defined by a one-step strategy usually in term rewriting. A one-step strategy is a function S which takes a term and returns a redex position to which a rule can be applied if it exists or "no redex" if it does not. For $(s(0)+0)+s(0)$ a one-step strategy must return the whole term $(s(0)+0)+s(0)$ or the first argument $s(0)+0$ respectively. For $s(s(0))$ which has no redex a one-step strategy must return the word "no redex". We can obtain a strategy for a one-step strategy as follows:

1. Apply S to an input term t.

2. If it returns a redex position, rewrite it and go to **1** with the result term as an input term, or if "no redex", output the term.

There are two well-known (one-step) strategies: the innermost strategy and the outermost strategy. They select the innermost and outermost redeces of an input term respectively.

We show an example to explain how to reduce a term by the leftmost innermost strategy red_I and the leftmost outermost strategy red_O. Consider $t \equiv (0+s(0))+s(0)$ and the above TRS R. The leftmost innermost redex of t is $0+s(0)$ at position 1. So the leftmost innermost strategy first rewrites the subterm. The result term is $s(0+0)+s(0)$ and the next redex is 0+0 at position 1.1. Repeating such search and rewrite, we finally get $red_I(t) = s(s(0))$. On the other hand, the leftmost outermost redex of t is $(0+s(0))+s(0)$ at the root position. So the leftmost outermost strategy first rewrite the whole term into $s((0+s(0))+0)$. The next redex is $(0+s(0))+0$ at position 1. Repeat the same thing, we also get $red_O(t) = s(s(0))$. The following is the reduction sequences from $(0+s(0))+s(0)$ by the both strategies, where $\rightarrow_I$ corresponds to the leftmost innermost strategy and $\rightarrow_O$ corresponds to the leftmost outermost strategy.

$$\begin{tikzcd}
\underline{(\underline{0+s(0)})+s(0)} \arrow[r, "I"'] \arrow[d, "O"] & s(\underline{0+0})+s(0) \arrow[r, "I"'] & \underline{s(0)+s(0)} \arrow[r, "I"'] & s(\underline{s(0)+0}) \arrow[d, "I"] \\
s(\underline{(0+s(0))+0}) \arrow[r, "O"'] & s(\underline{0+s(0)}) \arrow[r, "O"'] & s(s(\underline{0+0})) \arrow[r, "O"'] & s(s(0))
\end{tikzcd}$$

By chance, the reduced terms are identical, however they do not always return the same term in general. For $R = \{f(a) \rightarrow a, a \rightarrow b\}$, $f(a) \rightarrow_I f(b)$ and $f(a) \rightarrow_O a$.

These strategies have merits and demerits, respectively. It is well-known that the innermost strategy is more suitable for efficiently implementing TRSs than the outermost one. We can define the more efficient implementation red'_I of the leftmost innermost strategy than the trivial implementation red_I which searches the whole term in every step.

$$red'_I(t) \quad = \begin{cases} t & \text{if} \quad t \text{ is a variable} \\ eval_I(t,\ 1{:}2{:}\cdots{:}n{:}nil) & \text{if} \quad t \equiv f(t_1, \ldots, t_n) \end{cases}$$

$$eval_I(t,l) \quad = \begin{cases} t & \text{if} \quad l = nil \text{ and } t \text{ is not a redex} \\ red'_I(r\theta) & \text{if} \quad l = nil, t \equiv l\theta \text{ for a rule } l \to r \in R \\ eval_I(f(\ldots, red'_I(t_i), \ldots), l') & \text{if} \quad l = i : l' \text{ and } t \equiv f(t_1, \ldots, t_n). \end{cases}$$

In this function red'_I, many wasteful search for a redex may be avoided. Conversely it is also well-known that the outermost strategy is more suitable for finding a normal form than the innermost one. A set of normal forms is the set of the terms which cannot be rewritten more and is regarded as answers of a given TRS. For rules $0 \times x \to 0$ and $\perp \to \perp$, although reduction of $0 \times \perp$ by the innermost strategy falls into a loop by rewriting the innermost redex $\perp$, the outermost strategy rewrites the whole term $0 \times \perp$ into 0 without rewriting $\perp$. For orthogonal TRSs, the parallel-outermost reduction strategy is normalizing, i.e. reducing any term which has a normal form always returns the normal form [O'Do77].

1.3 The evaluation strategy

The evaluation strategy (the E-strategy), adopted by OBJ languages, OBJ2[FGJM85], OBJ3[GWMFJ00] and CafeOBJ [DF98, NSF98], is one of the compromises between the innermost and outermost strategies. It may be regarded as a generalization of the above implementation red'_I of the leftmost innermost strategy. The E-strategy finds a redex position according to local strategies given to function symbols, not the structure of a whole term. Local strategies are given to every function symbols as integer lists. Since we can choose local strategies flexibly, the E-strategy can express various strategies. We can specify the order of evaluating arguments and the timing of rewriting a whole term by a local strategy.

We show an example to explain the behaviour of the E-strategy by the following TRS.

$$R = \begin{cases} x + 0 \to x \\ x + s(y) \to s(x + y) \end{cases}$$

For example, we give the list $[1, 2, 0]$ to $+$ as its local strategy, which instructs to "(for a given term $t_1 + t_2$,) reduce t_1 first, t_2 second and finally rewrite the whole term if possible". A function *red* is a strategy which depends on each instruct of local strategies of function symbols. We show the reduction sequence from $(s(0) + 0) + (0 + 0)$ by *red* under this local strategy:

$$(\underline{s(0) + 0}) + (0 + 0) \to_R s(0) + (\underline{0 + 0}) \to_R \underline{s(0) + 0} \to_R s(0).$$

The first argument $s(0) + 0$ is evaluated first, the second argument $0 + 0$ next and then the whole term $s0 + 0$ is rewritten into $s(0)$ because it is a redex. This is a basic instruct of the E-strategy and then *red* acts like the leftmost innermost strategy. An advantageous point of the E-strategy is that a local strategy can be given flexibly, for example, exchange of the order of arguments (like $[2, 1, 0]$), abbreviation of arguments (like $[2, 0]$) and change of the timing of the check whether the whole term is a redex or not (like $[0, 2, 0, 1]$).

Note that since the E-strategy can express various strategies, however, inappropriate strategies can be described. $red(0 + (0 + 0)) = 0 + 0$ for the local strategy $[1, 0, 2]$, but it could be rewritten yet nevertheless.

1.3.1 Lazy evaluation by the evaluation strategy

On the other hand, we can simulate a kind of lazy evaluation by the E-strategy if we define local strategies well. One way of simulating lazy evaluation is giving a symbol a local strategy specifying its arguments which we want to evaluate lazily after 0. For example if we want to evaluate the first argument of + lazily, we may give $[2, 0, 1]$ to it. Another way of simulating lazy evaluation is omitting such arguments, like $[2, 0]$.

The following is a typical example indicating that we can express a kind of lazy evaluation by the E-strategy.

$$R = \begin{cases} hd(x\!:\!y) \to x \\ tl(x\!:\!y) \to y \\ inf \to 0\!:\!inf \end{cases}$$

where hd returns the head element of a list, tl returns the list which is the result of removing the head element, : is a constructor symbol of lists and inf stands for an infinite list $0\!:\!0\!:\!0\!:\!\cdots$. If the local strategy of : has 2, $red(inf)$ does not terminates:

$$inf \to_R 0\!:\!inf \to_R 0\!:\!(0\!:\!inf) \to_R \cdots$$

If we give [1] to :, $red(inf)$ terminates and the result is $0\!:\!inf$ because the second argument inf of : can not be touched. Hence $red(hd(tl(inf)))$ also works well.

$$hd(tl(\underline{inf})) \to_R hd(tl(0\!:\!inf))$$

Since the local strategy of : does not have 2, $tl(0\!:\!inf)$ is reduced next without reducing inf and finally we obtain $s(0)$.

$$hd(tl(0\!:\!inf)) \to_R hd(inf) \to_R hd(0\!:\!inf) \to_R 0$$

Reduction with the E-strategy works well or not depending on how to give local strategies. In order to obtain meaningful strategies, we need to find a condition of local strategies on which each evaluated term has some appropriate property, such as a normal form. Local strategies are formalized by a map φ from the set of function symbols to the set of natural number's lists. For example, $\varphi(+) = [1, 2, 0]$. An E-strategy map is called correct if each evaluated term is a normal form. Some sufficient conditions for correctness are proposed [Nag99, NO00, Pol01][1] . However, it is trivial that correctness is too strong to treat the above example because inf has no normal form and $red(inf)$ does not terminate if it is correct. In this paper we propose a new condition for the E-strategy map. An E-strategy map is μ-correct if each evaluated term is a normal form of context-sensitive rewriting, called a μ-normal form. Context-sensitive rewriting is a kind of rewriting restricted by a replacement map on function symbols [Luc98]. There are some useful properties for μ-normal forms. We also propose some sufficient conditions on which an E-strategy map is μ-correctness. By combining our results with properties of context-sensitive rewriting, we can obtain some useful properties for evaluated terms of the E-strategy, e.g. for redeces, normal forms and root-stable forms.

[1]Although such a property is called completeness in [NO00, Pol01], we use the word 'correctness' to refer the property because the word 'completeness' would be usually used for the opposite property: '$s = red(t)$ if s is a normal form of t' as rewritten in [Luc01b].

1.3.2 Lazy evaluation by the on-demand evaluation strategy

Although the E-strategy the E-strategy can express various strategies, there exist strategies which cannot be defined by any local strategies. For example, we cannot simulate the outermost strategy by the E-strategy. The reason is that we only instruct an order of reduction for subterms directly under function symbols. For example, consider $R = \{f(f(a)) \to a, c \to a, f(a) \to f(a)\}$ and $f(f(c))$. The outermost strategy first rewrite c into a and next rewrite the whole term: $f(f(c)) \to_R f(f(a)) \to_R a$. In this sequence, the outermost strategy selects the innermost position first and the root position next with skipping the redex $f(a)$. We cannot describe such an instruction by the E-strategy.

There are some examples which the E-strategy cannot treat well. Consider the following example.

$$R = \begin{cases} 2nd(x\,{:}\,(y\,{:}\,z)) \to y \\ \mathit{inf} \to 0\,{:}\,\mathit{inf} \end{cases}$$

$2nd$ returns the second element of an input list. $2nd(\mathit{inf})$ has a normal form 0 since there is the reduction sequence $2nd(\mathit{inf}) \to_R 2nd(0 : \mathit{inf}) \to_R 2nd(0 : (0 : \mathit{inf})) \to_R 0$. There is no local strategy to obtain this reduction sequence. If $\varphi(:)$ has 2, $red(\mathit{inf})$ does not terminates. If $\varphi(:)$ does not have 2, $red(2nd(\mathit{inf}))$ terminates halfway as follows: $2nd(\mathit{inf}) \to_R 2nd(0\,{:}\,\mathit{inf})$.

For solving this problem the on-demand E-strategy has been proposed and discussed in the literatures [NSF98, OF00, NO00]. In the on-demand E-strategy, local strategies may have negative integers. For negative integers $-i$, the i-th arguments are not evaluated until so forced. Such an argument is forced to evaluate if it is involved in matching. The subterm inf of $0 : \mathit{inf}$ is not evaluated until so forced if $\varphi(:)$ has -2. Trying to match the term $2nd(0\,{:}\,\mathit{inf})$ to the pattern $2nd(x\,{:}\,(y\,{:}\,z))$, inf have to be rewritten because the matching fails if not. On the other hand, for $2nd(0\,{:}\,(0\,{:}\,\mathit{inf}))$, we do not have to rewrite inf it is not involved in the matching. In the latter half of this thesis, we formalize the on-demand E-strategy and propose a sufficient condition on which each evaluated term is always a root-stable form. The on-demand E-strategy is formalized by a pair of an E-strategy map and an on-demand map which stand for order of reduction and order of matching respectively. For the on-demand E-strategy order of matching is important because a term may be changed which matching it to patterns. We show some examples that we can treat well owing to an on-demand map.

1.4 Overview

The next chapter gives the basic definition of term rewriting system and the evaluation strategy. First we present some basic mathematical notation and the abstract reduction system. The term rewriting systems is a kind of the abstract reduction system whose target is a set of terms. The evaluation strategy is introduced in the last section of this chapter. Local strategies are formalized by a map from signature Σ to the set of lists of natural numbers, called an E-strategy map. For a given E-strategy map we define the function red on terms, which takes a term wanted to be evaluated and returns a term evaluated according to the local strategies.

In the chapter 3 we introduce some basic properties of the E-strategy. We define the condition of the E-strategy, which should be satisfied at worst, called the safety condition.

We show that a fundamental property of the E-strategy holds if it is safe. We also introduce the notion of evaluated flags which is used to get a reasonable implementation of the E-strategy and prove that the safety condition justifies the use of evaluated flags.

In the chapter 4 we explain one of the main topics of this thesis: correctness and strictness analysis of the E-strategy. We introduce the context-sensitive rewriting and some useful properties of it. We generalize the notion of correctness to context-sensitive rewriting, called μ-correctness. Some useful properties for context-sensitive rewriting have been proposed. By combining those properties with our result of μ-correctness, it can be obtained some useful properties for the E-strategy. In the latter of this chapter, we define the notion of strictness in the E-strategy and propose some method of analyzing the strictness. In lazy functional languages, a function is strict in a certain argument if the eager evaluation of that argument does not change the termination behavior of the program. Strictness analysis is a compile-time analysis of a program that is used to tell whether or not a function is strict in its argument.

In the chapter 5, we formalize the on-demand E-strategy by the pair of maps whose first element is the E-strategy map φ defined in the above section and latter element is the on-demand map o. Although the E-strategy map defines the order of reduction, the on-demand map defines the order of matching. We also give methods for obtaining a suitable on-demand map. In the chapter 6, we discuss applications of the E-strategy to the CafeOBJ system.

Chapter 2

Preliminaries

In this chapter, we give the definitions and basic properties of term rewriting [BN98] and the evaluation strategy [DF98, NSF98, Eke98].

2.1 Some basic notations

For a set A, the set of all subsets of A is denoted by $\mathcal{P}(A) = \{B \mid B \subset A\}$. The set A^* of all strings of A is defined as $\varepsilon \in A^*$, $a \in A^*$ for $a \in A$ and $w.w' \in A^*$ for $w, w' \in A^*$, where ε is the empty string. The set of all maps from a set A to a set B is denoted by $B^A = \{f \mid f : A \to B\}$. The set $\mathit{List}(A)$ of lists whose elements are elements of A is defined as the smallest set satisfying the following: $\mathit{nil} \in \mathit{List}(A)$ and $a : l \in \mathit{List}(A)$ for $a \in A$ and $l \in \mathit{List}(A)$. We sometimes write $[a, b, c]$ instead of $a : (b : (c : \mathit{nil}))$. For a list l, the head element is denoted by $\mathit{hd}(l)$ and the remaining list obtained by removing $\mathit{hd}(l)$ is denoted by $\mathit{tl}(l)$, i.e. $\mathit{hd}(a : l) = a$ and $\mathit{tl}(a : l) = l$. $l@l'$ is the result of concatenation of lists l and l', for example, $[a, b, c]@[d, e] = [a, b, c, d, e]$. The set of all natural numbers is denoted by $\mathcal{N} = \{0, 1, 2, \ldots\}$ and the set of all positive natural numbers is denoted by $\mathcal{N}_+ = \mathcal{N} \setminus \{0\}$.

2.1.1 Abstract reduction systems

An abstract reduction system (ARS) is a pair $(A, \to)$ (or just $\to$ for short) of a set A and a binary relation $\to \in A \times A$, called a reduction relation, on the set A. We write $a \to b$ instead of $(a, b) \in \to$ and $a_1 \to a_2 \to \cdots \to a_n \to \cdots$ instead of $a_1 \to a_2, a_2 \to a_3$ and $a_{n-1} \to a_n, \cdots$. For two binary relations $\to$ and $\to'$ on a set A, the binary relation $\to \circ \to'$ is defined as $\{(a, c) \mid a \to b, b \to' c\}$. We define some binary relations and properties for an ARS $(A, \to)$ as follows:

1. The identity relation $\equiv$ is $\{(a, a) \mid a \in A\}$.

2. The n-steps relation, denoted by $\to^n$, is $\to^0 = \equiv$ and $\to^{n+1} = \to^n \circ \to$.

3. The transitive-reflexive closure is $\to^* = \bigcup_{n \in \mathcal{N}} \to^n$.

4. The transitive closure is $\to^+ = \bigcup_{n > 0} \to^n$.

5. An element $a \in A$ is in normal form with respect to $\to$ if and only if there is no element $b \in A$ such that $a \to b$.

6. An element $a \in A$ has a normal form with respect to $\rightarrow$ if and only if there is an element $b \in A$ such that b is in normal form and $a \rightarrow^* b$. We call b a normal form of a.

7. An element $a \in A$ terminates (or is terminating) if and only if there is no infinite reduction sequences $a \rightarrow a_1 \rightarrow a_2 \rightarrow \cdots$.

8. Elements $a, b \in A$ are joinable if there is an element c such that $a \rightarrow^* c$ and $b \rightarrow^* c$.

If each element $a \in A$ terminates, $\rightarrow$ terminates (is terminating). It is often said to that termination corresponds to the existence of solution when an ARS is regarded as some computational model. This means that any element has a solution (a normal form) and we can get it in finite times. For an ARS and an element, a normal form is not always unique in general. Hence we need some property for the uniqueness of solutions. $\rightarrow$ has the unique normal form property if a normal form of each term is unique, i.e. if $a \rightarrow^* b$, $a \rightarrow^* c$ and b and c are in normal form, then $a \equiv b$. We show useful properties for the uniqueness of solutions. $\rightarrow$ is confluent if for any elements $a, b, c \in A$ such that $a \rightarrow^* b$ and $a \rightarrow^* c$, then b, c are joinable. $\rightarrow$ is local confluent if for any elements $a, b, c \in A$ such that $a \rightarrow b$ and $a \rightarrow c$, then b, c are joinable. You can see that any confluent ARS has the unique normal form property. If $a \rightarrow^* b$, $a \rightarrow^* c$ and b and c are in normal form for a confluent ARS, b, c are joinable, i.e. there exists d such that $b \rightarrow^* d$ and $c \rightarrow^* d$. Since b and c are in normal form, we conclude $b \equiv d \equiv c$. Moreover any local confluent and terminating ARS has the unique normal form property. When an ARS terminates, it is decidable whether it is local confluent or not.

In general a reduction sequence from an element is not unique. So if we want to implement an ARS, we need a function which takes an element a and returns an element b which is a result element of reduction from a, i.e. $a \rightarrow^* b$. We call such a function F a strategy of an ARS. Note that in our definition an element $F(a)$ is not necessary to be in normal form. We call a strategy correct If it always returns a normal form, For a terminating ARS, we can easily to obtain a decidable correct strategy since any reduction sequence is finite. Moreover if it has the unique normal form property, the strategy always returns a unique normal form in finite times. A strategy plays an important role especially for a non-terminating one because there is a case that an element has a normal form but does not terminate. Even in such a case we may obtain a normal form if we select a suitable strategy.

2.2 Term rewriting

Term rewriting is an ARS $(T(\Sigma, V), \rightarrow_R)$ where $T(\Sigma, V)$ is a set of terms and $\rightarrow_R$ is a binary relation on $T(\Sigma, V)$ defined by a given term rewriting system R. In this section we introduce terms, term rewriting systems and some properties on term rewriting.

2.2.1 Terms

A term is a tree which has function symbols as its nodes and variables as its leafs. For example, $f(g(x), f(0, y))$ is a term for a binary function symbol f, a unary function symbol g, a constant c and variables x, y (Fig. 2.1).

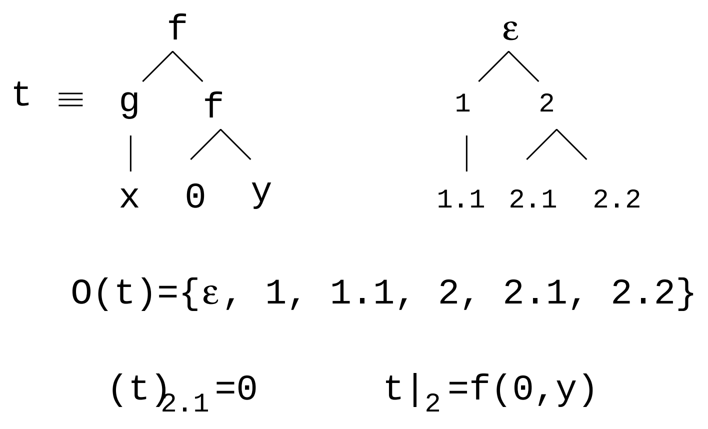

Figure 2.1: terms, positions and subterms

A signature Σ is a finite set of function symbols where every $f \in \Sigma$ has a fixed arity $ar(f) \in \mathcal{N}$. A countably infinite set of variables V is defined as $\Sigma \cap V = \emptyset$. A set of terms $T(\Sigma, V)$ (or T) is the smallest set defined as follows: $V \subset T(\Sigma, V)$ and $f(t_1, \ldots, t_n) \in T(\Sigma, V)$ for $t_1, \ldots, t_n \in T(\Sigma, V)$ and $f \in \Sigma$ with $ar(f) = n$. Note that $c \in \Sigma$ is called a constant if $ar(c) = 0$. We write just c instead of a term $c()$. $f(g(x), f(0, y)) \in T(\Sigma, V)$ for $\Sigma = \{f, g, 0\}$ and $V = \{x, y, \ldots\}$ where $ar(f) = 2$, $ar(g) = 1$ and $ar(c) = 0$. A term having no variable is called a ground term and a set of all ground terms are denoted by $T(\Sigma)$. This means that any leaf of ground terms is a constant. Therefore if there is no constant symbol in Σ, $T(\Sigma)$ must be the empty set. The set of positions $O(t) \subset \mathcal{N}_+^*$ of a term t is defined as follows:

$$O(t) = \begin{cases} \{\varepsilon\} & \text{if} \quad t \in V \\ \{\varepsilon\} \cup \displaystyle\bigcup_{i=1}^{n} \{i.p \in \mathcal{N}_+^* \mid p \in O(t_i)\} & \text{if} \quad t \equiv f(t_1, \ldots, t_n). \end{cases}$$

For example $O(f(g(x), f(0, y))) = \{\varepsilon, 1, 1.1, 2, 2.1, 2.2\}$.

We introduce some notations and properties on terms.

1. The subterm of t at a position $p \in O(t)$, denoted by $t|_p$, is defined as: $t|_\varepsilon \equiv t$ and $f(t_1, \ldots, t_n)|_{i \cdot p} \equiv t_i|_p$. If $p \neq \varepsilon$, we call $t|_i$ a strict subterm of t.

2. The term $t[s]_p$ is obtained from t by replacing the subterm at position p by s.

3. The symbol at a position p of a term t is denoted by $(t)_p$: $(t)_p = x$ if $t|_p = x \in V$ and $(t)_p = f$ if $t|_p = f(\ldots)$. Especially the symbol at the root position $(t)_\varepsilon$ of t is called the root symbol of t.

4. The set of positions restricted by a set $A \subset \Sigma \cup V$ is denoted by $O_A(t) = \{p \in O(t) \mid (t)_p \in A\}$.

5. The set of variables of a term t is denoted by $V(t) = \{(t)_p \mid (t)_p \in V\}$.

6. A term t is linear if for any positions $p, q \in O_V(t)$, $(t)_p = (t)_q$ implies $p = q$.

For example,

$$\begin{array}{rcl}
f(g(x), f(0, y))|_2 & = & f(0, y), \\
f(g(x), f(0, y))[g(g(x))]_2 & = & f(g(x), g(g(x))), \\
(f(g(x), g(g(x))))_{2.1} & = & g, \\
O_\Sigma(f(g(x), f(0, y))) & = & \{\varepsilon, 1, 2, 2.1\}, \\
O_V(f(g(x), f(0, y))) & = & \{1.1, 2.2\} \text{ and} \\
V(f(g(x), f(0, y))) & = & \{x, y\}.
\end{array}$$

$f(g(x), f(0, y))$ is linear but $f(g(x), g(g(x)))$ is not linear because x occurs twice.

A map θ from variables to terms is called a substitution if $\theta(x) \neq x$ for only finitely many xs. A substitution over terms is defined as homomorphic extension. For a term t and a substitution θ, $t\theta$ is written instead of $\theta(t)$. A term t is called an instance of a term s if there exists θ such that $t = s\theta$ and especially an instance is called a ground instance if it is a ground term. For example, $f(g(x), f(0, y))\theta = f(g(0, f(0, g(x))))$ for $\theta(x) = 0$, $\theta(y) = g(x)$ and $\theta(z) = z$ for any $z \in V - \{x, y\}$.

2.2.2 Rewrite relation

Next we define a binary relation $\to_R$ on terms, called a rewrite relation. A term rewriting system (TRS) is a set R of rewrite rules. A rewrite rule is a pair of terms, denoted by $l \to r$, such that $l \notin V$ and $V(r) \subset V(l)$. A rewrite relation $\to_R$ is a binary relation on T defined as follows (Fig: 2.2).:

$$s \to_R t \overset{\text{def}}{\Longleftrightarrow} \exists p \in O(s), l \to r \in R, \theta \in \mathcal{T}^V.s|_p \equiv l\theta, t \equiv s[r\theta]_p.$$

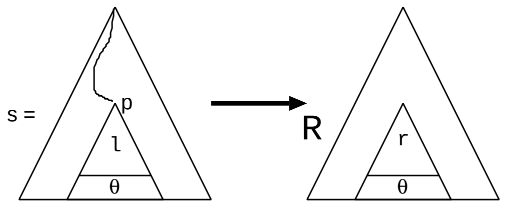

Figure 2.2: A rewrite relation $\to_R$

Especially a rewrite relation restricted a redex position to $p \in O(s)$ is denoted by $s \to_p t$. The condition of rewrite rules is important. If $x \to r \in R$, we will be able to rewrite any term into r. Under the condition $V(r) \subset V(l)$ we should consider only finite terms as terms which we rewrite a given term into if R is a finite set.

A subterm $t = s|_p$ is a redex of s if t is an instance of the left-hand side of a rewrite rule. We call p a redex position of t. t is a normal form if $t|_p$ is not redex for each $p \in O(t)$. t is a root stable form if there is no redex u such that $t \to_R^* u$.

Example 2.2.1 Consider a TRS $R = \{a \to b, f(c) \to c\}$. Of course normal forms $b, c, f(b)$ are root-stable forms and redeces $a, f(c)$ are not root-stable forms. Although the term $f(a)$ is not a normal form, it is a root-stable form since the only reduction from this term is $f(a) \to_R f(b)$ and $f(b)$ is not a redex. Although the term $f(f(c))$ is not a redex, it is not a root-stable form since $f(f(c)) \to_R f(c)$ and $f(c)$ is a redex (Fig: 2.3).

A root-stable form can be considered as a sub goal of to obtain a normal form. If there is a procedure R which takes a term t and returns a root-stable form reduced from t, we can define a procedure N which takes a term t and returns a normal form reduced from t as follows:

$$N(t) = \begin{cases} x & \text{if} \quad R(t) = x \in V \\ f(N(t_1), \ldots, N(t_n)) & \text{if} \quad R(t) = f(t_1, \ldots, t_n) \end{cases}$$

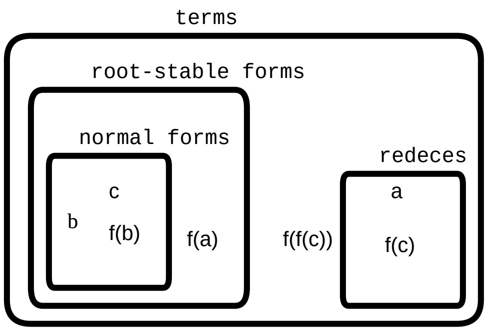

Figure 2.3: normal forms, root-stable forms and redeces

We introduce some notations and properties on TRS R.

1. The set of left-hand sides of R is $L(R) = \{l \in T \mid l \to r \in R\}$.
2. The set of defined symbols is defined as $D(R) = \{(l)_\varepsilon \in \Sigma \mid l \to r \in R\}$.
3. The set of constructor symbols is defined as $C(R) = \Sigma - D(R)$.
4. R is left-linear if l is linear for each $l \to r \in R$.
5. R is non-overlapping if there is no $l \to r \in R$ and $l' \to r \in R$ such that $l|_p\theta \equiv l'\theta'$ for some $\theta, \theta' \in T^V$ and $p \in O_\Sigma(l)$.
6. R is orthogonal if R is left-linear and non-overlapping.
7. A term is constructor if there is no defined symbol in the term. R is constructor if t_i is constructor for each $f(t_1, \ldots, t_n) \to r \in R$ and $1 \leq i \leq n$.

A TRS each of whose left-hand side is $f(x_1, \ldots, x_n)$ is called a recursive program scheme (RPS), where $f \in \Sigma$ and $x_i \neq x_j$ for $i \neq j$. It is easily to prove that a normal form for a RPS does not have any defined symbol. If there is a defined symbol for a term $(t)_p = f \in D(R)$ for an RPS R, the subterm $t|_p$ is a redex because there is a rule $f(x_1, \ldots, x_n) \to r \in R$ and we will be able to apply the rule to any term whose root symbol is f. For such a rule the defined symbol can be regarded as a total function, which means that a function is defined for any arguments.

A term is reducible if it is not in normal form. A term t is ground reducible if all its ground instances is reducible. A reducible term is of course ground reducible. There is a term such that it is not reducible but ground reducible. For example, we consider $\Sigma = \{+, s, 0\}$ with $ar(+) = 2, ar(s) = 1, ar(0) = 0$ and the following TRS:

$$R_{n_1} = \left\{ \begin{array}{l} +(x, 0) \to x \\ +(x, s(y)) \to s(+(x, y)) \end{array} \right.$$

A term $+(x, y)$ is ground reducible because any ground instance is reducible. We prove that $+(t, t')$ is reducible for any ground terms t, t' by induction on the structure of $+(t, t')$. There are only cases that the root symbol of t' is 0, s or $+$. In the case of $t' \equiv 0$, $+(t, 0)$ is an instance of the left-hand side of the first rule and is reducible. In the case of $t' \equiv s(t'')$, $+(t, s(t''))$ is reducible by the second rule. In the case of $t' \equiv +(t'', t''')$, by the induction hypothesis t' is reducible. Hence the whole term $+(t, t')$ is also reducible. Ground reducibility of a term $f(x_1, \ldots, x_n)$ can be regarded as totality of a function f on $T(\Sigma)$.

A function $S : T \to \mathcal{N}_+^* \cup \{$"no redex"$\}$ is a one-step reduction strategy if it satisfies the follows: $S(t) = p \in \mathcal{N}_+^*$ if p is a redex position of t, or $S(t) =$ "no redex" if t has no redex, i.e. is in normal form. We can define a strategy F_S of $\to_R$ from a one-step reduction strategy as follows:

$$F_S(t) = \left\{ \begin{array}{lll} F_S(r\theta) & \text{if} & S(t) = p \text{ and } t|_p \equiv l\theta \text{ for some } \theta \in T^V, l \to r \in R \\ t & \text{if} & S(t) = \text{"no redex"} \end{array} \right.$$

We can easily see that F_S is a strategy of $\to_R$ if S is a one-step reduction strategy, i.e. $s \to_R^* t$ if $F_S(s) = t$. Moreover F_S is correct, i.e. t must be in normal form if $F_S(s) = t$ because whenever F_S outputs a term t, $S(t) =$ "no redex".

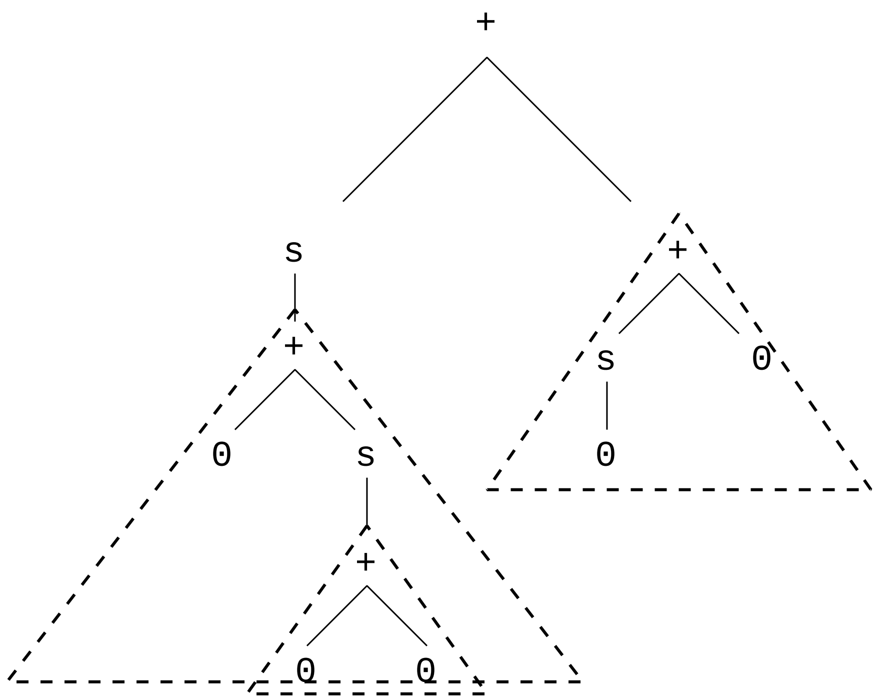

Figure 2.4: The leftmost innermost redex and the leftmost outermost redex

There are two well-known one-step reduction strategies: the leftmost innermost one and the leftmost outermost one. We call a position inner and outer when it is close by leafs and the root position respectively. For a term t the leftmost innermost redex position $p \in O(t)$ is a redex position such that if q is a redex position of t, $q.q' = p$ for some position q', or $p = p'.i.p''$ and $q = p'.j.q''$ for some positions p', p'', q'' and $i < j$. On the other hand, the leftmost outermost redex position $p \in O(t)$ is a redex position such that if q is a redex position of t, $q = p.p'$ for some position p', or $p = p'.i.p''$ and $q = p'.j.q''$ for some positions p', p'', q'' and $i < j$. The leftmost innermost strategy S_I and the leftmost outermost strategy S_O return the leftmost innermost and the leftmost outermost redex position respectively. We sometimes call F_{S_I} the leftmost innermost strategy and F_{S_O} the leftmost outermost strategy. They are often called the eager evaluation and the lazy evaluation.

Example 2.2.2 For example, $+(s(+(0, s(+(0, 0)))), +(s(0), 0))$ has the following three redeces:

$+(0, s(+(0, 0)))$ at position 1.1,

$+(0, 0)$ at position 1.1.2.1 and

$+(s(0), 0)$ at position 2.

The leftmost innermost redex position is 1.1.2.1 and the leftmost outermost redex position is 1.1 (Fig. 2.4).

2.3 The evaluation strategy

In this section, we introduce the evaluation strategy (the E-strategy)[DF98, NSF98, Eke98]. The E-strategy finds a redex position according to local strategies given to symbols, not the structure of a whole term. Since the E-strategy has to remember the redex position rewritten directly before to find a next redex, we cannot define the E-strategy as a one-step reduction strategy.

Local strategies are given to every symbol as integer lists. They are formalized by a map $\varphi : \Sigma \to \mathit{List}(\mathcal{N})$ called an E-strategy.

Definition 2.3.1 [Eke98] A map $\varphi : \Sigma \to \mathit{List}(\mathcal{N})$ is an E-strategy map if $0 \leq i \leq ar(f)$ for any $i \in \varphi(f)$. A map φ is extended to $\Sigma \cup V$ with $\varphi(x) = nil$ for any $x \in V$.

For example, if we want to give [120] to + as its local strategy, we give a replacement map φ as $\varphi(+) = [1, 2, 0]$. This corresponds to the instruction to reduce the first and second argument before the whole term $+(s, t)$.

A strategy *red* is a strategy which depends on each instruct of local strategies of function symbols. We sometimes call *red* an E-strategy. The reduction sequence from $(s(0) + 0) + (0 + 0)$ by *red* under this local strategy is that:

$$(\underline{s(0) + 0}) + (0 + 0) \to_R s(0) + (\underline{0 + 0}) \to_R \underline{s(0) + 0} \to_R s(0).n$$

Definition 2.3.2 [Eke98] For an E-strategy map φ and TRS R, the E-strategy $red : T \to T$ is defined as follows:

$$\begin{array}{lcl} red(t) & = & eval(t, \varphi((t)_\varepsilon)) \\ eval(t,l) & = & \begin{cases} t & \text{if} \quad l = nil \\ eval(t[red(t_i)]_i, l') & \text{if} \quad l = i\!:\!l', i > 0 \\ red(s) & \text{if} \quad l = 0\!:\!l', t \to_\varepsilon s \\ eval(t, l') & \text{if} \quad l = 0\!:\!l', t \notin Red(R) \end{cases} \end{array}$$

The function *red* is a top-level function that takes a term and returns the term which has been evaluated according to local strategies. For an input term t a term $s = red(t)$ is called an evaluated term of t. A term is evaluated according to the local strategy list of the root symbol $(t)_\varepsilon$.

$$red(t) = eval(t, \varphi((t)_\varepsilon))$$

For empty list the function *eval* returns the term in the first argument as an evaluated term. *eval* outputs a term only if the second argument is empty.

$$eval(t, nil) = t$$

For non-empty list, there are two cases depending on the head element of the list. If it is a positive integer i, remove i from the list and reduce the i-th argument of the term.

$$eval(f(t_1, \dots, t_n), i\!:\!l) = eval(f(t_1, \dots, red(t_i), \dots, t_n), l)$$

If it is 0, there are more two cases depending on the shape of the term: a redex or not. If it is a redex, i.e. $t \equiv l\theta$ for some $l \to r \in R$ and a substitution θ, rewrite the term into the corresponding right-hand side and apply the function *red* again.

$$eval(t, 0\!:\!l) = red(r\theta)$$

If it is not redex, remove the 0 and go on evaluating (Fig. 2.5).

$$eval(t, 0\!:\!l) = eval(t, l)$$

Example 2.3.3 Consider TRS R_{n_1} and $\varphi(+) = [2, 0, 1], \varphi(s) = [1], \varphi(0) = []$

$$R_{n_1} = \begin{cases} +(x, 0) \to x \\ +(x, s(y)) \to s(+(x, y)) \end{cases}$$

We show a reduction sequence of the term $+(+(+(0, 0), x), +(0, 0))$ in this strategy.

First the function *red* hands the input term and the local strategy $[2, 0, 1]$ of the root symbol $+$ to the function *eval*. The function *eval* reduce the second argument term $+(0, 0)$ into the term 0 according to the first element 2 of the local strategy list and remove 2 from the list.

$$\begin{array}{ll} red(+(+(+(0,0),x),+(0,0))) & \\ = eval(+(+(+(0,0),x),+(0,0)), & [2,0,1]) \\ = eval(+(+(+(0,0),x),0), & [0,1]) \end{array}$$

Now the head element of the local strategy list is 0. The target term is checked whether it is a redex or not. The term $+(+(+(0, 0), x), 0)$ is a redex for the first rule $+(x, 0) \to x$.

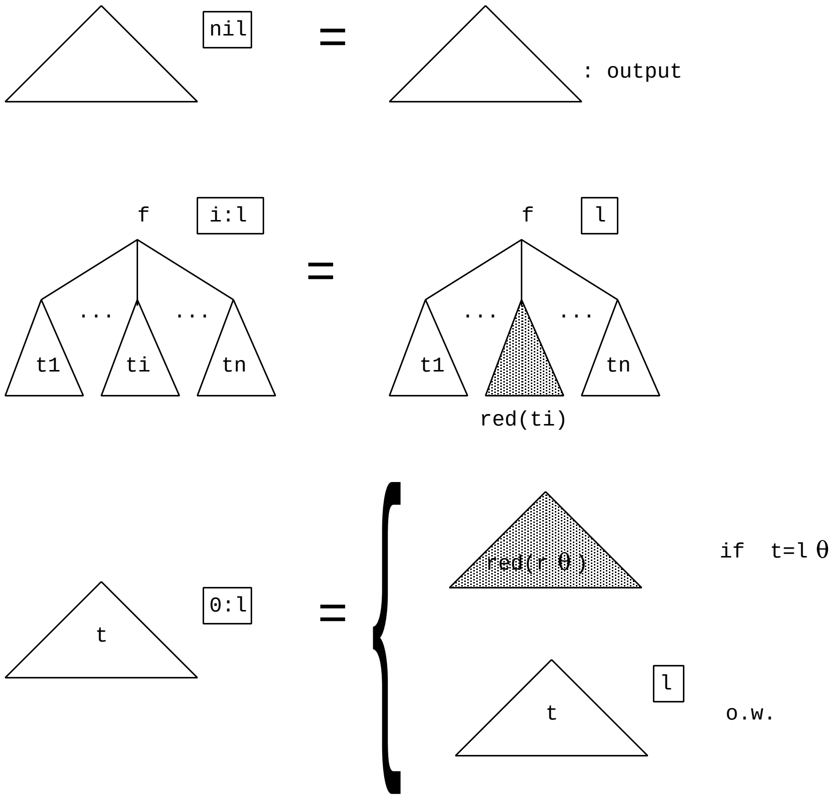

Figure 2.5: Definition of function *eval*

Then it is replaced with the corresponding instance of the right-hand side and again the function *red* is applied to the replaced term $+(+(0,0),x)$.

$$\begin{aligned}
&= red(+(+(0,0),x)) \\
&= eval(+(+(0,0),x), \quad [2,0,1]) \\
&= eval(+(+(0,0),x), \quad [0,1])
\end{aligned}$$

This time the target term $+(+(0,0),x)$ is not a redex. Hence the term was not changed and only the element 0 is removed. Since the head element is 1, the first argument $+(0,0)$ is reduced into 0. Finally the list becomes empty and the term $+(0,x)$ is outputted as an evaluated term of $+(+(+(0,0),x),+(0,0))$.

$$\begin{aligned}
&= eval(+(+(0,0),x), \quad [1]) \\
&= eval(+(0,x), \qquad\qquad [\,]) \\
&= +(0,x)
\end{aligned}$$

Therefore $red_\varphi(+(+(+(0,0),x),+(0,0))) = +(0,x)$.

Chapter 3

Basic properties of the evaluation strategy

Since we can define a local strategy as we like, *red* may behave undesirably. For example, assuming the local strategy of a defined symbol f has not the element 0, a term $f(\ldots)$ is not rewritten at the root position even if it is a redex. In this chapter we define a condition of an E-strategy, which should be satisfied at worst, called the safety condition. We show that some fundamental basic properties hold for a safe E-strategy, for example a term which is a result of evaluating by the E-strategy cannot be reduced more, i.e. $red(t) = red(red(t))$. In the literature [Eke98], the condition that local strategies are restricted to lists which end in zero is proposed and some properties are proved on this condition. In this chapter we prove these properties under a weaker condition which satisfies the safety condition.

3.1 Safety condition

As a condition which should be satisfied at worst we take the following safety condition on which a term is not a redex if it is a result of evaluating by the E-strategy.

Definition 3.1.1 An E-strategy *red* (or an E-strategy map φ) is safe if each evaluated term is not a redex, i.e. $t \notin Red(R)$ if $t = red(s)$ for some s.

For a safe E-strategy we can show the following fundamental property.

Theorem 3.1.2 *If red is safe, any evaluated term is not changed any more, i.e.* $red(t) = red(red(t))$ *for any* t.

Proof. We prove the claim by induction on the structure of the evaluated term t. It is trivial for $t \in V$. We assume $t \equiv f(t_1, \ldots, t_n)$ and $\varphi(f) = [i_1, \ldots, i_m]$. Hence $red(t) = eval(f(t_1, \ldots, t_n), [i_1, \ldots, i_m])$. If $i_1 \neq 0$, the term t_{i_1} is a result of evaluating some term, i.e. $t_{i_1} = red(s)$ for some s because t has been evaluated once by the same local strategy $\varphi(f)$. From the induction hypothesis $t_{i_1} = red(t_{i_1})$ and

$$\begin{array}{cll} & eval(t, & [i_1, i_2, \ldots, i_m]) \\ = & eval(f(\ldots, red(t_{i_1}), \ldots), & [i_2, \ldots, i_m]) \\ = & eval(f(\ldots, t_{i_1}, \ldots), & [i_2, \ldots, i_m]) \\ = & eval(t, & [i_2, \ldots, i_m]) \end{array} .$$

Next we consider the case where $i_1 = 0$. Since *red* is safe, the term t is not a redex. Hence the function *eval* does not change the term and remove the element $i_1 = 0$ from the list. Therefore for each element i_1 we conclude $eval(t, [i_1, i_2, \ldots, i_m]) = eval(t, [i_2, \ldots, i_m])$. From the same reason all element i_k ($k = 2, \ldots, m$) are removed without changing the term and the function *eval* returns the input term as it is. In brief, $eval(f(t_1, \ldots, t_n), [i_2, \ldots, i_m]) = eval(f(t_1, \ldots, t_n), []) = t$. $\Box$

This property is very fundamental for the E-strategy because it is essential to obtain the most results about the E-strategy that we will introduce after here, that are evaluated flags, simplifying the local strategies, correctness and strictness of the E-strategy and so on.

3.2 The evaluated flags

Theorem 3.1.2 justifies the use of evaluated flags (or reduced flags)[NMOF98] for a safe E-strategy. The evaluated flags are marked on terms that have been evaluated so that the terms cannot be evaluated more than once. The use of the evaluated flags gives us an efficient implementation of rewriting with the E-strategy[OF97].

Definition 3.2.1 [NMOF98] Let Σ^* be a set of marked function symbols from Σ, i.e. $\Sigma^* = \{f^* \mid f \in \Sigma\}$, such that $ar(f) = ar(f^*)$ and $\Sigma \cap \Sigma^* = \emptyset$. Let $T^* = T(\Sigma \cup \Sigma^*, V)$ be a set of marked and unmarked terms and $T = T(\Sigma, V)$ a set of unmarked terms. The function $mark : T^* \to T^*$ marks the root symbol of an input term and $erase : T^* \to T$ eliminates all marks.

$$\begin{array}{rcl} mark(t) & = & \begin{cases} t & \text{if} \quad t \in V \\ f^*(\vec{t_i}) & \text{if} \quad t \equiv f(\vec{t_i}) \text{ or } t \equiv f^*(\vec{t_i}) \end{cases} \\ erase(t) & = & \begin{cases} t & \text{if} \quad t \in V \\ f(\overrightarrow{erase(t_i)}) & \text{if} \quad t \equiv f(\vec{t_i}) \text{ or } t \equiv f^*(\vec{t_i}) \end{cases} \end{array}$$

We define the rewrite relation $\to_{R^*} \subset T^* \times T^*$, which is a marked version of the rewrite relation, as follows:

$$s \to_{R^*} t \overset{\text{def}}{\iff} \begin{array}{l} \exists p \in O(s), l \to r \in R, \theta : V \to T^*, l' \in T^*. \\ erase(l') \equiv l, s|_p \equiv l'\theta, t \equiv s[r\theta]_p. \end{array}$$

Note that when a marked term is rewritten by this rewrite relation $\to_{R^*}$, the marks in substituted subterms are preserved. For example $+(s^*(s^*(0^*)), 0^*) \to s(+(s^*(0^*), 0^*)$ by the rule $+(s(x), y) \to s(+(x, y))$. Especially $\to_{\varepsilon^*} \subset T^* \times T^*$ is the marked version of the rewrite relation at the root position.

Now we introduce the E-strategy reduction with the evaluated flags.

Definition 3.2.2 [NMOF98] For an E-strategy map φ and TRS R, the function $red^* : T^* \to T^*$ is defined as follows:

$$\begin{array}{rcl} red^*(t) & = & \begin{cases} t & \text{if} \quad (t)_\varepsilon \in \Sigma^* \\ eval^*(t, \varphi((t)_\varepsilon)) & \text{o.w.} \end{cases} \\ eval^*(t, l) & = & \begin{cases} mark(t) & \text{if} \quad l = nil \\ eval^*(t[red^*(t_i)]_i, l') & \text{if} \quad l = i\!:\!l', i > 0 \\ red^*(s) & \text{if} \quad l = 0\!:\!l', t \to_{\varepsilon^*} s \\ eval^*(t, l') & \text{if} \quad l = 0\!:\!l', erase(t) \notin Red(R) \end{cases} \end{array}$$

Main differences between the E-strategy with and without the evaluated flags are behaviors of the function *red** when the root symbol of an input term is marked and the function *eval** when it returns the evaluated term, i.e. the list is empty. In this definition a marked term stands for an evaluated term and we do not reduce it again. The function *red** checks whether the root symbol is marked or not.

$$red^*(t) = \begin{cases} t & \text{if} \quad (t)_\varepsilon \in \Sigma^* \\ eval^*(t, \varphi((t)_\varepsilon)) & \text{o.w.} \end{cases}$$

If it is marked, *red** returns an input term immediately because the mark of the root symbol means that the term is already evaluated. The function *eval** marks the term when the list is empty.

$$eval^*(t, nil) = mark(t)$$

The E-strategy returns a term only when consuming all elements of the local strategy list. Hence we can say that each evaluated term is marked in this definition.

By use of the evaluated flags we can obtain more efficient implementation than the original definition of the E-strategy because *red** avoids wasteful pattern matching with the left-hand sides of rules.

Example 3.2.3 Consider TRS R_{n_1} again

$$R_{n_1} = \begin{cases} +(x, 0) \to x \\ +(s(x), y) \to s(+(x, y)) \end{cases}$$

Let φ be an E-strategy map where $\varphi(+) = [1, 2, 0], \varphi(s) = [1], \varphi(0) = []$.

We show a reduction sequence of the term $+(+(0, s(0)), 0)$ by the E-strategy with the evaluated flags.

$$\begin{aligned} red^*(+(+(0, s(0)), 0)) &= eval^*(+(+(0, s(0)), 0), \quad [1, 2, 0]) \\ &= eval^*(+(red^*(+(0, s(0))), 0), \quad [2, 0]) \end{aligned}$$

Now the first argument of the input term is reduced by the function *red**. Then the result term is marked at the root symbol (and all symbols also are marked since they are results of *red** too). The second argument is also marked similarly.

$$\begin{aligned} &= eval^*(+(+^*(0^*, s^*(0^*)), 0), \quad [2, 0]) \\ &= eval^*(+(+^*(0^*, s^*(0^*)), 0^*), \quad [0]) \end{aligned}$$

For the term $+(+^*(0^*, s^*(0^*)), 0^*)$ the result of erasing marks is a redex. Then the term is rewritten into $+^*(0^*, s^*(0^*))$ by the rewrite relation $\to_{\varepsilon^*}$ and apply *red** again. Note that the root symbol of the result term is marked. Hence the function *red** returns the term immediately without wasteful matching.

$$\begin{aligned} &= red^*(+^*(0^*, s^*(0^*))) \\ &= +^*(0^*, s^*(0^*)) \end{aligned}$$

Although the use of evaluated flags gives us an efficient implementation of the E-strategy, there are some cases such that evaluated terms of the E-strategy function *red* are different from that of *red** with evaluated flags for a same E-strategy map φ.

Example 3.2.4 [NMOF98] Consider the following TRS R.

$$R = \begin{cases} f(x) \to x \\ g(b) \to c \\ a \to b \end{cases}$$

Let φ be an E-strategy map such that $\varphi(f) = [1, 0]$, $\varphi(g) = [0, 1]$ and $\varphi(a) = \varphi(b) = \varphi(c) = [0]$. We compare reduction sequences from the term $f(g(a))$ by the E-strategy with and without evaluated flags.

For the function *red* in the original definition of the E-strategy without evaluated flags, it is reduced as follows:

$$\begin{array}{ll} \mathit{red}(f(g(a))) & \\ = \mathit{eval}(f(g(a)), & [1,0]) \\ = \mathit{eval}(f(\mathit{red}(g(a))), & [0]) \\ = \mathit{eval}(f(g(b)), & [0]) \\ = \mathit{red}(g(b)) & \\ = \mathit{eval}(g(b), & [0,1]) \\ = \mathit{red}(c) = \cdots = c & \end{array}$$

where $\mathit{red}(g(a)) = \mathit{eval}(g(a), [0, 1]) = \mathit{eval}(g(a), [1]) = \mathit{eval}(g(b), []) = g(b)$. On the other hand, for the function *red* in the definition of the E-strategy with evaluated flags, the term $f(g(a))$ is reduced as follows:

$$\begin{array}{ll} \mathit{red}^*(f(g(a))) & \\ = \mathit{eval}^*(f(g(a)), & [1,0]) \\ = \mathit{eval}^*(f(\mathit{red}^*(g(a))), & [0]) \\ = \mathit{eval}^*(f(g^*(b^*)), & [0]) \\ = \mathit{red}^*(g^*(b^*)) & \\ = g^*(b^*) & \end{array}$$

Since the term $g(b)$ is marked, the term cannot be reduced in the last step of the above reduction sequence.

It is desired that evaluated terms by the E-strategies with and without evaluated flags are same, i.e. $\mathit{red}(t) = \mathit{erase}(\mathit{red}^*(t))$. We show that it holds if *red* is safe.

Theorem 3.2.5 *If red is safe, $\mathit{red}(t) = \mathit{erase}(\mathit{red}^*(t))$ for each $t \in T$.*

Proof. We only show a proof sketch. The complete proof is similar to that of the literature [NMOF98]. The main difference between the E-strategy with and without evaluated flags is the treatment of evaluated term. For the E-strategy with evaluated flags, $\mathit{red}^*(t) = t$ if $t = \mathit{red}^*(s)$ for some term $s \in T^*$ because the root symbol of t is marked from the definition of red^*. On the other hand the same thing holds for a safe E-strategy, i.e. $\mathit{red}(t) = t$ if $t = \mathit{red}(s)$ for some term $s \in T$ from Theorem 3.1.2. Hence we can say that in the two definition of the E-strategies each term is evaluated into the same term if *red* is safe. □

3.3 Simplifying local strategies

There are infinitely many local strategies for a function symbol because the definition of local strategy does not restrict duplication of an element in a list. However for safe E-strategy map we can decrease the number of local strategies finitely. In this section we show how to simplify local strategies.

Theorem 3.3.1 *Let φ be a safe E-strategy map. A local strategy $\varphi(f) = [i_1, \ldots, i_n]$ where $i_j = i_k \neq 0$ and $j < k$ can be replaced by a local strategy $[i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_n]$ with no effect on the sequence of rewrites.*

Proof. When $eval(f(t_1, \ldots t_m), [i_k, \ldots, i_n])$, the term t_{i_k} is already evaluated because $i_j = i_k$ and $j < k$. Since $red(t_{i_k}) = t_{i_k}$ from Theorem 3.1.2, $eval(f(t_1, \ldots t_m), [i_{k+1}, \ldots, i_n])$. Hence removing the element i_k has no effect on the sequence of rewrites. □

This property enables us to delete a non-zero element if the same element also occurs before it.

Theorem 3.3.2 *Let φ be a safe E-strategy map. $\varphi(f) = [\ldots, 0]$ for each $f \in \Sigma$. A local strategy $\varphi(f) = [i_1, \ldots, i_j, 0, 0, i_{j+3} \ldots, i_n]$ can be replaced by a local strategy $[i_1, \ldots, i_j, 0, i_{j+3} \ldots, i_n]$ with no effect on the sequence of rewrites.*

Proof. Trivial. □

This property enables us to delete 0 if the immediately before element is also 0.

Example 3.3.3 Consider TRS R_{n_1} and φ where $\varphi(+) = [2, 0, 2, 0, 1], \varphi(s) = [1], \varphi(0) = []$

$$R_{n_1} = \left\{ \begin{array}{l} +(x, 0) \to x \\ +(x, s(y)) \to s(+(x, y)) \end{array} \right.$$

Since φ is safe (we will be able to prove it by a result of the next section), we can simplify the local strategy $\varphi(+) = [2, 0, 2, 0, 1]$ to $\varphi(+) = [2, 0, 0, 1]$ from Theorem 3.3.1 and do further to $\varphi(+) = [2, 0, 1]$ from Theorem 3.3.2 with no effect on the sequence of rewrites.

3.4 Defining safe local strategies

In this section we give a method for obtaining local strategies which satisfies the safety condition. It is trivial that an E-strategy is safe if each local strategy list ends in 0. The properties in the above sections about the evaluated flags and simplifying local strategies had been proved on this condition in the literatures [NMOF98, Eke98]. We can give a weaker condition on which an E-strategy is safe by analyzing the occurrence of variables in the left-hand sides of the rules of a given TRS.

Definition 3.4.1 For a TRS R a map $V_R : \Sigma \to \mathcal{P}(N)$ is defined as follows:

$$\begin{array}{l} LV(t) \overset{def}{=} \{(t)_p \in V(t) \mid \forall q \in O_V(t), (t)_p = (t)_q \Rightarrow p = q.\} \\ V_R(f) \overset{def}{=} \left\{ i \in \{1, \ldots, ar(f)\} \mid \forall f(\vec{l_i}) \to r \in R. l_i \in LV(f(\vec{l_i})) \right\} \end{array}$$

where $LV(t)$ is the set of linear variables in t. We call an argument in $V_R(f)$ a variable argument of f.

By the following property about variable arguments, we can obtain another method for simplifying local strategies.

Lemma 3.4.2 *[NO00] Let $t = f(\vec{t_k})$ and $i \in V_R(f)$. If a term t is redex, the term $t' = f(\ldots, t_{i-1}, s, t_{i+1}, \ldots)$ is also redex for any term s.*

Theorem 3.4.3 *Let φ be an E-strategy map. A local strategy $\varphi(f) = [\ldots, 0, i_1 \ldots, i_n, 0]$ where $i_k \in V_R(f)$ for $1 \leq k \leq n$ can be replaced by a local strategy $\varphi(f) = [\ldots, 0, i_1 \ldots, i_n]$ with no effect on the sequence of rewrites.*

Proof. When $eval(t, [i_1, \ldots, i_n, 0])$ where $(t)_\varepsilon = f$, the term t is not a redex because 0 was removed from the list. And when $eval(t', [0])$, the term t' is also not a redex because each i_k is a variable argument and Lemma 3.4.2 holds. Hence $eval(t', [0]) = eval(t', []) = t'$ and removing the last element 0 has no effect on the sequence of rewrites. $\square$

We prove that an E-strategy map is also safe even if any variable argument i is evaluated after the whole term, i.e. $\varphi(f) = [\ldots, 0, i]$.

Theorem 3.4.4 *Let R be a TRS and φ an E-strategy map. φ is safe if it satisfies the following condition,*

Condition: $\forall f \in D(R).\varphi(f) = [i_1, \ldots i_n]$ *such that* $\exists j.i_j = 0, \forall k > j.i_k \in V_R(f)$.

Proof. We prove that a term t is not a redex if $t = red(s)$ by induction on the structure of t. It is trivial if t is a variable or the root symbol of t is a constructor symbol, i.e. $(t)_\varepsilon \notin D(R)$. Now we assume $t \equiv f(\vec{t_i})$ and $f \in D(R)$. From the definition of red, there are terms $\vec{s_i}$ such that

$$eval(f(\vec{s_i}), \varphi(f)) = \cdots = eval(t, []) = t.$$

From the assumption, $\varphi(f)$ must have 0 such that all elements after the 0 are variable arguments. Thus,

$$eval(f(\vec{s_i}), \varphi(f)) = \cdots = eval(f(\vec{u_i}), [0, i_{j+1}, \ldots, i_n]) = \cdots = eval(t, []).$$

Eliminating 0 from the list means that matching $f(\vec{u_i})$ with left-hand sides fails. Hence, $f(\vec{u_i})$ is not a redex. From $i_{j+1}, \ldots, i_n \in V_R(f)$ and Lemma 3.4.2, we conclude that t is not a redex. $\square$

It is easy to check whether the condition of this theorem holds or not. The condition that each local strategy list ends in 0 is of course stronger than the condition of Theorem 3.4.4. Hence we can generalize some properties proposed in the literatures [NMOF98, Eke98] as follows.

Corollary 3.4.5 *Let φ be an E-strategy map satisfying the condition of Theorem 3.4.4. For each term $t \in T$, $red(t) = erase(red^*(t))$.*

Corollary 3.4.6 *Let φ be an E-strategy map satisfying the condition of Theorem 3.4.4. Any evaluated term cannot be further rewritten, i.e. $red(red(t)) = red(t)$ for any t.*

Corollary 3.4.7 *Let φ be an E-strategy map satisfying the condition of Theorem 3.4.4. A local strategy $\varphi(f) = [i_1, \dots, i_n]$ where $i_j = i_k \neq 0$ and $j < k$, can be replaced by a local strategy $[i_1, \dots, i_{k-1}, i_{k+1}, \dots, i_n]$ with no effect on the sequence of rewrites.*

Corollary 3.4.8 *Let φ be an E-strategy map satisfying the condition of Theorem 3.4.4. A local strategy $\varphi(f) = [i_1, \dots, i_j, 0, 0, i_{j+3} \dots, i_n]$ can be replaced by a local strategy $[i_1, \dots, i_j, 0, i_{j+3} \dots, i_n]$ with no effect on the sequence of rewrites.*

From these properties we can conclude that although there are infinitely many E-strategies map for a signature Σ, we consider essentially only finite cases if an E-strategy is safe.

Example 3.4.9 Consider the binary function symbol $+$ in TRS R_{n_1}, all local strategies we should consider are following:

$$(0,1), (2,0), (0,2,0),$$

$$\begin{array}{cccccccccccc}
(& & 1, & 0, & 2, & 0 &), (& & 2, & 0, & 1 &), \\
(& & 1, & & 2, & 0 &), (& & 2, & & 1 &), \\
(& 0, & 1, & 0, & 2, & 0 &), (& 0, & 2, & 0, & 1 &), \\
(& 0, & 1, & & 2, & 0 &), (& 0, & 2, & & 1 &)
\end{array}$$

Note that the first argument of $+$ is a variable argument for TRS R_{n_1}.

3.5 Normalizability

The literature [Nag99] discussed the normalizability of the E-strategy. For an ARS $(A, \rightarrow)$, a relation $\rightarrow_s$ is a reduction strategy for $\rightarrow$ if $\rightarrow_s \subseteq \rightarrow^+$ and each normal form of $\rightarrow_s$ is also a normal form of $\rightarrow$. A reduction strategy $\rightarrow_s$ is normalizing if for each $a \in A$ having a normal form, there is no infinite sequence $a \rightarrow_s a_1 \rightarrow_s a_2 \rightarrow_s \cdots$. In the literature [Nag99] the E-strategy is formalized by an ARS $(T \times N^*, \rightarrow_\varphi)$, in which an element is a pair of a term t and a position p, which means that the subterm $t|_p$ has to be evaluated next. The definition of the E-strategy in [Nag99] is following:

Definition 3.5.1 [Nag99] We define Σ_L, V_L and T_L as $\Sigma_L = \{f_l \mid f \in \Sigma, l \in List(\mathcal{N})\}$, $V_L = \{x_{nil} \mid x \in V\}$ and $T_L = T(\Sigma_L, V_L)$. We extend the E-strategy map from T to T_L as

$$\varphi(f(t_1, \dots t_n)) \equiv f_{\varphi(f)}(\varphi(t_1), \dots, \varphi(t_n)).$$

The map $\mathbf{e} : T_L \rightarrow T$ erases all lists of function symbols and variables in a term defined as follows:

$$\mathbf{e}(f_l(t'_1, \dots t'_n)) \equiv f(\mathbf{e}(t'_1), \dots, \mathbf{e}(t'_n)).$$

Definition 3.5.2 [Nag99] Let R be a TRS and φ an E-strategy map. An E-strategy rewrite relation (or φ-rewrite relation) $\rightarrow_\varphi$ is a binary relation on $T_L \times \mathcal{N}^*$ defined as follows: $\langle t, p\rangle \rightarrow_\varphi \langle s, q\rangle$ if and only if $p \in O(t)$ and one of the following conditions is satisfied:

1. $(t)_p = e_{nil}$, $s \equiv t$ and $p = q \cdot i$ for some i,

2. $(t)_p = e_{0::l}$, $t|_p \equiv l'\theta$, $\mathbf{e}(l') \equiv l$, $s \equiv t[\varphi(r)\theta]_p$ for some θ and $l \to r \in R$, and $q = p$.

3. $t|_p \equiv e_{0::l}(t_1, \ldots, t_n)$, $\mathbf{e}(t|_p)$ is not a redex, $s \equiv t[e_l(t_1, \ldots, t_n)]_p$, $q = p$,

4. $t|_p \equiv e_{i::l}(t_1, \ldots, t_n)$ with $i > 0$, $s \equiv t[e_l(t_1, \ldots, t_n)]_p$ and $q = p \cdot i$,

The normalizability of $\to_\varphi$ has been discussed in [Nag99]. In order for an E-strategy to become a normalizing strategy, it is necessary to introduce some notions for a TRS and an E-strategy. For a fresh symbol Ω, an element of a set $T(\Sigma \cup \Omega, V)$ (or T_Ω for short) is called an Ω-term. A position p of a term $t \in T$ is an index if there is no $s \in T$ such that $t[\Omega]_p \to_R^* s$ and s is a normal form. The set of indices of t is defined by $I(t)$. We define the index reduction $\to_I$ as follows: $s \to_I t$ if and only if $s \to_p t$ and $p \in I(s)$. The following property holds.

Theorem 3.5.3 *[HL91] Let R be an orthogonal TRS. The index reduction $\to_I$ is a normalizing strategy for $\to_R$.*

Here we introduce only needed notations for obtaining a normalizing strategy by the E-strategy. More details are shown in [HL91, Nag99].

Definition 3.5.4 [Nag99] A TRS R is index-transitive if for every term $t \in T$, $p \in I(t)$ and $q \in I(t|_p)$ imply $p.q \in I(t)$.

Next we introduce the condition of an E-strategy map which is called the carefulness. In the above section, we analyze variable arguments for obtaining a safe E-strategy. A variable argument is a variable for all left-hand sides of R and we define non-variable arguments before 0 and variable arguments after 0. For the carefulness, we analyze variable occurrences of a defined symbol for each left-hand side one by one.

Definition 3.5.5 [Nag99] Let R be a TRS. An E-strategy map φ is careful if it satisfies the following condition: for each $f \in \Sigma$

i. $\varphi(f)$ contains $1, \ldots, ar(f)$,
ii. the last element of $\varphi(f)$ is 0 if $f \in D(f)$,

and for each $l \to r \in R$ such that $(l)_\varepsilon = f$ and $\varphi(f) = [i_1, \ldots, i_n]$, there exists k such that

1. $i_k = 0$,
2. $1 \leq \forall j \leq k, i_j = 0$ or $i_j \in I(l)$,
3. $k < \forall j \leq n, i_j = 0$ or $l|_{i_j} \in V$.

For an index-transitively TRS and a careful E-strategy map, the following property holds.

Proposition 3.5.6 *[Nag99] Let R be an index-transitive orthogonal TRS and φ a careful E-strategy map. Then $\to_\varphi$ is normalizing.*

Example 3.5.7 Consider the following TRS.

$$R = \begin{cases} f(x, a, b) \to x \\ f(x, b, y) \to x \\ g(x) \to a \\ c \to g(c) \end{cases}$$

If we define local strategies as $\varphi(f) = [2, 0, 3, 0, 1, 0]$, $\varphi(g) = [0, 1, 0]$, $\varphi(c) = [0]$ and $\varphi(a) = \varphi(b) = nil$. Since R is orthogonal and index-transitive and φ is careful (**Proof :**See [Nag99]), then $\to_\varphi$ is normalizing from Proposition 3.5.6.

It is easy to say that if $\to_\varphi$ is normalizing, $red(t)$ terminates for each t having a normal form. In the previous section we define the safety condition to generalize the condition (i) and (ii), which is occurred in the first half of the definition of the carefulness. Even if the condition (i) and (ii) is replaced with the safety condition, we can also show the above normalizing property straightforwardly. For the above example we can remove the last 0 from the local strategy $\varphi(f) = [2, 0, 3, 0, 1, 0]$ with no effect on the sequence of rewrites from Theorem 3.4.3.

Chapter 4

Correctness and strictness analysis

In this chapter we consider one of the main topics of this thesis: correctness and strictness analysis of the E-strategy.

We first introduce the notion of context-sensitive rewriting [Luc98], which is a restriction of rewriting formalized by a replacement map on function symbols. Although in the E-strategy a local strategy describes order of evaluation for each function symbol, in context-sensitive rewriting a replacement set describes which arguments can be evaluated or not for each function symbol. They are formalized by a list and a set respectively. Context-sensitive rewriting is not a strategy. Although a strategy chooses a redex for a term, context-sensitive rewriting chooses a subset of redeces for a term. It is easy for context-sensitive rewriting to analyze theoretical properties, which are termination, confluence and so on, rather than the E-strategy and many useful properties are proposed in [Luc96, Luc98, Luc00, Luc01a, Luc01b, GM99, Zan97, FR99]. We clarify a relation between the E-strategy and context-sensitive rewriting to obtain useful properties of the E-strategy through those of context-sensitive rewriting.

We consider a method for obtaining a replacement map μ of context-sensitive rewriting from a given E-strategy map φ. It is easy to obtain a transformation from an E-strategy map to a replacement map because a set is simpler than a list. We show some relation between a given E-strategy map φ and the corresponding replacement map, denoted by μ^{φ}.

Next we investigate μ-correctness of E-strategies. We discussed in the previous chapter what conditions are needed for evaluated terms for holding some basic properties of the E-strategy and did not discuss the meaning of evaluated terms as an answer. Hence we need to discuss more strong condition for evaluated terms. The literatures [Nag99, NO00, Pol01] proposed such conditions, on which an E-strategy is correct, i.e. each evaluated term is always a normal form. Although a normal form leaves nothing to be desired as an answer, it is too strong to deal with the example of the infinite list:

$$R_{l_1} = \begin{cases} \mathit{inf} \to 0 : \mathit{inf} \\ \mathit{hd}(x : y) \to x \\ \mathit{tl}(x : y) \to y \end{cases}$$

because there is a term which has no normal form, like a term *inf*. If *red* is correct, reduction does not terminate not only for *inf* but also for $\mathit{hd}(\mathit{tl}(\mathit{inf}))$ which has a normal form 0. Normal forms of context-sensitive rewriting, called μ-normal forms, may be considered as a suitable compromise because we can choose a replacement map for a given TRS flexibly and there are some useful properties about μ-normal forms proposed in [Luc98, Luc00, NF01] and the following section.

We also investigate strictness of E-strategies. In lazy functional languages, a function is strict in a certain argument if the eager evaluation of that argument does not change the termination behavior of the program. We define the notion of strictness for term rewriting systems and propose a method for analyzing strict arguments. If we find that a function symbol is strict in an argument by our method of strictness analysis, we can evaluate it eagerly with no effect on termination. By evaluating an argument eagerly we gain some efficiency in rewrite steps.

By correctness and strictness analyses, we give methods to obtain a suitable E-strategy for a given context-sensitive rewriting. Our analyses proposed in this chapter can apply any TRS and it is easy to implement them. Hence we can use these methods for computing a default strategy, which the system (such as CafeOBJ) gives us automatically when we omit to write local strategies.

4.1 Context-sensitive rewriting

In this section we introduce context-sensitive rewriting, which is an ARS $(T(\Sigma, V), \rightarrow_\mu)$ where $T(\Sigma, V)$ is a set of terms and $\rightarrow_\mu$ is a binary relation on $T(\Sigma, V)$ defined by a term rewriting system R and a replacement map μ. First we introduce some definitions of context-sensitive rewriting: a replacement map, an active position and a context-sensitive rewrite relation.

In context-sensitive rewriting we decide which argument can be replaced and only redeces under such arguments are permitted to be reduced. For example if we select the first argument of $+$ and s as replacement arguments. For $+(s(+(0, s(+(0, 0)))), +(s(0), 0))$, only the redex $+(0, s(+(0, 0)))$ can be reduced because the other redeces are under second arguments of $+$. Replacement arguments are formalized by a map which takes a function symbol and returns a set of replacement arguments.

Definition 4.1.1 A replacement map μ is a map from Σ to $\mathcal{P}(\mathcal{N})$ such that $\mu(f) \subseteq \{1, \ldots, ar(f)\}$ for each $f \in \Sigma$.

Definition 4.1.2 Let μ be a replacement map and t a term. A set of active positions $O_\mu(t) \subset \mathcal{N}_+^*$ of a term t is defined as follows:

$$O_\mu(t) = \begin{cases} \{\varepsilon\} & \text{if} \quad t \in V \\ \{\varepsilon\} \cup \bigcup_{i \in \mu(f)} \{i.p \in \mathcal{N}_+^* \mid p \in O_\mu(t_i)\} & \text{if} \quad t \equiv f(t_1, \ldots, t_n). \end{cases}$$

The context-sensitive rewrite relation, denoted by $\rightarrow_\mu$, (μ-rewrite relation) is a rewrite relation whose redex positions is restricted to active positions (Fig. 4.1).

Definition 4.1.3 Let R be a TRS and μ a replacement map.

$$s \rightarrow_\mu t \overset{\text{def}}{\iff} \exists p \in O_\mu(s), s \rightarrow_p t.$$

The trivial replacement map $\mu_\top$ is defined as $\mu_\top(f) = \{1, \ldots, ar(f)\}$ for each $f \in \Sigma$. Note that $\rightarrow_{\mu_\top} = \rightarrow_R$. A normal form of μ-rewrite relation $\rightarrow_\mu$ is called a μ-normal form.

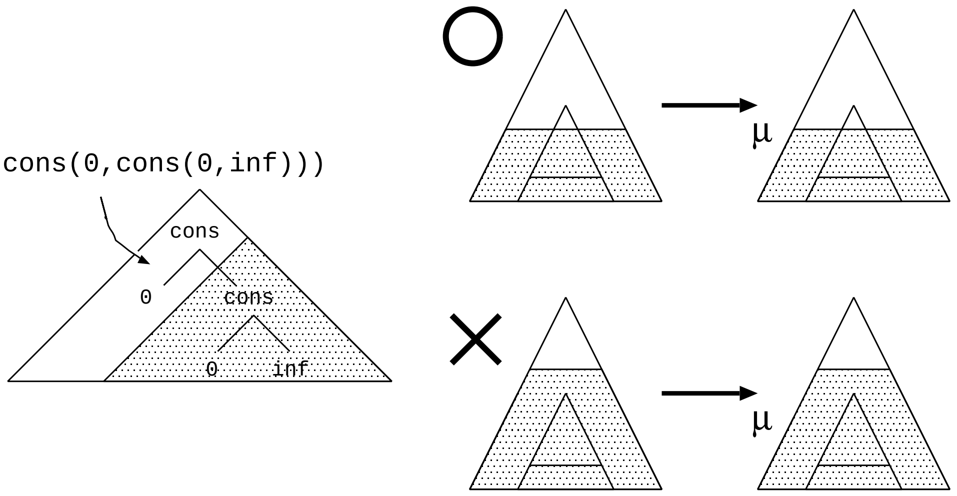

Figure 4.1: Active positions for $\mu(cons) = \{1\}$

Example 4.1.4 We consider the following TRS R_{l_1} of an infinite list.

$$R_{l_1} = \begin{cases} inf \to 0 : inf \\ hd(x : y) \to x \\ tl(x : y) \to y \end{cases}$$

where : is a binary symbol and we write $t_1 : t_2 : t_3 \cdots$ instead of $:(t_1, :(t_2, :(t_3, \cdots)))$ for a simple view. Of course R_{l_1} is not terminating, i.e. there is an infinite reduction sequence, e.g. $hd(tl(inf)) \to_{R_{l_1}} hd(tl(0 : inf)) \to_{R_{l_1}} hd(tl(0 : 0 : inf)) \to_{R_{l_1}} \cdots$. We can avoid such an infinite reduction sequence by defining a replacement map μ as $\mu(:) = \mu(hd) = \mu(tl) = \{1\}$. Since each position under the second argument of : is not active, the redex inf in $hd(tl(0 : inf))$ cannot be rewritten. The following is an example showing that context-sensitive rewriting can avoid such an infinite reduction sequence.

$$\begin{tikzcd}
hd(tl(\underline{inf})) \arrow[r, "\mu"'] & hd(\underline{tl(0:\underline{inf})}) \arrow[r, "\not\to_\mu", phantom] \arrow[d, "\mu"] & hd(tl(0:0:inf)) & \\
 & hd(\underline{inf}) \arrow[r, "\mu"'] & \underline{hd(0:\underline{inf})} \arrow[r, "\not\to_\mu", phantom] \arrow[d, "\mu"] & hd(0:0:inf) \\
 & & 0 &
\end{tikzcd}$$

4.2 Termination of the E-strategy

The definition of E-strategy maps is very similar to that of replacement maps. The differences are the range of the map and whether the element 0 is included or not.

The E-strategy map	**The replacement map**
$\varphi : \Sigma \to List(\mathcal{N})$	$\mu : \Sigma \to \mathcal{P}(\mathcal{N})$
$\forall i \in \varphi(f).0 \leq i \leq ar(f)$	$\forall i \in \mu(f).1 \leq i \leq ar(f).$

It is easy to obtain the corresponding replacement map for a given E-strategy map by forgetting the order of elements and removing 0.

Definition 4.2.1 [Luc00] For an E-strategy map φ, the replacement map μ^{φ} is defined as follows: $\mu^{\varphi}(f) = \{i_1, \ldots, i_n\} \setminus \{0\}$ for each function symbol f with $\varphi(f) = [i_1, \ldots, i_n]$.

Trivially, any E-strategy is covered with term rewriting, i.e. if $t = red(s)$, there is a reduction sequence $s \to_R^* t$. We can also say that if R terminates, so does $red(t)$ for any t. In this section we show that some E-strategies are also covered with context-sensitive rewriting.

Lemma 4.2.2 *Let μ be a replacement map. $f(t_1, \ldots, t_n) \to_\mu^* f(t'_1, \ldots, t'_n)$ if $t_i \to_\mu^* t'_i$ for $i \in \mu(f)$ and $t_i \equiv t'_i$ for $i \notin \mu(f)$.*

Proof. We prove that $f(t_1, \ldots, t_n) \to_\mu f(t'_1, \ldots, t'_n)$ if $t_i \to_\mu t'_i$ for a $i \in \mu(f)$ and $t_j \equiv t'_j$ for each $j \neq i$. There is a position $p \in O_\mu(t)$ such that $t_i \to_p t'_i$ from the definition of $\to_\mu$. $i.p \in O_\mu(f(t_1, \ldots, t_n))$ because $i \in \mu(f)$. Hence $f(t_1, \ldots, t_n) \to_\mu f(t'_1, \ldots, t'_n)$. □

Theorem 4.2.3 *[Luc00] Let φ be an E-strategy map and μ a replacement map such that $\mu^{\varphi} \sqsubseteq \mu$. If $t = red(s)$, there is a reduction sequence $s \to_\mu^* t$.*

Proof. We prove the claim by induction and the number of calls of *red* at the root position in the reduction sequence $red(s) = red(t_1) = red(t_2) = \cdots = red(t_n) = t$.

(Base case) We assume that applying the function *red* at the root symbol is only once, i.e. $n = 0$ for the above sequence. We prove the claim by induction on the structure of s. If $s \in V$, $red(s) = s$ and trivially $s \to_\mu^* s$. We assume that $s \equiv f(\vec{s_i})$. If $\varphi(f) = []$, $red(s) = eval(s, []) = s$. If $\varphi(f) = [i_1, \ldots, i_m] \neq []$, there should be the following reduction sequence:

$$\begin{array}{lll} red(s) & = eval(f(s_1, \ldots, s_n), & [i_1, i_2, \ldots, i_m]) \\ & = eval(f(s_1, \ldots, red(s_{i_1}), \ldots, s_n), & [i_2, \ldots, i_m]). \\ & = eval(f(s_1, \ldots, s'_{i_1}, \ldots, s_n), & [i_2, \ldots, i_m]) \\ & = \cdots = & \\ & = eval(f(s'_1, \ldots, s'_n), & []) \\ & = f(s'_1, \ldots, s'_n) & \end{array}$$

where $s'_{i_k} = red(s_{i_k})$ for each i_k and $s_i \equiv s'_i$ for $i \neq i_k$ for any k. From the induction hypothesis, $s_{i_k} \to_\mu^* s'_{i_k}$. Hence we obtain $s \to_\mu^* f(s'_1, \ldots, s'_n) \equiv t$ from Lemma 4.2.2.

(I.S.) We assume that $red(s) = red(t_1) = \cdots = red(t_n) = t$ for $n \neq 0$. Again we prove the claim by induction on the structure of s. If $s \in V$, $red(s) = s$ and trivially $s \to_\mu^* s$. We assume that $s \equiv f(\vec{s_i})$. There should be the following reduction sequence:

$$\begin{array}{ll} red(s) & = eval(f(s'_1, \ldots, s'_n), \quad [i_k, i_{k+1}, \ldots, i_m]) \\ & = red(s') = \cdots = t \end{array}$$

where $i_k = 0$, $s_i \to_\mu^* s'_i$ if $i = i_j$ for some $j < k$ and $s_i \equiv s'_i$ if $i \neq i_j$ for any $j < k$. $s \to_\mu^* f(s'_1, \ldots, s'_n)$ from the same reason as above, $f(s'_1, \ldots, s'_n) \to_\mu^* s'$ from $\to_\varepsilon \subset \to_\mu$ and $s' \to_\mu^* t$ from the induction hypothesis. Therefore we conclude that $s \to_\mu^* t$ if $t = red(s)$.□

One of the most useful property of the replacement map μ^φ is about termination. For an E-strategy map φ, we can prove that *red* is terminating if the context-sensitive rewrite relation $\to_{\mu^\varphi}$ is terminating

Theorem 4.2.4 *[Luc01b] Let φ be an E-strategy map and μ a replacement map such that $\mu^\varphi \sqsubseteq \mu$. If $\to_\mu$ terminates, $red(t)$ also terminates for each $t \in T$.*

Proof. We prove the claim by induction of the structure of a term t. If $t \in V$, it is trivial that $red(t)$ terminates and returns t immediately. We assume that $t \equiv f(t_1, \ldots, t_n)$. From the induction hypothesis, $red(t_i)$ terminates for each i. There are two cases of the reduction sequence of $red(t)$.

Case 1.)

$$\begin{aligned} red(t) &= eval(f(t_1, \ldots, t_n), && [i_1, i_2, \ldots, i_m]) \\ &= eval(f(t_1, \ldots, red(t_{i_1}), \ldots, t_n), && [i_2, \ldots, i_m]) \\ &= \cdots = \\ &= eval(f(t'_1, \ldots, t'_n), && []) \\ &= f(t'_1, \ldots, t'_n) \end{aligned}$$

In this case it is trivial that $red(t)$ terminates.

Case 2.)

$$\begin{aligned} red(t) &= eval(f(t_1, \ldots, t_n), && [i_1, i_2, \ldots, i_m]) \\ &= \cdots = \\ &= eval(f(t'_1, \ldots, t'_n), && [0, i_k, \ldots]) \\ &= red(t') \end{aligned}$$

In this case, we prove the claim by induction of the context-sensitive rewrite sequence from t. If t is a μ-normal form, it is inconsistent with the fact that $f(t'_1, \ldots, t'_n)$ is a redex in the last step $eval(f(t'_1, \ldots, t'_n), [0, i_k, \ldots]) = red(t')$. If t is not a μ-normal form, $t \to^+_\mu t'$ from Theorem 4.2.3. From the induction hypothesis, $red(t')$ terminates.□

Example 4.2.5 Consider TRS R_{l_1}.

$$R_{l_1} = \begin{cases} inf \to 0 : inf \\ hd(x : y) \to x \\ tl(x : y) \to y \end{cases}$$

Let $\varphi(:) = [1]$ and $\varphi(hd) = \varphi(tl) = [1, 0]$. Then, $\mu^\varphi(:) = \mu^\varphi(hd) = \mu^\varphi(tl) = \{1\}$. By methods for proving termination of context-sensitive rewriting in the literatures [Luc96, Zan97, GM99], we can prove that $\to_{\mu^\varphi}$ is terminating. Hence $red(t)$ is terminating from Theorem 4.2.4.

Since we showed the property about termination, it is natural that the question about confluence is occurred. Unfortunately, the fact that μ-rewriting is confluent does not imply the fact that an evaluated term of a term is unique even if $\mu^\varphi \sqsubseteq \mu$. The reason is that an evaluated term may be a term on the half way of μ-rewriting, i.e. may not be a μ-normal form. In the next section we propose the conditions on which each evaluated term is always a μ-normal form.

Example 4.2.6 Consider TRS $R = \{a \to b, a \to f(a), f(b) \to b\}$ and $\varphi(a) = [0]$, $\varphi(b) = nil$ and $\varphi(f) = [0, 1]$. The corresponding replacement map μ^φ is trivial. Although $\to_{\mu^\varphi}$ is confluent, an evaluated term is not unique, such as $red(a) = eval(a, [0]) = red(b) = eval(b, nil) = b$ and $red(a) = eval(a, [0]) = red(f(a)) = eval(f(a), [0, 1]) = eval(f(a), [1]) = eval(f(b), nil) = f(b)$.

4.3 Correctness

An E-strategy map φ is called correct if red is a correct strategy, i.e. each evaluated term is a normal form. If $\varphi(f) = [1, \ldots, ar(f), 0]$ for each $f \in \Sigma$, the E-strategy is trivially correct and identical with the leftmost innermost strategy. The literature [Nag99] gives a condition on which an E-strategy is correct.

Proposition 4.3.1 *[Nag99] An E-strategy map φ is correct if the following conditions are satisfied:*

- *$\varphi(f)$ contains $1, \ldots, ar(f)$ for any $f \in \Sigma$,*
- *the last element of $\varphi(f)$ is 0 for any $f \in D(R)$.*

From Proposition 4.3.1 we can give a proof of correctness of the above leftmost innermost E-strategy since $[1, \ldots, ar(f), 0]$ satisfies the conditions in Proposition 4.3.1. There are other studies about the correctness of E-strategy [NO00][NF01][Pol01].

Although the correctness is quite satisfactory for shapes of evaluated terms, the condition may be too strong for some TRSs. Now we assume that for a TRS R a term t has not a normal form, i.e. there is no normal form s such that $t \to_R s$. For any correct E-strategy the evaluation of t does not terminate because if $red(t)$ terminates, the result must be a normal form and it is a reduced term from t. In order to deal with such TRSs we need to propose a condition which is weaker than the correctness but has some meaning. In this section, as such a condition we propose the notion of the μ-correctness for the E-strategy where μ is a replacement map in context-sensitive rewriting.

Definition 4.3.2 An E-strategy map φ is μ-correct if each evaluated term is a μ-normal form.

Before giving a method of defining a μ-correct E-strategy map, we show the following useful properties for μ-correct E-strategies.

Theorem 4.3.3 *Let φ be an E-strategy map and μ a replacement map such that $\mu^\varphi \sqsubseteq \mu$. If φ is μ-correct and $\to_\mu$ has the unique normal form property, an evaluated term is unique if it exists.*

Proof. We assume that $t = red(s)$ and $t' = red(s)$ for some term s. From Theorem 4.2.3 $s \to_\mu^* t$ and $s \to_\mu^* t'$. Since $\to_\mu$ has the unique normal form property, $t \equiv u \equiv t'$ because φ is μ-correct, i.e. t and t' are in μ-normal form. □

Theorem 4.3.4 *Let φ be an E-strategy map and μ a replacement map such that $\mu^\varphi \sqsubseteq \mu$. If φ is μ-correct and $\to_\mu$ has the unique normal form property and is terminating, there exist a unique evaluated term for any term.*

Proof. From Theorem 4.2.4 and 4.3.3. □

Corollary 4.3.5 *Let φ be an E-strategy map and μ a replacement map such that $\mu^\varphi \sqsubseteq \mu$. If φ is μ-correct and $\to_\mu$ is confluent, an evaluated term is unique if it exists.*

Corollary 4.3.6 *Let φ be an E-strategy map and μ a replacement map such that $\mu^\varphi \sqsubseteq \mu$. If φ is μ-correct and $\to_\mu$ is confluent and is terminating, there exist a unique evaluated term for any term.*

4.3.1 μ-correct E-strategies

If a local strategy $\varphi(f)$ is defined as $[i_1, \ldots, i_n, 0]$ for each $\mu(f) = \{i_1, \ldots, i_n\}$ (where $i_1 < \cdots < i_n$), the E-strategy map φ is μ-correct and *red* is identical with the leftmost innermost strategy of the context-sensitive rewriting $\to_\mu$. Hereafter we analyze which arguments can be evaluate lazily with keeping μ-correctness. We prove that an E-strategy map is μ-correct even if a variable argument i evaluates after the whole term, i.e. $\varphi(f) = [\ldots, 0, i]$.

Definition 4.3.7 Let φ be an E-strategy map and μ a replacement map. The function $\Phi : \mathcal{P}(\mathcal{N})^\Sigma \to \mathcal{P}(List(\mathcal{N})^\Sigma)$, which takes a replacement map and returns a set of E-strategy maps, is defined as follows: $\varphi \in \Phi(\mu)$ if and only if for any $f \in \Sigma$, $\varphi(f) = [i_1, \ldots, i_n]$, (1) $\mu(f) \subset \{i_1, \ldots, i_n\}$ and (2) $i_k = 0$ and $\{i_{k+1}, \ldots, i_n\} - \{0\} \subset V_R(f)$ for some k if $f \in D(R)$.

The condition of (1) means that each replacement argument $i \in \mu(f)$ must be reduced in the E-strategy. That of (2) guarantees φ to be safe. For proving the μ-correctness of $\varphi \in \Phi(\mu)$, we prepare the following lemma about μ-normal forms.

Lemma 4.3.8 *Let μ be a replacement map. A term $t \equiv f(t_1, \ldots t_n)$ is a μ-normal form if t is not a redex and t_i is a μ-normal form for any $i \in \mu(f)$.*

Proof. We assume that t is not a μ-normal form. There is a position $p \in O(t)$ such that p is active and $t|_p$ is a redex. Since t is not a redex, p is not empty, i.e. $p \neq \varepsilon$. From Definition 4.1.2 we can break p to $i.q$ where $i \in \mu(f)$ and $q \in O_\mu(t_i)$. Hence t_i has a redex subterm $t_i|_q \equiv t|_p$. It is inconsistent. □

Theorem 4.3.9 *If $\varphi \in \Phi(\mu)$, φ is μ-correct.*

Proof. We prove that a term t is a μ-normal form if $t = red(s)$ by induction on the structure of t. It is trivial for $t \in V$. We assume $t \equiv f(\vec{t_i})$. From the definition of *red*, there exist terms $\vec{s_i}$ such that

$$red(s) = \cdots = red(f(\vec{s_i})) = eval_\varphi(f(\vec{s_i}), \varphi(f)) = \cdots = eval_\varphi(t, []) = t.$$

From the assumption 2 and Theorem 3.4.4, φ is safe, i.e. the evaluated term t is not a redex. From the assumption 1, $t_i = red(s_i)$ if $i \in \mu(f)$ and from the induction hypothesis t_i is a μ-normal form. From Lemma 4.3.8 the term t is a μ-normal form. □

Example 4.3.10 Consider TRS R_{l_1}.

$$R_{l_1} = \begin{cases} \mathit{inf} \to 0 : \mathit{inf} \\ hd(x : y) \to x \\ tl(x : y) \to y \end{cases}$$

Let $\mu(:) = \mu(hd) = \mu(tl) = \{1\}$. We define an E-strategy map φ as an element of $\Phi(\mu)$. For example, $\varphi(:) = [1]$, $\varphi(hd) = \varphi(tl) = [1, 0]$. From Theorem 4.3.9, each evaluated term is a μ-normal form.

4.3.2 Properties for μ-normal form

Although a normal form is of course in μ-normal form, the reverse is not always true. Like as the term $0 : \mathit{inf}$ in the above example there exists a term which is in μ-normal form but not in normal form (Fig. 4.2). Hence, it is important to analyze what properties μ-normal forms have in context-sensitive rewriting. In this section we analyze relations between μ-normal forms and each of redeces, root-stable forms and normal forms. By combining these analyses with Theorem 4.3.9, we can obtain useful properties about evaluated terms of the E-strategy.

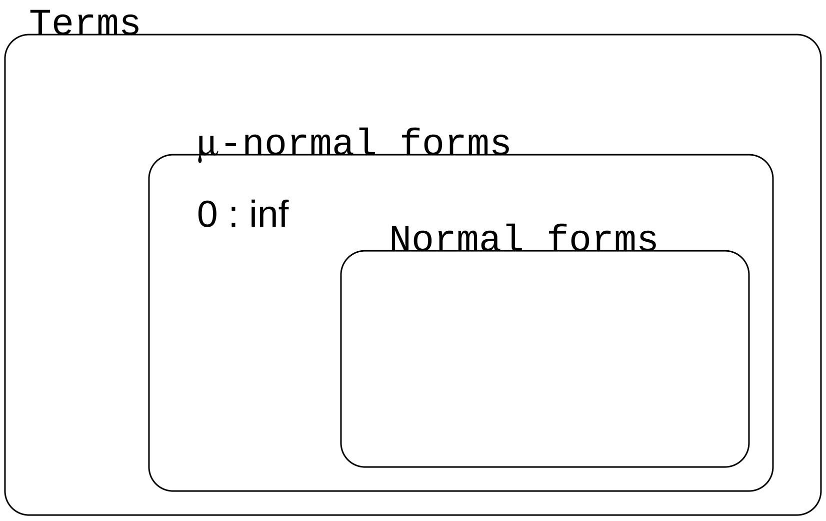

Figure 4.2: μ-normal form and normal form

Redex

We first show a trivial property of μ-normal form. For any replacement map μ we cannot give the context-sensitive rewriting which has not the root position of a given term as an active position. A position which is above an active position is also active, i.e. p is active in t if $p.q$ is active for some q. This implies that if the root position is not active, no active position can exist. Since such a restriction is meaningless, context-sensitive rewriting is defined as always being able to rewrite the root position (Fig. 4.3) .

Lemma 4.3.11 *Each redex is not a μ-normal form.*

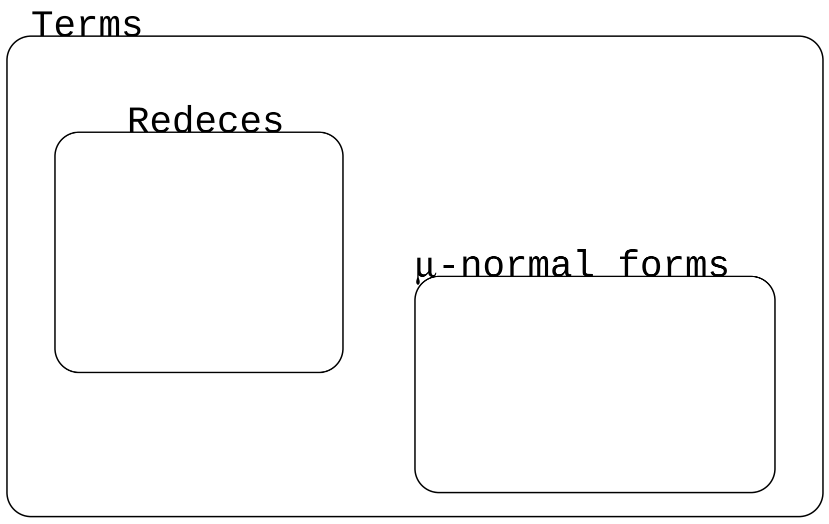

Figure 4.3: μ-normal form and redex

Proof. For any replacement map μ and term t, from Definition 4.1.2 the root position is always active. Hence any redex can be rewritten by context-sensitive rewriting. $\Box$

Root-stable form

In the paper [Luc98], an important replacement map whose name is the canonical replacement map μ_R is proposed. In the canonical replacement map, arguments where only variables occur in each left-hand side are restricted to non replacement arguments.

Definition 4.3.12 [Luc98] The canonical replacement map μ_R is defined as $\mu_R(f) = \{i \in \{1, \ldots, ar(f)\} \mid \exists l \to r \in R, \exists p \in T^V.(l)_p = f$ and $p.i \notin O_V(l).\}$.

Example 4.3.13 Note that both V_R and μ_R analyze occurrences of variables in left-hand sides, however $V_R(f)$ is not always the same as $\overline{\mu_R(f)} = \{1, \ldots, ar(f)\} - \mu_R(f)$. The definition of V_R is depend on the only root position of the left-hand sides. That of μ_R is depend on all positions of the left-hand sides. For the TRS $R = \{f(x, f(0, y)) \to f(x, y)\}$, $\overline{\mu_R(f)} = \emptyset \neq \{1\} = V_R(f)$.

When an argument is not in the canonical replacement set, i.e. $i \notin \mu_R(f)$, for any subterm $f(t_1, \ldots, t_n)$ of any left-hand side, t_i is a variable. It can be said that such an argument is not necessary for a redex for a left-linear TRS. Hence if all subterms except such arguments are reduced, the result whole term cannot become a redex (Fig. 4.4) .

Definition 4.3.14 [Luc98] The quasi ordering $\sqsubseteq$ on replacement maps is defined as $\mu \sqsubseteq \mu'$ if and only if $\mu(f) \subseteq \mu'(f)$ for each $f \in \Sigma$.

Proposition 4.3.15 *[Luc98] Let R be a left-linear TRS and μ a replacement map such that $\mu \sqsupseteq \mu_R$. Each μ-normal form is a root-stable form.*

From Proposition 4.3.15 we can obtain the following useful property about evaluated terms of the E-strategy.

Corollary 4.3.16 *Let R be a left-linear TRS and $\varphi \in \Phi(\mu_R)$. Each evaluated term is a root-stable form.*

Proof. From Theorem 4.3.9 and Proposition 4.3.15. □

Example 4.3.17 Again consider R_{l_1} and $\varphi(:) = [1]$, $\varphi(hd) = \varphi(tl) = [1, 0]$.

$$R_{l_1} = \begin{cases} inf \to 0 : inf \\ hd(x : y) \to x \\ tl(x : y) \to y \end{cases}$$

Corollary 4.3.16 guarantees each evaluated term to be a root-stable form.

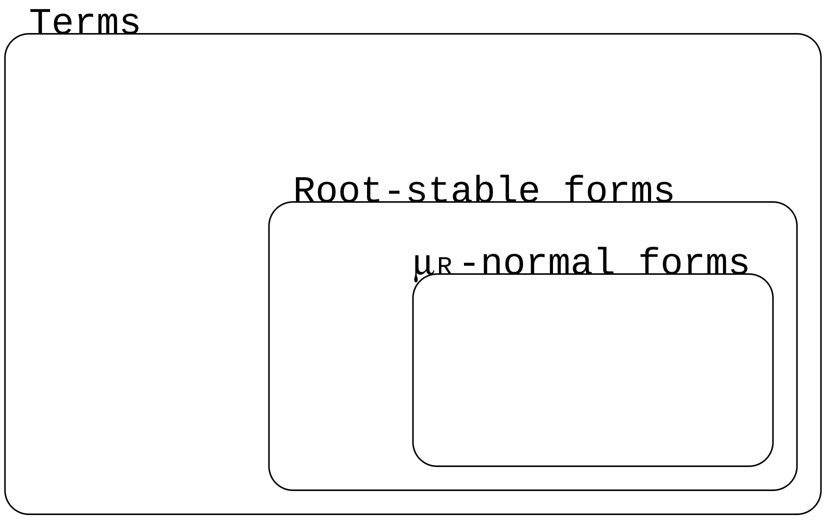

Figure 4.4: μ_R-normal form and root-stable form

Normal form

Moreover we propose a condition of a replacement map μ for normal forms. A mathematical function is called total on $\mathcal{D}$ when the function is defined for all elements of $\mathcal{D}$. We introduce the totality of function symbols for TRSs. For a function symbol $f \in \Sigma$, we call f a total function symbol if and only if a term $f(\ldots)$ is a redex if each argument is in normal form.

Definition 4.3.18 A function symbol f is total if and only if $f(t_1, \ldots, t_n)$ is a redex whenever each t_i is a normal form.

Since a variable is always a normal form, $f(x_1, \ldots, x_n)$ must be a redex if f is total. Moreover in order for $f(x_1, \ldots, x_n)$ to be a redex, there must be a rule $f(x_1, \ldots, x_n) \to r$ in a given TRS since only a variable or $f(x_1, \ldots, x_n)$ is a term of which $f(x_1, \ldots, x_n)$ is an instance.

Lemma 4.3.19 *If f is total, there is a rule $f(x_1, \ldots, x_n) \to r \in R$ where each x_i is a variable.*

Proof. Trivial. □

For a total function symbol f there is not problem if we remove all its argument from its replacement set.

Lemma 4.3.20 *[NF01] Let μ be a replacement map satisfying the following condition:*

$$\mu(f) = \{1, \ldots, ar(f)\} \text{ if there is no rule } f(\vec{x_i}) \to r \in R$$

where each x_i is a variable and $x_i \neq x_j$ for $i \neq j$. Each μ-normal form is a normal form.

Proof. Suppose that a μ-normal form has one or more $f \in \Sigma$ such that $f(\vec{x_i}) \to r \in R$, f occurring at the outermost position is at an active position as well as a redex position. It contradicts the definition of a μ-normal form. Thus, each μ-normal form does not have such a symbol f. From the assumption, all positions of a term are active and each μ-normal form has no redex. □

Theorem 4.3.21 *Let μ be a replacement map satisfying the following condition:*

$$\mu(f) = \{1, \ldots, ar(f)\} \text{ if } f \text{ is not total.}$$

Each μ-normal form is a normal form.

Proof. From Lemma 4.3.19 and 4.3.20. □

Example 4.3.22 For the TRS $R = \{f(x, y) \to x, g(0, x) \to x\}$. The canonical replacement map μ_R is that $\mu_R(f) = \emptyset, \mu_R(g) = \{1\}, \mu_R(0) = \emptyset$. A μ_R-normal form is a root-stable form because of the left-linearity of R and Proposition 4.3.15. If we define a replacement map μ as $\mu(f) = \emptyset, \mu(g) = \{1, 2\}, \mu(0) = \emptyset$, a μ-normal form is a normal form from Lemma 4.3.20.

Corollary 4.3.23 *Let R be TRS and φ an E-strategy map such that $\varphi(f)$ has 0 if there is a rule $f(x_1, \ldots, x_n) \to r \in R$, and $\varphi(f)$ has all $i \in \{1, \ldots, ar(f)\}$ for $f \notin D(R)$. φ is correct, i.e. each evaluated term is a normal form.*

Proof. From Theorem 4.3.9 and Theorem 4.3.21. □

Corollary 4.3.24 *Let R be RPS and φ an E-strategy map such that $\varphi(f)$ has 0 for $f \in D(R)$ and $\varphi(f)$ has all $i \in \{1, \ldots, ar(f)\}$ for $f \notin D(R)$. φ is correct, i.e. each evaluated term is a normal form.*

Proof. Trivial. □

Example 4.3.25 For a RPS, in which each left-hand side is $f(x_1, \ldots, x_n)$, we define an E-strategy map φ as $\varphi(f) = [0]$ if $f \in D(R)$ and $\varphi(f) = [1, \ldots, ar(f)]$ if $f \notin D(R)$. From Theorem 4.3.21 and Theorem 4.3.9, each evaluated term is a normal form.

4.3.3 Ground totality

We showed that evaluation of variable arguments of $f \in \Sigma$ is not necessary for any term $f(\ldots)$ to be a redex. We conversely analyze which arguments of $f \in \Sigma$ are sufficient for any term $f(\ldots)$ to be a redex. In this subsection we restrict an evaluated term to be a ground term and propose another condition for the correctness of the E-strategy. Note that we assume that the signature Σ is all function symbols occurred in a given TRS R.

A term t is ground reducible with respect to a TRS R if all its ground instances contain a redex [JK86]. Ground reducibility is decidable for ordinary TRSs [KNZ97]. For example, we consider TRS R_{n_1} again.

$$R_{n_1} = \begin{cases} +(x, 0) \to x \\ +(x, s(y)) \to s(+(x, y)) \end{cases}$$

A reducible term (i.e. not a normal form) is of course ground reducible. There is a term such that it is not reducible but ground reducible. A term $+(x, y)$ is ground reducible because any ground instance is reducible, that are $+(t, 0)$, $+(t, s(t'))$ and $+(t, +(t', t''))$ (where $+(t', t'')$ is reducible by the induction hypothesis). Ground reducibility of a term $f(x_1, \ldots, x_n)$ can be regarded as totality of a function f.

Observing the above example of the ground reducible term $+(x, y)$, only the second argument of $+$ is essential to be a redex, i.e. a term $+(s, t)$ is a redex for any term s and any ground term t. In this example it may be sufficient to evaluate the second argument for the term to be a redex. Then we call $+$ a ground total function symbol on $\{2\}$.

Definition 4.3.26 A function symbol f is ground total on I if and only if $f(t_1, \ldots, t_n)$ is a redex whenever t_i is a ground normal form for each $i \in I$. A replacement map μ is ground total if and only if $\mu(f) = I$ such that (1) $I = \{1, \ldots, ar(f)\}$, or (2) f is ground total on I.

We show that for a ground total replacement map μ each μ-normal form is a normal form if it is a ground term.

Theorem 4.3.27 *Let μ be a replacement map. If μ is ground total, each ground μ-normal form is a normal form.*

Proof. We assume that $t \equiv f(t_1, \ldots, t_n)$ is a ground μ-normal form. I is a set of arguments satisfying the condition of the definition of the ground total replacement map in Definition 4.3.26. For $i \in I$, t_i is a μ-normal form. From the induction hypothesis the t_i is a normal form. In the case of (1), since all t_i are normal form and t is not redex from Lemma 4.3.11, t is a normal form. In the case of (2), t is a redex from the definition of ground totality. It is inconsistent. Hence, we can say that for a ground total replacement map a ground μ-normal form has not any function symbol which is ground total on some I. $\square$

We redefine the (μ-)correctness for ground terms and propose a condition on which the E-strategy is μ-correct for ground terms.

Definition 4.3.28 An E-strategy map φ is ground (μ-)correct if each ground evaluated term is a (μ-)normal form.

Theorem 4.3.29 *Let μ be a replacement map and φ an E-strategy map. If μ is ground total and $\varphi \in \Phi(\mu)$, φ is ground correct.*

Proof. From Theorem 4.3.9 and 4.3.27. □

Corollary 4.3.30 *Let μ be a replacement map, φ an E-strategy map and s a ground term. If μ is ground total, $\varphi \in \Phi(\mu)$ and $t = red(s)$, t is a μ-normal form.*

Proof. A term reduced from a ground term is also a ground term. □

Example 4.3.31 Consider the following TRS R_{b_1}:

$$R_{b_1} = \begin{cases} \wedge(1, x) \to x \\ \wedge(0, x) \to 0 \\ \wedge(x, 1) \to x \\ \wedge(x, 0) \to 0 \end{cases}$$

Since there is no variable argument for a function symbol $\wedge$, we cannot evaluate any argument lazily with keeping μ-correctness. On the other hand, under the assumption that an input term is a ground term, we can stop evaluating either first or second argument of $\wedge$ with keeping μ-correctness. Since $\wedge$ is ground total on both $\{1\}$ and $\{2\}$, both E-strategy maps $\varphi(\wedge) = [1, 0]$ and $\varphi'(\wedge) = [2, 0]$ are ground μ-correct.

For a TRS, a function symbol f and a set I of arguments it is decidable whether f is ground total on I. For a left-linear TRS, it is easy to see that whether a term is a redex or not is decidable because it is sufficient to check terms at most deep as the same depth of the largest left-hand side of a given TRS. Even if a TRS is not left-linear, we can see that in finite times.

Example 4.3.32 Consider R_{n_1} again.

$$R_{n_1} = \begin{cases} +(x, 0) \to x \\ +(x, s(y)) \to s(+(x, y)) \end{cases}$$

To prove that $+$ is ground total on $I = \{2\}$ we must show that $+(x, t)$ is a redex for any ground normal form t. It is enough to consider only terms which are smaller than or equal to $+(x, s(y))$, that are $+(x, 0)$ and $+(x, s(0))$. Since they are redeces, $+$ is ground total on I. This does not hold for a non-linear TRS.

$$R = \begin{cases} f(x, 0, y) \to 0 \\ f(x, y, 0) \to 0 \\ f(x, y, y) \to s(0) \end{cases}$$

We now check whether f is ground total on $I = \{2, 3\}$ or not, like the above. So we will show that $f(x, t, t')$ is a redex for any ground normal form t. If we only consider terms which are smaller than or equal to $f(x, y, y)$, t and t' should become 0 and $s(0)$. So we conclude that f is ground total on I since all $f(x, 0, 0)$, $f(x, s(0), 0)$, $f(x, 0, s(0))$ and

$f(x, s(0), s(0))$ are redeces. Of course it is not correct, for example $f(x, s(0), s(s(0)))$ is not a redex.

In such a case, we add terms which have variables to ground terms to be considered, for example $s(x_1), s(x_2), s(x_3), \ldots$ Then there is a term which is not a redex, such as $f(x, s(x_1), s(x_2))$. Is it a guarantee to exist a "ground" term which is not a redex? The answer is yes because we can substitute ground terms t_i for x_i where each t_i differs from each other. There are such ground terms, for example $t_1 = 0$, $t_2 = s^k(0)$, $t_3 = s^{2k}(0) \ldots$ where k is a maximal height of left-hand sides of rules.

4.3.4 Order-sorted terms

For a TRS with the order-sort, we can obtain weaker condition on which each μ-normal form is a normal form.

Definitions

An order-sorted signature is formalized as a triple $(S, \leq, \Sigma)$ where S is a set of sorts, Σ is an $S^* \times S$-sorted family $\{\Sigma_{w,s} \mid w \in S^* \text{ and } s \in S\}$ and $\leq \in S \times S$ is a partial order [DF98, GM92].

We write $f : w \to s$ for $f \in \Sigma_{w,s}$ and call w its arity, s its value (or result or coarity or sort). Especially $c : s$ is written instead of $c : \varepsilon \to s$. A variable V is an S-sorted family $\{V_s \mid s \in S\}$ of countably infinite sets.

A set of order-sorted terms $T(\Sigma, V)$ (or T) is constructed as the least family $\{T(\Sigma, V)_s \mid s \in S\}$ (or T_s) of sets of S-sorted terms satisfying the following conditions: $V_s \subset T_s$ and $f(t_1, \ldots, t_n) \in T_s$ for $t_i \in T_{s'_i}$, $s'_i \leq s_i$, $s' \leq s$ and $f : s_1 \ldots s_n \to s'$. We write $t : s$ for $t \in T_s$ and call s a sort of t. A rewrite rule is a pair of terms $l : s \to r : s'$ where $s \leq s'$ or $s' \leq s$. Other subjects are defined straightforwardly, term rewriting, context-sensitive rewriting and so on. More details are found in the literatures [DF98, GM92].

Hierarchical order

For order-sorted TRS, we propose a method for obtaining suitable context-sensitive rewriting in which some waste search for a redex can be eliminated.

Definition 4.3.33 Let $(S, \leq, \Sigma)$ be an order-sorted signature. We define a hierarchical order $\trianglelefteq_\Sigma$ as the least quasi order on S satisfying following condition:

$$\begin{array}{l} s_i \trianglelefteq_\Sigma s \text{ if } \exists f : s_1 \ldots s_n \to s, \\ s \trianglelefteq_\Sigma s' \text{ if } s \leq s' \text{ or } s' \leq s. \end{array}$$

We write $s \triangleleft_\Sigma s'$ if $s \trianglelefteq_\Sigma s'$ but $s' \not\trianglelefteq_\Sigma s$.

Example 4.3.34 Let $(S, \leq, \Sigma)$ be an order-sorted signature such that $S = \{s_1, s_2, s_3\}$, $\Sigma = \{f : s_1\, s_2 \to s_3, g : s_1\, s_1 \to s_2, 0 : s_1\}$ and $s_1 \leq s_2$. Then the hierarchical order $\trianglelefteq_\Sigma$ is that $s_1 \trianglelefteq_\Sigma s_3$, $s_2 \trianglelefteq_\Sigma s_3$ from $f : s_1, s_2 \to s_3$, $s_1 \trianglelefteq_\Sigma s_2$ from $g : s_1\, s_1 \to s_2$ and $s_2 \trianglelefteq_\Sigma s_1$ from $s_1 \leq s_2$. Hence we get $s_1 \sim s_2 \triangleleft_\Sigma s_3$.

The hierarchical order $\trianglelefteq_\Sigma$ preserves subterm relations of order-sorted terms (Fig. 4.5).

Lemma 4.3.35 *$s \trianglelefteq_\Sigma s'$ if $t : s$ is a subterm of a term $t' : s'$.*

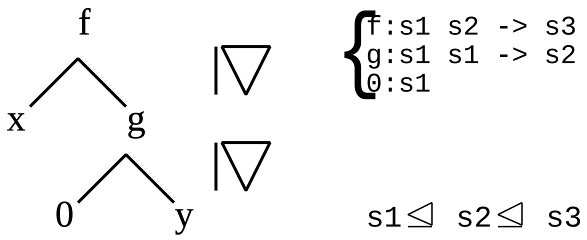

Figure 4.5: $\trianglelefteq_\Sigma$ preserves subterm relations

Proof. We prove the claim by the induction on the structure of the term t'. It is trivial if $t \equiv t'$. If t is a strict subterm of t', we can write $t' = f(t_1, \ldots, t_n)$ where $f : s_1 \ldots s_n \to s'$ and t is a subterm of t_i for some i. From the induction hypothesis, $s \trianglelefteq_\Sigma s_i$ for some i. From the definition of $\trianglelefteq_\Sigma$, $s_i \trianglelefteq_\Sigma s'$ for any i. Hence $s \trianglelefteq_\Sigma s'$. □

Definition 4.3.36 Let $(S, \leq, \Sigma)$ be an order-sorted signature and R a TRS. We define the set of least sorts S_R of the TRS R as the least sorts among the all rules of R with respect to the hierarchical order, i.e.

$$S_R = \{s \in S \mid l \to r : s \in R \text{ and there is no } l \to r : s' \in R \text{ such that } s' \triangleleft_\Sigma s\}$$

where we write $l \to r : s \in R$ if $l : s$, $r : s'$ (or $l : s'$, $r : s$) and $s \geq s'$.

Example 4.3.37 Let $(S, \leq, \Sigma)$ be the same as the above example and $R = \{f(x, y) \to c, g(x) \to c\}$. Then $S_R = \{C, D\}$.

For an order-sorted term, it is unnecessary to search a redex in a term having a sort which is less than a sort of the least sort set S_R

Lemma 4.3.38 *Let R be a TRS. If a term $t : s$ has a redex, there is a sort $s' \in S_R$ such that $s' \trianglelefteq_\Sigma s$.*

Proof. First we prove the claim for the case that $t : s$ is a redex itself. From the assumption there is a rule $l \to r : s \in R$. If $s \in S_R$, $s' = s$ exists. If $s \notin S_R$, there is a $l \to r : s' \in R$ such that $s' \triangleleft_\Sigma s$. Of course $\triangleright_\Sigma$ terminates. There exists a least sort s' such that $l \to r : s' \in R$ and $s' \triangleleft_\Sigma s$. From Definition 4.3.36 the sort s' is a member of S_R.

Now we assume a term $t : s$ has a redex $u : s''$, which means that u is a subterm of t. From the above proof there is a sort $s' \in S_R$ such that $s' \trianglelefteq_\Sigma s''$. From Lemma 4.3.35 $s'' \trianglelefteq_\Sigma s$ and finally we obtain $s' \trianglelefteq_\Sigma s$. □

From the above properties of the hierarchical order and the least sort set, we obtain a suitable replacement map for a TRS with order-sorted signatures.

Theorem 4.3.39 *Let $(S, \leq, \Sigma)$ be an order-sorted signature, R a TRS and μ a replacement map satisfying the following condition:*

$$i \in \mu(f) \text{ if } f : s_1 \ldots s_n \to s, \text{there is an } s' \in S_R \text{ such that } s' \trianglelefteq_\Sigma s_i.$$

The context-sensitive rewrite relation $\to_\mu$ is identical with the rewrite relation $\to_R$.

Proof. It is sufficient that $p \in O_\mu(t)$ if $t|_p$ is a redex for any term t. If $p \notin O_\mu(t)$, there is a $f \in \Sigma$ such that $t|_{p'} = f(t_1, \ldots, t_n)$, $i \notin \mu(f)$ and $t|_p$ is a subterm of t_i. From the condition there is no $s' \in S_R$ such that $s' \trianglelefteq_\Sigma s_i$. From Lemma 4.3.38 the term $t_i : s_i$ has no redex. □

Consider the following example which is the result of adding the some equations to the above specification. $S = \{Zero, NzNat, Nat, Bool\}$, $NzNat < Nat$, $\Sigma = \{0 : Zero, s : Nat \to NzNat, t : Bool, f : Bool, eq : Nat\,Nat \to Bool\}$ and

$$R_e = \begin{cases} eq(0,0) \to t \\ eq(0, sN) \to f \\ eq(s(x), 0) \to f \\ eq(s(x), s(y)) \to eq(x, y) \end{cases}$$

A function eq replies whether arguments are equal or not. Since $Nat \triangleleft_\Sigma Bool$, we let μ a replacement map such that $\mu(0) = \mu(s) = \cdots = \mu(eq) = \emptyset$. The context-sensitive rewrite relation $\to_\mu$ is identical with the rewrite relation $\to_R$ from Theorem 4.3.39.

Corollary 4.3.40 *Let φ be an E-strategy map. Let $(S, \leq, \Sigma)$ be an order-sorted signature, R a TRS and μ a replacement map satisfying the condition of Theorem 4.3.39. If $\varphi \in \Phi(\mu)$, φ is correct.*

Proof. From Theorems 4.3.39 and 4.3.9. □

4.4 Strictness

In the above section, we discussed which arguments could be evaluated later with keeping (μ-)correctness. On the other hand, it is not always that such lazy evaluation is most efficient. Consider the following TRS for example: $R = \{f(x) \to x\}$. Since the argument of the symbol f is a variable argument, we can move it after 0 in the local strategy of f: $\varphi(f) = [0, 1]$. However, even if the whole term is rewritten before the argument term, the argument must be rewritten because $f(t) \to_R t$. In such a case it is not meaningful to move such arguments after 0. We call such an argument a strict argument.

Example 4.4.1 Consider TRS R_{n_2} and φ such that $\varphi(+) = [1, 0, 2]$, $\varphi(\times) = [1, 0, 2]$.

$$R_{n_2} = \begin{cases} +(0, x) \to x \\ +(s(x), y) \to s(+(x, y)) \\ \times(0, x) \to 0 \\ \times(s(x), y) \to +(\times(x, y), y) \end{cases}$$

For any term $t \equiv +(t_1, t_2)$, if evaluating t_2 does not terminate, trivially the whole term t does not terminate too. For example,

$$\begin{aligned} red(+(s(s(\cdots(s(0)))), t_2)) &= s(red(+(s(\cdots(s(0))), t_2)))) \\ &\cdots \\ &= s(s(\cdots(red(+(0, t_2))))) \\ &= s(s(\cdots(red(t_2)))) \end{aligned}$$

Thus, the function symbol $+$ is strict in the second argument. On the other side, the function symbol $\times$ is not strict in the second argument because $red(\times(0, t)) = 0$ for any term t even if $red(t)$ does not terminate.

4.4.1 Strictness function

In lazy functional languages, a function is strict in a certain argument if the eager evaluation of that argument does not change the termination behavior of the program. This can be defined as follows: a function f of arity n is strict in its i-th argument if and only if

$$f x_1 \dots x_{i-1} \bot x_{i+1} \dots x_n = \bot$$

for all possible values $x_j (j \neq i)$ where $\bot$ stands for any non-terminating expression. Strictness analysis is a compile-time analysis of a program that is used to tell whether or not a function is strict in its argument [PE93]. If it restates, for a strategy F, a function f of arity n is strict in its i-th argument if and only if $F(f(t_1, \dots, t_n))$ does not terminates for each terms t_k where $F(t_i)$ does not terminates.

We define a function $S_\mu : \Sigma \to \mathcal{P}(\mathcal{N})$ to analyze strict arguments of a function symbol. An object of this analysis is a variable argument because it is regarded as evaluating lazily.

Definition 4.4.2 For a replacement map μ, functions $S_i : \Sigma \to N$ $(i \in N)$ is defined as follows: for $f \notin D(R)$, $S_i(f) = \mu(f)$ for all $i \in N$ and for $f \in D(R)$,

$$\begin{aligned} S_0(f) &= \mu(f) \cap V_R(f) \\ S_{i+1}(f) &= \left\{ i \in S_i(f) \middle| \forall f(\vec{l_j}) \to r \in R, \exists q \in O_{S_i}(r).r|_q \equiv l_i \in V. \right\}, \end{aligned}$$

where $O_{S_i}(t)$ is similarly defined as $O_\mu(t)$ (S_i can be regarded as a replacement map). We define $S_\mu = S_i$ for a first S_i such that $S_i = S_{i+1}$.

Since $S_i \sqsupset S_{i+1}$, we can compute S_μ in finite times. For a function symbol f an argument in the set $S_\mu(f)$ is called a strict argument of f or the function is strict in the argument.

Example 4.4.3 Consider TRS R_{n_2} and $\mu = \mu_\top$.

$$R_{n_2} = \begin{cases} +(0, x) \to x \\ +(s(x), y) \to s(+(x, y)) \\ \times(0, x) \to 0 \\ \times(s(x), y) \to +(\times(x, y), y) \end{cases}$$

We can get the map S_μ as follows.

$$\begin{array}{lclcl} S_0(+) = \{2\} & & S_1(+) = \{2\} & & S_2(+) = \{2\} \\ S_0(\times) = \{2\} & \Rightarrow & S_1(\times) = \emptyset & \Rightarrow & S_2(\times) = \emptyset \\ S_0(s) = \{1\} & & S_1(s) = \{1\} & & S_2(s) = \{1\} \end{array}$$

$S_1 = S_2$,

$$\begin{array}{l} S_\mu(+) = \{2\} \\ S_\mu(\times) = \emptyset \\ S_\mu(s) = \{1\} \end{array}$$

In the above example, the second argument of $+$ is a strict argument because $2 \in S_\mu(+)$. In our definition a strict argument means that it does not disappear by any reduction unless the argument itself is reduced. So $\times$ is not strict in 2 because it disappears by applying rule $\times(0, x) \to 0$.

4.4.2 Preserving termination

In the previous section, we showed which arguments may be evaluated later with keeping μ-correctness. However, in the view of efficiency, it is not always the best choice that all such arguments are evaluated later. If it is known that a variable argument is strict, it is better to evaluate the argument eagerly. We show that the eager evaluation of a strict argument does not change the termination behavior.

Of course the eager evaluation of some argument has no effect on the correctness.

Lemma 4.4.4 *Let μ be a replacement map and φ and φ' E-strategy maps such that $\varphi'(f) = [k, i_1, \dots, i_n]$ for some function symbol $f \in \Sigma$ and $k \in \mu(f)$ where $\varphi(f) = [i_1, \dots, i_n]$, and $\varphi'(g) = \varphi(g)$ for any other function symbol $g \in \Sigma$. Then, if φ is μ-correct, φ' is also μ-correct.*

Proof. Trivial. $\Box$

We show some lemmata for proving that the eager evaluation of a strict argument preserves termination.

Lemma 4.4.5 *Let R be a TRS, t a term and $t' \equiv t|_p$ a strict subterm under some strict argument, i.e. $p \in O_{S_\mu}(t)$ and $p \neq \varepsilon$. If a term t is rewritten into a term s at the root position, i.e. $t \to_\varepsilon s$, then the term t' is also a subterm of s under some strict argument, i.e. $t' \equiv s|_{p'}$ for some $p' \in O_{S_\mu}(s)$.*

Proof. We assume that $t \equiv l\theta \to_\varepsilon r\theta \equiv s$, $t \equiv f(t_1, \dots, t_n)$, $l \equiv f(l_1, \dots, l_n)$ and $p = i.p''$. It is trivial from the definition of the function S_μ that a strict argument is a variable argument for a defined symbol, i.e. $S_\mu(f) \subset V_R(f)$ for $f \in D(R)$. So l_i is a variable because i is a variable argument of f and the term t' is under the substitution, i.e. $t' \equiv \theta(l_i)|_{p''}$. From the definition of S_μ, a variable l_i also occurs in the right-hand side r at the position q. It is trivial that active positions are transitive. Hence we conclude that $t' \equiv s|_{p'}$ for $p' = q.p'' \in O_{S_\mu}(s)$. $\Box$

Lemma 4.4.6 *Let R be a TRS and $\perp$ a fresh variable. If a function symbol f is strict in an argument i, i.e. $i \in S_\mu(f)$, a variable $\perp$ does not disappear by any reduction from a term $t \equiv f(t_1, \dots, t_{i-1}, \perp, t_{i-1}, \dots, t_n)$ for any terms $t_j (j \neq i)$, i.e. $t \to_\mu^* s \Rightarrow \exists p \in O_\mu(s).(s)_p = \perp$.*

Proof. We prove the claim by the induction of the number of the reduction steps $t \to_\mu^* s$. It is trivial for $t \equiv s$ because $\perp = (t)_i$ and $i \in O_{S_\mu}(t) \subset O_\mu(t)$. We assume that $t \to_\mu^* s \to_\mu s'$ and $(s)_p = \perp$ for some $p \in O_\mu(s)$. If $s \to_{p'} s'$ for a disjoint position p' of p, $(s')_p = \perp$ for same $p \in O_\mu(s)$. If $s \to_{p'} s'$ for a position p' above p, i.e. $p = p'.p''$ for some p'', the variable $\perp$ occurs in s' from Lemma 4.4.5. $\Box$

Theorem 4.4.7 *Let μ be a replacement map and φ and φ' E-strategy maps such that $\varphi'(f) = [k, i_1, \dots, i_n]$ for some function symbol $f \in \Sigma$ and $k \in S_\mu(f)$ where $\varphi(f) = [i_1, \dots, i_n]$, and $\varphi'(g) = \varphi(g)$ for any $g \neq f$. Then, if $red_\varphi(t)$ terminates, $red_{\varphi'}(t)$ also terminates for each term t (where red_φ is a function red defined by φ).*

Proof sketch. Let t be a term $f(t_1, \ldots, t_n)$ and $k \in S_\mu(f)$. We prove that if $red(t)$ terminates, $red(t_k)$ also terminates. From Lemmata 4.4.5 and 4.4.6 the term t_k does not disappear until reducing itself. From Lemma 4.4.4 the term must be reduced because it is in an active position. So if $red(t_k)$ does not terminate, $red(t)$ also does not terminate. Because of this, we can evaluate a strict argument eagerly with no effect on termination (Fig. 4.6). □

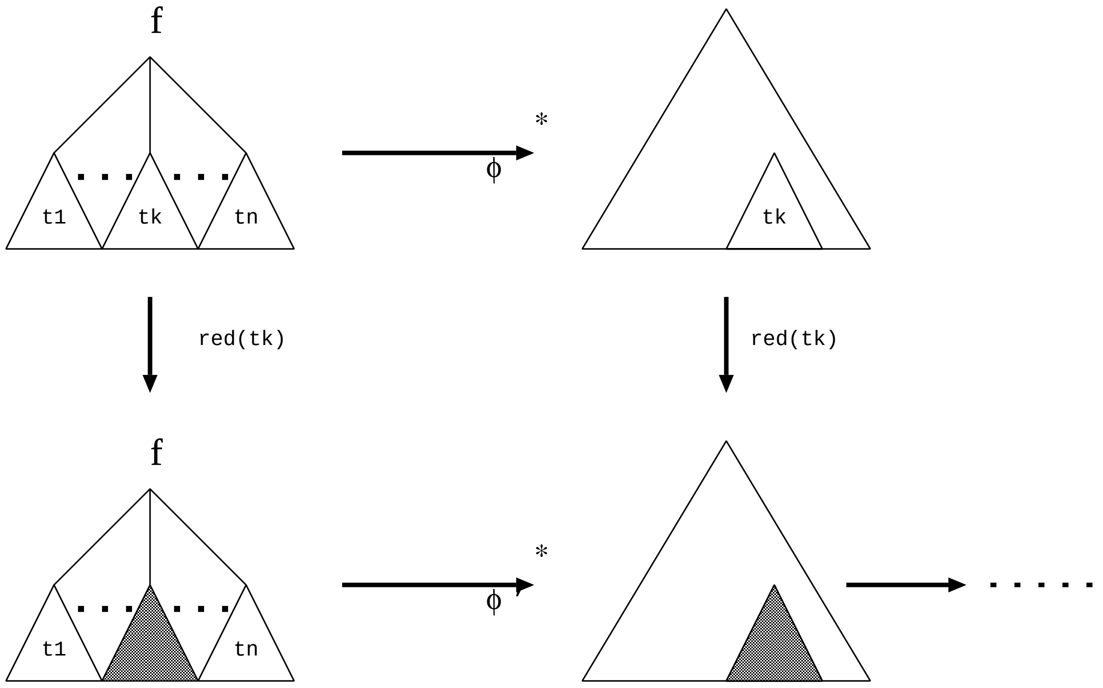

Figure 4.6: Preserving termination

Example 4.4.8 Consider TRS R_{n_3}.

$$R_{n_3} = R_{n_2} \cup \{double(x) \to x + x\}$$

Although the argument of *double* is trivially in $V_{R_{n_3}}(double)$, evaluation under $\varphi(double) = [0, 1]$ spends more rewrite steps than that under $[1, 0]$.

$$\begin{aligned} \varphi : &\quad double(+(0,0)) \to +(+(0,0), +(0,0)) \to +(0, +(0,0)) \to +(0,0) \to 0 \\ \varphi' : &\quad double(+(0,0)) \to double(0) \to +(0,0) \to 0 \end{aligned}$$

One of popular solution of this problem is sharing variables. But we can give another solution by a strictness analysis. The argument of *double* is a strict argument, i.e. $1 \in S_\mu(double)$. So we can evaluate it eagerly with no effect on termination.

Chapter 5

The on-demand evaluation strategy

In this chapter we show one of the main topics in this thesis: the on-demand evaluation strategy. In the above chapter, we gave a method of defining local strategies to obtain root-stable forms, Corollary 4.3.5. However there are some cases such that we cannot define suitable local strategies. Consider the TRS R_{l_2}.

$$R_{l_2} = \begin{cases} \mathit{inf} \to \mathit{cons}(0, \mathit{inf}) \\ 2nd(\mathit{cons}(x, \mathit{cons}(y, z))) \to y \end{cases}$$

$2nd$ returns the second element of an input list. $2nd(\mathit{inf})$ has a normal form 0 since there is the reduction sequence

$$2nd(\mathit{inf}) \to_R 2nd(\mathit{cons}(0, \mathit{inf})) \to_R 2nd(\mathit{cons}(0, \mathit{cons}(0, \mathit{inf}))) \to_R 0.$$

Unfortunately there is no local strategy to obtain this reduction sequence. If $\varphi(\mathit{cons})$ has 2, $\mathit{red}(\mathit{inf})$ does not terminates. If $\varphi(\mathit{cons})$ does not have 2, $\mathit{red}(2nd(\mathit{inf}))$ terminates halfway as follows: $2nd(\mathit{inf}) \to_R 2nd(\mathit{cons}(0, \mathit{inf}))$. One of the reasons why it does not work well is that we cannot decide whether the second argument of *cons* should be reduced or not. The need to reduce an argument depends on the shape of rules. For a term whose root symbol is a defined symbol, it can be said that its goal is to be a redex of the corresponding rules. Consider the rule $2nd(\mathit{cons}(x, \mathit{cons}(y, z))) \to y \in R$. For $2nd(\mathit{cons}(t, t'))$ we need to reduce t', but $2nd(\mathit{cons}(t, \mathit{cons}(t', t'')))$ we need not to reduce any subterm because it is already a redex. In the ordinary E-strategy there are only two kinds of instructions for an argument: "reduce" or "do not reduce". For solving this problem we need to define a new indication, which stands for "reduce on demand".

In this section we define a function $\mathit{match} : T \to T$ as a function to do on-demand matching. *match* is a strategy which reduces subterms with the necessity to be reduced. For example, consider the TRS $R = \{f(s(x), p(y)) \to 0\}$ and the term $f(t, p(t'))$ where t and t' are redeces. *match* reduces a term t.

5.1 Definition

We formalize the on-demand E-strategy by the pair of maps that are the E-strategy map φ defined in the above section and the on-demand map o defined as follows. If the E-strategy map is regarded as defining the order of reduction, the on-demand map can be regarded as defining the order of matching. An E-strategy map is the same as the ordinary

E-strategy which stands for the order in which a term is reduced. An on-demand map is defined newly in this section. We can control the order in which a term is matched to left-hand sides of a TRS by an on-demand map. For the on-demand E-strategy the order of matching is very important because a target term may be changed while it is being matched to left-hand sides of a TRS in reduction with the on-demand E-strategy. We show some example which can be treated well by our on-demand E-strategy. We also show a useful property of the on-demand E-strategy about the shape of evaluated terms.

Definition 5.1.1 A map $o : \Sigma \to \mathit{List}(\mathcal{N})$ is an on-demand map if and only if $1 \leq i \leq ar(f)$ for any $i \in o(f)$. We call $o(f)$ an on-demand list. Note that an on-demand list has not the element 0. For an E-strategy map φ and an on-demand map o we call (φ, o) on-demand E-strategy maps.

Definition 5.1.2 Let o be an on-demand map. The priority function $pr_o : T \to \mathit{List}(N^*)$ is defined as follows:

$$pr_o(t) = \begin{cases} \varepsilon & \text{if } \ t \in V \\ \varepsilon :: (add(i_1, l_1)@\cdots@add(i_n, l_n)) & \text{if } \begin{array}{c} t \equiv f(\vec{t_i}), o(f) = [\vec{i_j}] \\ \text{and } pr_o(t_{i_j}) = l_j \end{array} \end{cases}$$

where the function $add : N^* \times \mathit{List}(N^*) \to \mathit{List}(N^*)$ is defined as follows: $add(p, nil) = nil$ and $add(p, p' : ps) = p.p' : add(p, ps)$, and @ is the concatenation of lists. We also define the function $cut : N^* \times \mathit{List}(N^*) \to \mathit{List}(N^*)$ where $cut(p, l)$ returns the list obtained by cutting off all the consecutive head elements under position p from the list l.

$$cut(p, l) = \begin{cases} cut(p, tl(l)) & \text{if } hd(l) = p.p' \text{ for some } p' \\ l & \text{o.w.} \end{cases}$$

The priority function takes a term and returns the priority list for traversing an input term for matching. In the on-demand matching which will be defined as the function *match* after, we compare a term t to the left-hand sides of a given TRS in the order of $pr_o(t)$.

Example 5.1.3 Let $o(f) = [2, 3], o(g) = [2, 1], o(0) = []$. The priority list of the term $f(x, g(0, y), z)$ is obtained as follows (Fig. 5.1):

$$\begin{array}{rcl} pr_o(g(0, y)) & = & \varepsilon :: (add(2, pr_o(y))@add(1, pr_0(y))) \\ & = & \varepsilon :: (add(2, [\varepsilon])@add(1, [\varepsilon])) \\ & = & \varepsilon :: ([2]@[1]) \\ & = & [\varepsilon, 2, 1] \end{array}$$

$$\begin{array}{rcl} pr_o(f(x, g(0, y), z)) & = & \varepsilon :: (add(2, pr_o(g(0, y)))@add(3, pr_0(z))) \\ & = & \varepsilon :: (add(2, [\varepsilon, 2, 1])@add(3, [\varepsilon])) \\ & = & \varepsilon :: ([2, 2.2, 2.1]@[3]) \\ & = & [\varepsilon, 2, 2.2, 2.1, 3] \end{array}$$

If a symbol of a term which is different from all the corresponding ones of the left-hand sides is found at a position while the term is being compared to the left-hand sides, the on-demand matching reduces the subterm at the position. The cut function is used

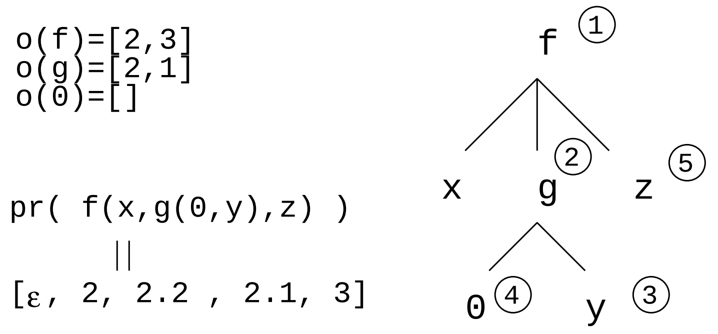

Figure 5.1: The priority function

when such a subterm is reduced. If such a subterm is reduced, we must redefine the priority list of the term depending on the result of reducing the subterm. Suppose that we are comparing the term $f(x, g(0, y), z)$ to the left hand sides of a TRS according to the priority list shown in Fig. 5.1 and find g at position 2 different from all the symbols of the left-hand sides at the same position and the subterm $g(0, y)$ at position 2 is reduced into a term $f(0, 0, 0)$. In order to continue the on-demand matching, we have to calculate a new priority list for $f(x, f(0, 0, 0), z)$. First we eliminate the positions of $g(0, y)$ from the list by the cut function (Fig. 5.2).

$$cut(2, [2, 2.2, 2.1, 3]) = [3]$$

Next we add the priority list $[2, 2.2, 2.3]$ of $f(0, 0, 0)$ obtained by reducing $g(0, y)$ to the whole priority list $[3]$, and continue the on-demand matching according to $[2, 2.2, 2.3, 3]$ (Fig. 5.3). This operation will be occurred the following the definition of the on-demand matching.

$$\begin{aligned} add(2, pr_o(f(0, 0, 0)))@[3] &= add(2, [\varepsilon, 2, 3])@[3] \\ &= add([2, 2.3, 2.3])@[3] \ ll \\ &= [2, 2.2, 2.3, 3] \end{aligned}$$

We define the on-demand E-strategy by adding the function $match : T \to T$ to Definition 2.3.2 of the ordinal E-strategy.

Definition 5.1.4 Let (φ, o) be on-demand E-strategy maps and R a TRS. The functions $red : T \to T$, $eval : T \times List(\mathcal{N}) \to T$ and $match : T \to T$ are defined simultaneously as follows:

$$red(t) \quad = \quad eval(t, \varphi((t)_\varepsilon))$$

$$eval(t, l) \quad = \quad \begin{cases} t & \text{if} \quad l = nil \\ eval(t[red(t_i)]_i, l') & \text{if} \quad l = i : l', i > 0 \\ red(s) & \text{if} \quad l = 0 : l', t' = match(t), t' \to_\varepsilon s \\ eval(t, l') & \text{if} \quad l = 0 : l', t' = match(t), t' \notin Red(R) \end{cases}$$

```
cut(2, [2, 2.2 , 2.1, 3])=[3]
```

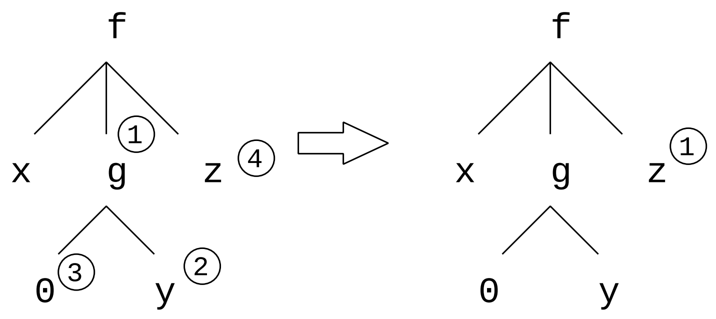

Figure 5.2: The cut function

```
add(2, pr(f(0,0,0))) @ cut(2, [2, 2.2, 2.1, 3])
                          ||
                 [2, 2.2, 2.3, 3]
```

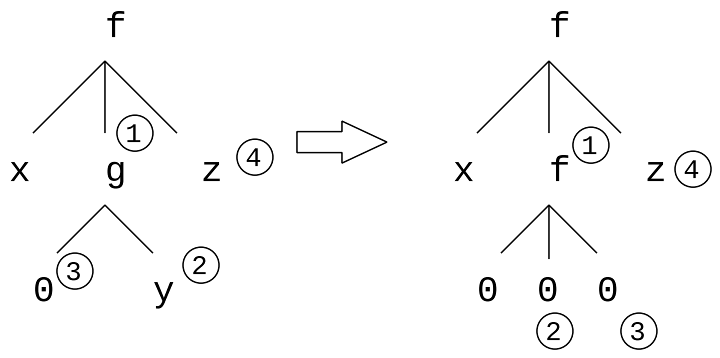

Figure 5.3: Re-calculating the priority list

$$
\begin{array}{lcl}
match(t) & = & match2(t, pr_o(t), L(R)) \\[1em]
match2(t, l, T) & = & \begin{cases} t & \text{if} \quad l = nil \\ match2(t, tl(l), T') & \text{if} \quad filter(T, t, hd(l)) = T' \neq \emptyset \\ & \qquad filter(T, t, hd(l)) = \emptyset, \\ match3(t[t']_p, l', T) & \text{if} \quad t' = red(t|_p) \text{ and} \\ & \qquad l' = add(p, pr_o(t'))@cut(p, l) \end{cases} \\[1em]
match3(t, l, T) & = & \begin{cases} match2(t, tl(l), T') & \text{if} \quad filter(T, t, hd(l)) = T' \neq \emptyset \\ t & \text{if} \quad filter(T, t, hd(l)) = \emptyset \end{cases} \\[1em]
filter(T, t, p) & = & \{s \in T \mid (t)_p = (s)_p \text{ or } (s)_p \in V \text{ or } p \notin O(s)\}
\end{array}
$$

red and *eval* are the same as that of Definition 2.3.2 except the last cases of *eval*. In the original definition $eval(t, 0 : l)$ checks whether t is a redex or not. In this definition it checks whether t' is a redex or not where t' is obtained by on-demand matching, i.e. $t' = match(t)$.

The function *match* is a top-level of the on-demand matching which hands an input term, its priority list and the set of all left-hand sides of a TRS R to the function *match*2.

$$match(t) = match2(t, pr_o(t), L(R))$$

For on-demand matching, a term is compared to all left-hand sides at the same time. $match2(t, l, T)$ compares t to all terms in T in order of l. T is a set of left-hand sides to which it is still possible to match the term t. *match*2 finds a position which is a cause of failure as a redex in order of l. When l is empty, the on-demand matching is over and *match*2 replies t immediately.

$$match2(t, nil, T) = t$$

When l is not empty, there are two cases whether the set obtained by filtering T is empty or not.

$filter(T, t, p)$ removes terms l from T such that $(l)_p$ is different from $(t)_p$.

$$filter(T, t, p) = \{s \in T \mid (t)_p = (s)_p \text{ or } (s)_p \in V\}$$

filter is a function to filter T through a check whether the symbol at p is same as that of t. Consider the following example:

$$filter(\{f(0, x), f(0, s(0)), f(0, 0), f(0, s(s(0)))\}, f(0, s(0)), 2.2).$$

This returns a T's subset whose elements have not the symbols differing from that of t at p. *filter* compares the symbol at 2.2 of $f(0, s(0))$ ($(f(0, s(0)))_{2.2} = 0$) to the symbols at 2.2 of terms $T = \{f(0, x), f(0, s(0)), f(0, 0), f(0, s(s(0)))\}$. It removes $f(0, s(s(0)))$ because the corresponding symbol is s and $f(0, 0)$ because there is no corresponding symbol and there is no variable above p. Thus, $f(0, x)$ and $f(0, s(0))$ remain. Note that it is impossible for $f(0, s(0))$ to be instances of the removed terms (Fig. 5.4).

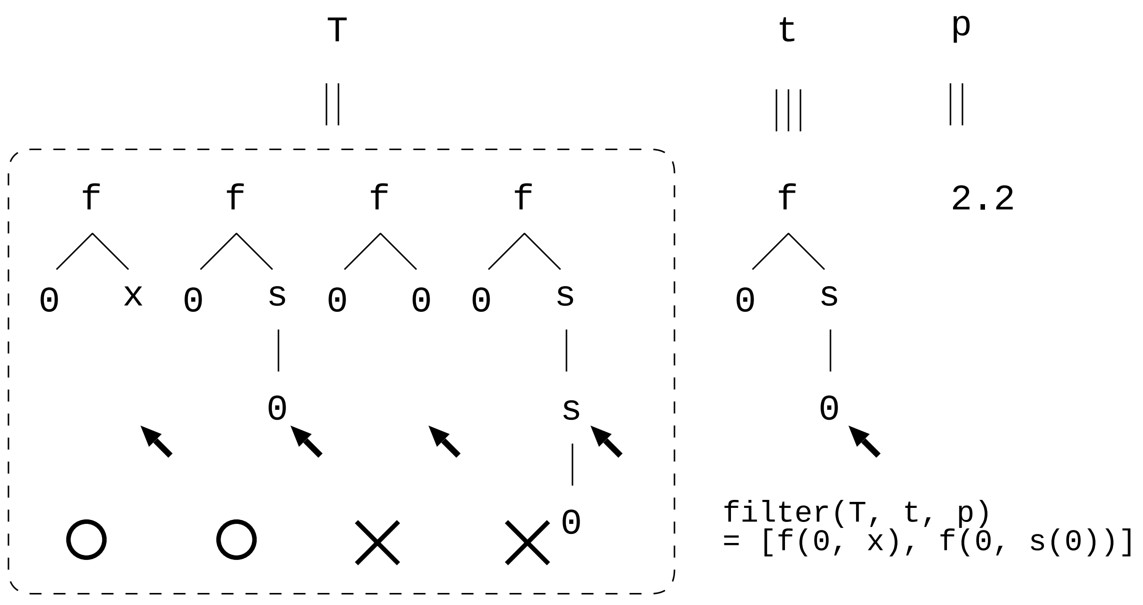

Figure 5.4: The filter function

If $T' = filter(T, t, hd(l))$ is not empty which means that there still exist left-hand sides to match the term t, the head element is removed from l and the on-demand matching are continued.

$$\begin{array}{l} match2(t, l, T) = match2(t, tl(l), T') \\ \text{ if } filter(T, t, hd(l)) = T' \neq \emptyset \end{array}$$

If T' is empty which means that there is no left-hand side to match the term t at position p, the subterm $t|_p$ is reduced for succeeding in matching. After the reduction, on-demand matching is tried again. In this case the list l is refreshed, i.e. the positions for $t|_p$ is removed from l and a new priority list of a reduced term t' is added.

$$\begin{array}{l} match2(t, l, T) = match3(t[t']_p, l', T) \\ \text{ if } \ filter(T, t, hd(l)) = \emptyset, t' = red(t|_p) \text{ and } l' = add(p, pr_o(t'))@cut(p, l) \end{array}$$

A role of the function $match3$ is checking only after on-demand reduction. If the on-demand reduction is succeed, i.e. $filter(T, t, p)$ is not empty, the on-demand matching are continued. Unfortunately it is still impossible to be a redex, i.e. $filter(T, t, p) = \emptyset$. Then it is considered that the on-demand matching is completely failed and $match3$ returns t.

$$match3(t, l, T) = \begin{cases} match2(t, tl(l), T') & \text{if} \quad filter(T, t, hd(l)) = T' \neq \emptyset \\ t & \text{if} \quad filter(T, t, hd(l)) = \emptyset. \end{cases}$$

By using the on-demand E-strategy we can treat TRS R_{l_2} of infinite lists with $2nd$ well.

Example 5.1.5 Again consider TRS R_{l_2}.

$$R_{l_2} = \begin{cases} inf \rightarrow cons(0, inf) \\ 2nd(cons(x, cons(y, z))) \rightarrow y \end{cases}$$

Let(φ, o) be on-demand E-strategy maps such that $\varphi(\mathit{inf}) = \varphi(2nd) = [0]$ and $\varphi(\mathit{cons}) = \varphi(0) = []$. and $o(\mathit{inf}) = [1], o(2nd) = [1], o(\mathit{cons}) = [2], o(0) = []$. We show an example of reduction from $2nd(\mathit{inf})$.

$$\mathit{red}(2nd(\mathit{inf})) = \mathit{eval}(2nd(\mathit{inf}), [0])$$

Since the head element of the list is 0, the on-demand E-strategy begins the on-demand matching.

$$\begin{aligned}&\mathit{match}(2nd(\mathit{inf}))\\ &= \mathit{match}2(2nd(\mathit{inf}), [\varepsilon, 1], \{\mathit{inf}, 2nd(\mathit{cons}(x, \mathit{cons}(y, z)))\})\end{aligned}$$

First, the root position is checked according to the list $[\varepsilon, 1]$. Because the root symbol of the left-hand side inf of the first rule is different from that of the term $2nd(\mathit{inf})$, it is removed from the set. On the other hand, because $2nd(\mathit{cons}(x, \mathit{cons}(y, z)))$ is identical with $2nd(\mathit{inf})$ at the root position, it remains.

$$= \mathit{match}2(2nd(\mathit{inf}), [1], \{2nd(\mathit{cons}(x, \mathit{cons}(y, z)))\})$$

Hereafter we write L instead of $\{2nd(\mathit{cons}(x, \mathit{cons}(y, z)))\}$. Next, the symbol at position 1 is checked according to the list [1]. Since the symbol of $(2nd(\mathit{inf}))$ at position 1 differs form that of $(2nd(\mathit{cons}(x, \mathit{cons}(y, z))))$ $(\mathit{inf} \neq \mathit{cons})$, inf is reduced into $\mathit{cons}(0, \mathit{inf})$. The priority list of $\mathit{cons}(0, \mathit{inf})$ is $[\varepsilon, 2]$.

$$= \mathit{match}3(2nd(\mathit{cons}(0, \mathit{inf})), [1, 1.2], L)$$

Again the symbol at position 1 is checked and it succeeds. The on-demand matching is continued.

$$= \mathit{match}2(2nd(\mathit{cons}(0, \mathit{inf})), [1.2], L)$$

The same action is done at position 1.2.

$$\begin{aligned}&= \mathit{match}2(2nd(\mathit{cons}(0, \mathit{inf})), [1.2], L)\\ &= \mathit{match}3(2nd(\mathit{cons}(0, \mathit{cons}(0, \mathit{inf}))), [1.2, 1.2.2], L)\\ &= \mathit{match}2(2nd(\mathit{cons}(0, \mathit{cons}(0, \mathit{inf}))), [1.2.2], L)\end{aligned}$$

Finally, the position 1.2.2 is checked. Since the symbol of $2nd(\mathit{cons}(x, \mathit{cons}(y, z)))$ at position 1.2.2 is a variable, matching may succeed with a substitution $\theta(z) = 0$. Therefore the list become empty and $\mathit{match}2$ returns the first argument term $2nd(\mathit{cons}(0, \mathit{cons}(0, \mathit{inf})))$.

$$\begin{aligned}&= \mathit{match}2(2nd(\mathit{cons}(0, \mathit{cons}(0, \mathit{inf}))), [], L)\\ &= 2nd(\mathit{cons}(0, \mathit{cons}(0, \mathit{inf})))\end{aligned}$$

It can be easily seen that more than one occurrences of an element in an on-demand list are meaningless, i.e. we can change a list $[i_1, \ldots, i_n]$ into $[i_1, \ldots, i_{j-1}, i_{j+1}, \ldots, i_n]$ with no effect in reduction if $i_k = i_j$ for some $k < j$. Hereafter we assume that there is no element which occurs more than once in an on-demand list.

5.2 On-demand strategies for the root-stable form

The order of elements of an on-demand list is very important. The action of $o(f) = [1, 2]$ may be different from that of $o(f) = [2, 1]$. We explain this fact by the following example.

Example 5.2.1 Consider the following TRS R_{b_2} of Boolean with conjunction.

$$R_{b_2} = \begin{cases} \wedge(x, 0) \to 0 \\ \wedge(0, 1) \to 0 \\ \wedge(1, 1) \to 1 \end{cases}$$

For $\varphi(\wedge) = [0]$ and $o(\wedge) = [1, 2]$, reduction of $\wedge(\wedge(0, 0), 1)$ does not work well.

$$\begin{aligned}&\mathit{match}(\wedge(\wedge(0,0),1))\\ &= \mathit{match2}(\wedge(\wedge(0,0),1), [\varepsilon, 1, 1.1, 1.2, 2], \{\wedge(x,0), \wedge(0,1), \wedge(1,1)\})\end{aligned}$$

The root symbol $\wedge$ of $\wedge(\wedge(0, 0), 1)$ is the same as that of each left-hand side.

$$= \mathit{match2}(\wedge(\wedge(0,0),1), [1, 1.1, 1.2, 2], \{\wedge(x,0), \wedge(0,1), \wedge(1,1)\})$$

$\wedge$ at position 1 is different from each of the left-hand sides $\wedge(0, 1)$ and $\wedge(1, 1)$, which are 0 and 1. Hence they are removed from the set. Only $\wedge(x, 0)$ remains because the symbol at position 1 is a variable.

$$= \mathit{match2}(\wedge(\wedge(0,0),1), [1.1, 1.2, 2], \{\wedge(x,0)\})$$

Since the positions 1.1 and 1.2 are under a variable, they are removed from the list.

$$= \mathit{match2}(\wedge(\wedge(0,0),1), [2], \{\wedge(x,0)\})$$

The symbol 1 at position 2 is different from the symbol 0 of the left-hand side. So 1 is reduced. However 1 is not changed, i.e. $\mathit{red}(1) = 1$, the on-demand matching is failed.

$$\begin{aligned}&= \mathit{match3}(\wedge(\wedge(0,0),1), [2], \{\wedge(x,0)\})\\ &= \wedge(\wedge(0,0),1).\end{aligned}$$

A cause of this failure is that the left-hand sides $\wedge(0, 1)$ and $\wedge(1, 1)$ are removed in the step of checking at position 1. These may be matched to the target term under another strategy. Indeed, if there is no rule $\wedge(x, 1) \to 1$ in this TRS, the subterm $\wedge(0, 0)$ at position 1 should be reduced because there is no left-hand sides whose symbol at position 1 is $\wedge$ or a variable. However, there is a rule $\wedge(x, 1) \to 1$ and the left-hand side $\wedge(x, 1)$ remains in the set because it has possibility of succeeding in matching at the time when position 1 is checked but the position 2 is not yet.

On the other hand, reduction works well for $o(\wedge) = [2, 1]$.

$$\begin{aligned}&\mathit{match}(\wedge(\wedge(0,0),1))\\ &= \mathit{match2}(\wedge(\wedge(0,0),1), [\varepsilon, 2, 1, 1.1, 1.2], \{\wedge(x,0), \wedge(0,1), \wedge(1,1)\})\\ &= \mathit{match2}(\wedge(\wedge(0,0),1), [2, 1, 1.1, 1.2], \{\wedge(x,0), \wedge(0,1), \wedge(1,1)\})\end{aligned}$$

The symbol of $\wedge(\wedge(0, 0), 1)$ at position 2 is 1. Hence the left-hand side $\wedge(x, 0)$ is removed from the set because the symbol at this position is $0 \neq 1$.

$$= \mathit{match2}(\wedge(\wedge(0,0),1), [1, 1.1, 1.2], \{\wedge(0,1), \wedge(1,1)\})$$

Here, there is no left-hand side which may be matched at position 1 because $\wedge(x, 1)$ is already removed. The subterm $\wedge(0, 0)$ is reduced into the term 0 and the target term becomes a redex.

$$
\begin{aligned}
&= match3(\wedge(0,1), [1, 1.1, 1.2], \{\wedge(0,1), \wedge(1,1)\}) \\
&= match2(\wedge(0,1), [1], \{\wedge(0,1), \wedge(1,1)\}) \\
&= match2(\wedge(0,1), [], \{\wedge(0,1)\}) \\
&= \wedge(0,1).
\end{aligned}
$$

A term is left-normal if no variable occurs before a function symbol. For example, $f(g(0,x),y)$ is left-normal but $f(s(x),s(y))$ is not left-normal because x occurs before s. A TRS R is left-normal if each left-hand side is left-normal. It seems that the problem in the above example may not be caused for a left-normal TRS and an on-demand map such that $o(f) = [1, 2, \ldots, ar(f)]$. We generalize the notion of left-normal terms, called o-normal term, in order to obtain a suitable on-demand map o for a given TRS even if it is not left-normal.

Definition 5.2.2 Let o be an on-demand E-strategy map such that $o(f) = [i_1, \ldots, i_n]$ has all arguments of $f \in \Sigma$, i.e. $\{i_1, \ldots, i_n\} = \{1, \ldots, ar(f)\}$, and $i_j \neq i_k$ for any $j \neq k$. a term t is o-normal if and only if $pr_o(t) = [p_1, \ldots, p_n]$ and there is a p_i such that $(t)_{p_j} \in \Sigma$ for each $j < i$ and $(t)_{p_k} \in V$ for each $k > i$. A TRS is called o-normal if each left-hand side is o-normal.

Example 5.2.3 Let o be an on-demand E-strategy map such that $o(inf) = nil$, $o(0) = nil$, $o(2nd) = [1]$ and $o(cons) = [2,1]$. $t \equiv 2nd(cons(x, cons(y,z)))$ is o-normal because $pr_o(t) = [\varepsilon, 1, 1.2, 1.2.2, 1.2.1, 1.1]$ and $(t)_\varepsilon, (t)_1, (t)_{1.2} \in \Sigma$, $(t)_{1.2.2}, (t)_{1.2.1}, (t)_{1.1} \in V$. Since inf is of course o-normal too, R_{l_2} is an o-normal TRS (Fig. 5.5).

$$
R_{l_2} = \begin{cases} inf \to cons(0, inf) \\ 2nd(cons(x, cons(y,z))) \to y. \end{cases}
$$

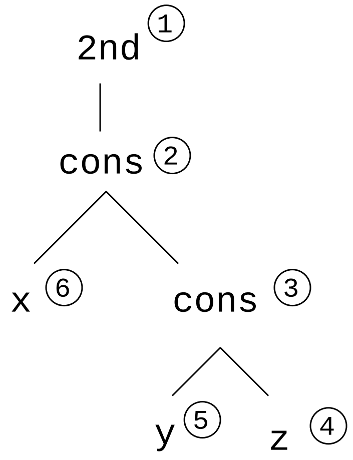

Figure 5.5: o-normal

It is easy to see that an o-normal term is left-normal if $o(f) = [1, \ldots, ar(f)]$. A function $tr_o : T \to T$ is defined as follows: $tr_o(x) = x$, $tr_o(f(t_1, \ldots, t_m)) = f(tr_o(t_{i_1}), \ldots, tr_o(t_{i_n}))$ for $o(f) = [i_1, \ldots, i_n]$. Then t is an o-normal if and only if $tr_o(t)$ is left-normal.

We show a useful property of the on-demand E-strategy about the shape of evaluated terms: each evaluated term is always a root-stable form. We need some restrictions for a TRS : left-linear, constructor and o-normal.

Even if a symbol of l at each position of $O_\Sigma(l)$ is equivalent to that of t, i.e. $(l)_p = (t)_p$ for any $p \in O_\Sigma(l)$, t is not always an instance of l thanks to the non-linear variable. For example, $f(a, b)$ is not an instance of $f(x, x)$, but $(l)_1 = f = (t)_1$ where $\{1\} \in O_\Sigma(l)$. If there is only linear variable in l, t is an instance of l.

Lemma 5.2.4 *Let l be a linear term and t a term. If $(l)_p = (t)_p$ for any $p \in O_\Sigma(l)$, t is an instance of l.*

Proof. Trivial. □

Lemma 5.2.5 *Let t and l be terms and R a constructor TRS. If there is a position $p \in O_\Sigma(l)$ such that $t|_p$ is a root-stable form, $(l)_p \neq (t)_p$ and $(l)_q = (t)_q$ for any $q \in O_\Sigma(l)$ such that $p = q.q'$ for some q', then there is no reduction sequence $t \to_R^* l\theta$ for any theta.*

Proof. All symbols $(l)_q = (t)_q$ except the root symbol are not defined symbols. So we cannot modify the symbol of $(t)_p$ and cannot reduce it into an instance of l. □

Lemma 5.2.6 *Let o be an on-demand map, t a term and l a linear o-normal term where $pr_o(l) = [p_1, \ldots, p_n]$. If $(l)_{p_i} \in V$ and $(l)_{p_j} = (t)_{p_j}$ for each $j < i$, then t is an instance of l.*

Proof. From the definition of o-normal term $(l)_{p_k} \in V$ for each $k \geq i$ because $(l)_{p_i} \in V$. So $(l)_p = (t)_p$ for any $p \in O_\Sigma(l)$ and t is an instance of l from Lemma 5.2.4. □

Lemma 5.2.7 *If a term t is a root-stable form, any term reduced from the term is a root-stable term, i.e. $t \to_R^* s \Rightarrow s$ is a root-stable form.*

Proof. If s is not a root-stable form, there exists a redex u such that $s \to_R^* u$. It is inconsistent with the root-stability of t because $t \to_R^* s \to_R^* u$. □

Theorem 5.2.8 *Let R be a left-linear constructor TRS and (φ, o) E-strategy maps. If R is o-normal and $\varphi(f)$ has 0 for any $f \in D(R)$, each evaluated term is a root-stable form.*

Proof. We prove this claim by the induction of the structure of an evaluated term t. t is trivially in head normal form if the root symbol is a variable or a constructor symbol. We assume $t \equiv f(t_1, \ldots, t_n)$ and $f \in D(R)$. From the assumption the local strategy $\varphi(f)$ has 0. Since the list $\varphi(f)$ gets empty, the element 0 is removed which means that there is a term $s = match(u)$ for some u, $s \notin Red(R)$ and $s \to_R^* t$ where there is no rewrite step at the root position in this sequence. Any term reduced from a root-stable form is a root-stable form Lemma 5.2.7. So it is sufficient to be prove that the term s is a root-stable form.

We prove this claim by the induction of the structure of a term s. Hereafter we use the notation of Definition 5.1.4 of the on-demand E-strategy. *match* returns a term only if l or T becomes empty. If l becomes empty, there is a left-hand side l' such that $(l')_p = (t)_p$ for each $p\ inO_\Sigma(l')$. s is a redex from Lemma 5.2.4. It is inconsistent with the fact $s \notin Red(R)$. Hence T should become empty. If for some left-hand side l', the condition of Lemma 5.2.6 holds, i.e. $(l')_{p_i} \in V$ and $(l')_{p_j} = (s)_{p_j}$ for each $j < i$, on-demand matching should be succeed and the list become empty. So there is no such a case. Since T becomes empty, we can say that for each left-hand side l', the condition of Lemma 5.2.6 holds, i.e. there is a position $p \in O_\Sigma(l')$ such that $s|_p$ is a root-stable form, $(l')_p \neq (s)_p$ and $(l')_q = (s)_q$ for any $q \in O_\Sigma(l')$ such that $p = q.q'$ for some q'. Therefore s can not be reduced into an instance $l'\theta$ for any $l' \to r' \in R$, which means that s is a root-stable form. $\square$

Example 5.2.9 Again consider TRS R_{b_2}.

$$R_{b_2} = \begin{cases} \wedge(x,0) \to 0 \\ \wedge(0,1) \to 0 \\ \wedge(1,1) \to 1 \end{cases}$$

Let $\varphi(\wedge) = [0]$ and $o(\wedge) = [2,1]$. R_{b_2} is left-linear, constructor and o-normal. Then each evaluated term is guaranteed to be a root-stable form from Theorem 5.2.8.

Because the multiple occurrences make no sense for an on-demand strategy list, a number of on-demand maps we should consider are finite. Hence we can easily obtain a suitable on-demand strategy if it exists. Note that a TRS does not always have an on-demand map such that the TRS is o-normal.

Example 5.2.10 Consider TRS R_{b_3} which is another definition of conjunction.

$$R_{b_3} = \begin{cases} \wedge(x,0) \to 0 \\ \wedge(0,x) \to 0 \\ \wedge(1,1) \to 1 \end{cases}$$

The left-hand side $\wedge(x,0)$ is not o-normal for $o(\wedge) = [1,2]$ and the left-hand side $\wedge(0,x)$ is not o-normal for $o(\wedge) = [2,1]$. Hence there is no on-demand map such that R_{b_3} is o-normal.

Without the restriction of the left-linear and constructor TRS, Theorem 5.2.8 does not hold. We show two counterexamples.

Example 5.2.11 Let $R = \{f(x,x) \to a, a \to b\}$, $o(f) = [1,2], o(a) = o(b) = nil$ and $\varphi(f) = \varphi(a) = [0], \varphi(b) = nil$. Although $f(a,b) = red(f(a,b))$, the term $f(a,b)$ is not a root-stable form since $f(a,b) \to_R f(b,b)$.

Example 5.2.12 Let $R = \{f(f(a)) \to a, f(b) \to f(a)\}$, $\varphi(f) = [0]$, $\varphi(a) = \varphi(b) = nil$ and $o(f) = [1], o(a) = o(b) = nil$. Although $f(f(b)) = red(f(f(b)))$, $f(f(b))$ is not a root-stable form since $f(f(b)) \to_R f(f(a))$. Note that the subterm $f(b)$ cannot be evaluated because $(f(f(b)))_1 = f = (f(f(a)))_1$.

5.3 Conclusive remarks

If each evaluated term is always a root-stable form for an (on-demand) E-strategy, we can construct a correct (on-demand) E-strategy, such that $\varphi'(f) = \varphi(f)@[1,\ldots,ar(f)]$ for each $f \in \Sigma$. However, this construction is not good for an example of an infinite list. If the second argument of : is in the list $\varphi(:)$, we can not avoid an infinite sequence.

$$\begin{aligned} red(inf) &= eval(inf, [0]) \\ &= red(0 : inf) \\ &= eval(0 : inf, [1,2]) \\ &= eval(red(0) : inf, [2]) \\ &= eval(0 : red(inf), []) \\ &= \cdots . \end{aligned}$$

So we cannot reduce a term having *inf* into a normal form without an infinite sequence.

$$red(hd(inf)) = eval(hd(inf), [1,0]) = eval(hd(red(inf)), [0]) = \cdots .$$

Hence, in order to get a normal form by the on-demand E-strategy, we need to define a meta function $RED : T \to T$ as follows:

$$\begin{aligned} RED(t) &= \begin{cases} t' & \text{if } t' = red(t) \text{ and } t' \in V \\ EVAL(t', o(f)) & \text{if } t' = red(t) \text{ and } (t')_\epsilon = f \in \Sigma \end{cases} \\ EVAL(t,l) &= \begin{cases} t & \text{if } l = nil \\ EVAL(t[RED(t|_i)]_i, l') & \text{if } l = i : l' \end{cases} \end{aligned}$$

We can easily prove that this function RED is correct, i.e. an evaluated term is always a normal form, if each evaluated term by the function red is always a root-stable form. Since $t' = red(t)$ is a root-stable form, the term $f(t'_1,\ldots,t'_n)$ is also a root-stable form by Lemma 5.2.7 if $t' \equiv f(t_1,\ldots,t_n)$. From the induction hypothesis, each $t'_i = RED(t_i)$ is a normal form. A root-stable form is not a redex itself. So $f(t'_1,\ldots t'_n)$ has no redex and is a normal form.

The behavior of the function RED is very similar to the functional strategy [PE93] if $\varphi(f) = [0]$ for $f \in D(R)$, $\varphi(g) = []$ for $g \notin D(R)$ and $o(f) = [1,2,\ldots,ar(f)]$ for $f \in \Sigma$. The main difference between them is the timing of reduction of a subterm. The on-demand E-strategy reduces a subterm when the symbol differs from that of each left-hand side of rules. The functional strategy does when the root symbol of the subterm is a defined symbol. It can be said that the functional strategy is more eager than the on-demand E-strategy with the above operation. If we restrict a TRS to a constructor one, both strategies are almost the same, i.e. $t = red(s)$ if and only if the function strategy reduces s into t. It is known that the functional strategy is normalizing for a left-normal orthogonal TRS. We can easily show that for E-strategy maps (φ, o) where $t = red(s)$ if and only if $tr_o(t) = red(tr_o(s))$ for a TRS $tr_o(R) = \{tr_o(l) \to tr_o(r) \mid l \to r \in R\}$ and (φ', o') such that $o'(f) = [1,2,\ldots,ar(f)]$ for $f \in \Sigma$ and φ' defined properly. Therefore the function RED is normalizing for orthogonal constructor and o-normal TRS if $\varphi(f) = [0]$ for $f \in D(R)$ and $\varphi(g) = []$ for $g \notin D(R)$.

Chapter 6

Application to CafeOBJ

The OBJ languages, OBJ2[FGJM85], OBJ3[GWMFJ00], CafeOBJ[DF98, NSF98] etc, are broad spectrum algebraic programming and specification languages, based on order-sorted equational logic, possibly enriched with other logics, such as rewriting logic, hidden equational logic, or first order logic. All the OBJ languages are rigorously based upon a logical system; more precisely, they are logical languages, in the sense that their programs are sets of sentences in some logical system, and their operational semantics is given by deduction in that logical system. They have been successfully used for research and teaching in software design and specification, rapid prototyping, theorem proving, user interface design, and hardware verification, among other things[GM97, GWMFJ00, DF98]. In this chapter, we introduce how to do verification in CafeOBJ system and show some applications of the E-strategy.

6.1 CafeOBJ module

We show an example of a CafeOBJ module specifying an addition on natural numbers.

```
mod! NAT+ {
  [ Nat ]
  op 0     : -> Nat         {strat: ()}
  op s_    : Nat -> Nat     {strat: (1)}
  op _+_   : Nat Nat -> Nat {strat: (1 2 0)}
  vars N M : Nat
  eq N + 0 = N .
  eq N + s M = s(N + M) .
}
```

The operator symbols are `0`, `s` and `+` declared after `op`. An operator symbol may contain under-bars "`_`". Under-bars reserve the places where arguments are inserted. Equations are pairs of terms declared after `eq` where `N` and `M` are variables. When the system tries to reduce a given term, equations are regarded as left-to-right rewrite rules. So it can be said that `NAT+` corresponds to R_{n_1}.

$$R_{n_1} = \begin{cases} +(x,0) \to x \\ +(x,s(y)) \to s(+(x,y)) \end{cases}$$

Local strategies are operator attributes declared after `strat:`. For example, the local strategy of the operation symbol `+` is `(1 2 0)}`, which instructs CafeOBJ system to "(for a given term `t1 + t2`,) reduce the term `t1` first, `t2` second and finally rewrite the whole term if possible". This corresponds to $\varphi(+) = [1, 2, 0]$.

`redece` is the command to reduce an input term. We show the trace of the reduction of the term `(s 0 + 0) + (0 + 0)` in the context of `NAT+`.

```
NAT+> reduce (s 0 + 0) + (0 + 0) .
s 0: Nat
```

`[1]`, `[2]` and `[3]` are rewrite steps from the input term `(s 0 + 0) + (0 + 0)` to the output term `s 0`. Since the root symbol of the term is `+`, the evaluation first follows the local strategy `(1 2 0)`. The first argument `s 0 + 0` is evaluated first (`[1]`), the second argument `0 + 0` next (`[2]`) and then the whole term `s 0 + 0` is rewritten into the term `s 0` (`[3]`).

6.2 Proof by CafeOBJ

In CafeOBJ system, there is a special symbol `_==_` which is a predicate to verify that two terms are equivalent to each other. When we input `reduce s == t`, CafeOBJ system reduce the both side terms `s`, `t` and check whether the results terms are equivalent or not. If it so, the whole term `s == t` is reduced into `true: Bool`. If not so, it is reduced into `false: Bool`. When we want to verify whether an equation $s = t$ is true in a specification, open the specification and reduce s==t. For example, the following is a proof score of CafeOBJ to prove $1 + 2 = 2 + 1$:

```
open NAT+
reduce s 0 + s s 0 == s s 0 + s 0 .
close
```

where `open` and `close` are commands for opening or closing a module. When we input the above lines, CafeOBJ system outputs the following:

```
-- opening module NAT+.. done.
-- reduce in NAT+ : s 0 + s (s 0) == s (s 0) + s 0
true : Bool
NAT+>
```

The effects of declarations supplied in between open and close commands are temporary. An open command creates a new module. This module contains a copy of declarations in the given module. Until the module is closed, all the declared sorts, equations, etc. are added to the new module. This open/closing mechanism is an ecological tool to (1) make hypotheses, (2) prove, by evaluation, your favorite theorems upon the hypotheses, (3) finish the proof, and delete the hypotheses, (1') start again with new hypotheses. For example, consider the module NAT+ again.

```
mod! NAT+ {
  [ Nat ]
  op 0     : -> Nat         {strat: ()}
  op s_    : Nat -> Nat     {strat: (1)}
  op _+_   : Nat Nat -> Nat {strat: (1 2 0)}
  vars N M : Nat
  eq N + 0 = N .
  eq N + s M = s(N + M) .
}
```

Suppose you want to show that `0` is a left identity of `+` which means that `N + 0` is also equal to `N` for any term `N` in `Nat`. Using the standard structural induction, you can prove it easily, by showing (1) `0 + 0 = 0`, and (2) for any `N: Nat`, if `0 + N = N`, `0 + s N = s N`. A score of this proof may be written as

```
  open NAT+
  op n : -> Nat .
  reduce 0 + 0 == 0 .
  eq 0 + n = n .
  reduce 0 + s n == s n .
  close
```

Then CafeOBJ system outputs as follows:

```
-- reduce in NAT+ : 0 + 0 == 0
true : Bool

-- reduce in NAT+ : 0 + s n == s n
true : Bool
```

After opening, module elements and various commands can be input. The above score adds a new constant `n` (to represent "any" natural number) and an equation (for induction hypothesis), and invokes reduction commands (base case and induction step). The system evaluates both `0 + 0 == 0` and `0 + s n == s n` into `true`.

Although this proof can be shown by induction straightforwardly, there are examples which cannot be proven easily like this. We show an example which we need to some lemmata to prove it, that is the commutativity of `+`: "`N + M == M + N` for any terms `N, M: Nat`". Now we write a proof score for this statement straightforwardly as follows:

```
  open NAT+
  ops n m : -> Nat .
  reduce 0 + m == m + 0 .
  eq n + m = m + n .
  reduce s n + m == m + s n .
  close
```

where we prove it by induction on `N`. Then CafeOBJ system outputs as follows:

```
-- reduce in NAT+ : 0 + m == m + 0
false : Bool

-- reduce in NAT+ : s n + m == m + s n
false : Bool
```

Here both equations we want to prove were not proven. Note that although an equation is true if CafeOBJ system answers `true`, it is not always false that if the system answers `false`. it can be seen that they coincide with each other for a terminating and confluence TRS. So if the system returns `false`, we have to check what cause went wrong by reducing both sides of the equation `0 + m == m + 0`.

```
NAT+>   op m : -> Nat .
NAT+>   reduce 0 + m .
-- reduce in NAT+ : 0 + m
0 + m : Nat
NAT+>   reduce m + 0 .
-- reduce in NAT+ : m + 0
m : Nat
NAT+>
```

Here they were reduced into `0 + m` and `m`. So we can see that it is necessary to show "`0 + M = M` for any `M: Nat`" as a lemma. The equation is true as we already showed above. Next we reduce both sides of `s n + m == m + s n` and get the lemma "`(s N) + M = s(N + M)` for any `N, M: Nat`" to prove. This can be shown straightforwardly. Hence we can use these two lemmata to prove the commutativity of `+`.

```
  open NAT+

  vars N M : Nat .
  eq 0 + N =  N .
  eq (s N) + M = s(N + M) .

  ops n m : -> Nat .
  reduce 0 + m == m + 0 .
  eq n + m = m + n .
  reduce s n + m == m + s n .
  close
```

The system returns `true` for the both reduction.

```
-- reduce in NAT+ : 0 + m == m + 0
true : Bool

-- reduce in NAT+ : s n + m == m + s n
true : Bool
```

6.3 Proof on infinite lists

To find out necessary lemmata, it is important that reduction terminates. The specification in the above section is terminating in the view of TRSs. So any reduction terminates and we can see result terms to find out lemmata. In a specification which is not terminating, the E-strategy may play an important role. The following is the CafeOBJ module which specifies natural numbers lists with infinite elements.

```
mod! LIST { protecting(NAT+)
  [ List ]
  op hd    : List -> Nat       {strat: (1 0)}
  op tl    : List -> List      {strat: (1 0)}
  op _::_  : Nat List -> List  {strat: (1)}
  op inf   : Nat -> List       {strat: (0)}
  var N    : Nat
  var L    : List
  eq hd(N :: L) = N .
  eq tl(N :: L) = L .
  eq inf(N) = N :: (inf (s N)).
}
```

`nth` returns the n-th element of a list for the first argument n. `::` is a constructor symbol of lists. `inf` stands for an infinite list `0 :: 0 :: 0 ::`$\cdots$. Note that the local strategy of the constructor symbol `::` is `(1)` which does not have 2 standing for the second argument. It can be seen that each evaluate term is guaranteed to be a root-stable form from theorems in the above chapters. Under this E-strategy we can show some equation where `inf` must be reduced, for example, `(hd(inf(N)) :: tl(inf(N))) == inf(N) .`

```
-- reduce in %LIST : hd(inf(N)) :: tl(inf(N)) == inf(N)
false : Bool
```

Since we could not prove it straightforwardly, we check what the both sides are reduced into.

```
-- reduce in LIST : hd(inf(N)) :: tl(inf(N))
```

```
N :: tl(inf(N)) : List

-- reduce in LIST : inf(N)
N :: inf(s N) : List
```

We prove the equation `tl(inf(N)) == inf(s N)`.

```
-- reduce in LIST : tl(inf(N)) == inf(s N)
true : Bool
```

So we can use the lemma `M :: tl(inf(N)) = M :: inf(s N)` to prove the goal equation.

```
eq M :: tl(inf(N)) = M :: inf(s N) .
reduce hd(inf(N)) :: tl(inf(N)) == inf(N) .
```

Then CafeOBJ system outputs the following:

```
-- reduce in LIST : hd(inf(N)) :: tl(inf(N)) == inf(N)
true : Bool
```

6.4 Order-sorted terms

In CafeOBJ specifications, signatures have sorts and constructing terms is restricted to the sort information. Consider the following example of a CafeOBJ specification.

```
mod! A {
  [ Zero NzNat < Nat, Bool ]
  op 0     : -> Zero
  op s_    : Nat -> NzNat
  op true  : -> Bool
  op false : -> Bool
  op eqn   : Nat Nat -> Bool

  vars N M : Nat .

  eq eqn(0,   0  ) = true .
  eq eqn(s N, 0  ) = false .
  eq eqn(0,   s M) = false .
  eq eqn(s N, s M) = eqn(N, M) .
}
```

The module A has sorts `Zero`, `NzNat`, `Nat` and `Bool` which is written in brackets `[...]`. The signature `eq` has the string of sorts `Nat Nat` as the arity and the sort `Bool` as the value, which means that for a term `eq(t1, t2)`, the argument terms `t1` and `t2` must have the sort `Nat` and the whole term has the sort `Bool`. A term which ignore sorts is not permitted, like `eq(true, 0)`. The sorts `Zero` and `NzNat` are subsorts of the sort `Nat`, which means that `Nat` includes `Zero` and `NzNat` if sorts are regarded as sets. A term having the sort `Zero` or `NzNat` also has the sort `Nat`. Hence the term `eq(0, s 0)` is allowed to be an order-sorted term.

We let μ a replacement map satisfying the condition of Theorem 4.3.39 such that $\mu(f) = \emptyset$ for each symbol $f \in \Sigma$. If an E-strategy map is defined as $\varphi(f) = [0]$ for each defined symbol $f \in D(R)$ and $\varphi(g) = []$ for each non-defined symbol $f \notin D(R)$, the E-strategy map φ is correct because $\varphi \in \Phi(\mu)$. This means that each evaluated term is normal form under the following local strategies:

```
mod! A {
  [ Zero NzNat < Nat, Bool ]
  op 0     : -> Zero          {strat: ()}
  op s_    : Nat -> NzNat     {strat: ()}
  op true  : -> Bool          {strat: ()}
  op false : -> Bool          {strat: ()}
  op eq    : Nat Nat -> Bool  {strat: (0)}
}
```

6.5 Ground totality

The sort information may help us to analyze which set of arguments a function symbol is ground total on more efficiently. Consider the following example.

```
mod! NATLIST {
  [ Nat, List ]
  op 0     : -> Nat
  op s_    : Nat -> Nat
  op _+_   : Nat Nat -> Nat
  op nil   : -> List
  op _::_  : Nat List -> List
  op hd    : List -> Nat
  op tl    : List -> List

  vars M N : Nat
  var L    : List

  eq N + 0 = N .
  eq N + s M = s(N + M) .

  eq hd(N :: L) = N .
  eq tl(N :: L) = L .

}
```

For the pure TRS, without sort information, abstracted from the above, `+` is not ground total on $\{2\}$ because ill-formed terms may occur with no restriction of sorts. `0 + nil` is not a redex even although `nil` is a ground normal term. This term is not well-formed in the above specification. So it is sufficient to check the ground term whose sort is `Nat` for the arguments of `+`, and we can show that `+` is ground total on $\{2\}$, considering the sort information. This means that for each ground term the evaluated term is normal form under the following local strategies:

```
mod! NATLIST {
  [ Nat, List ]
  op 0     : -> Nat            {strat: ()}
  op s_    : Nat -> Nat        {strat: (1)}
  op _+_   : Nat Nat -> Nat    {strat: (2 0)}
  op nil   : -> List           {strat: ()}
  op _::_  : Nat List -> List  {strat: (1 2 0)}
  op hd    : List -> Nat       {strat: (1 0)}
  op tl    : List -> List      {strat: (1 0)}
}
```

6.6 Defining suitable local strategies

By the results of Chapter 4, we can get a following method to define suitable local strategies for a given replacement map.

1. Define $\varphi(f) = [1, 2, \ldots, ar(f), 0]$ for each $f \in D(R)$ and define $\varphi(f) = [1, 2, \ldots, ar(f)]$ for each $f \notin D(R)$
2. Analyze $V_R(f) = \{\vec{k_i}\}$ and move them after 0. $\varphi(f) = [\vec{j_i}, 0, \vec{k_i}]$
3. Analyze $S_\mu(f) = \{\vec{k'_i}\}$ and put them on the head of the list. $\varphi(f) = [\vec{k'_i}, \vec{j_i}, 0, \vec{k_i}]$
4. Simplify the local strategies.

First, define all local strategies as the leftmost innermost one. Next, Analyze variable arguments and move the arguments after 0. Finally, Analyze strict arguments and move the arguments before 0. We can easily get the variable arguments and the strict arguments for a term rewriting system and a replacement map. The following is an example of applying this method to R_{n_3}.

$$R_{n_3} = \begin{cases} +(0, x) \to x \\ +(s(x), y) \to s(+(x, y)) \\ \times(0, x) \to 0 \\ \times(s(x), y) \to +(\times(x, y), y) \\ double(x) \to x + x \end{cases}$$

For easy understanding we assume the trivial replacement map.

$$1. \begin{cases} \varphi(+) = [1, 2, 0] \\ \varphi(\times) = [1, 2, 0] \\ \varphi(s) = [1] \\ \varphi(double) = [1, 0] \end{cases} \Rightarrow 2. \begin{cases} \varphi(+) = [1, 0, 2] \\ \varphi(\times) = [1, 0, 2] \\ \varphi(s) = [1] \\ \varphi(double) = [0, 1] \end{cases} \Rightarrow 3. \begin{cases} \varphi(+) = [2, 1, 0, 2] \\ \varphi(\times) = [1, 0, 2] \\ \varphi(s) = [1] \\ \varphi(double) = [1, 0, 1] \end{cases}$$

$$\Rightarrow 4. \begin{cases} \varphi(+) = [2, 1, 0] \\ \varphi(\times) = [1, 0, 2] \\ \varphi(s) = [1] \\ \varphi(\mathit{double}) = [1, 0] \end{cases}$$

The E-strategy 1 is of course correct and it is the leftmost innermost strategy. The E-strategy 2 is also correct because the arguments moved after 0 are variable arguments and the Theorem 4.3.9 holds. The E-strategy 3 is correct from the Theorem 4.4.4 and preserves the termination behavior, i.e. for each term t if $\mathit{red}(t)$ terminates for the E-strategy 2, it also terminates for the E-strategy 3 from the Theorem 4.4.7. For the final E-strategy 4, reduction is identical with that of the E-strategy 3 because of the Theorem 3.3.1.

When we restrict a target term to a ground term, the method is improved a little. We add analyzing whether a function symbol is ground total on a set of arguments or not.

1. Define $\varphi(f) = [1, 2, \ldots, ar(f), 0]$ for each $f \in D(R)$ and define $\varphi(f) = [1, 2, \ldots, ar(f)]$ for each $f \notin D(R)$

2′. Analyze inductively complete arguments and if $I = \{\vec{i_k}\}$ is so, $\varphi(f) = [\vec{i_k}, 0]$
If it does not exist, analyze $V_R(f) = \{\vec{k_i}\}$ and move them after 0.
$\varphi(f) = [\vec{j_i}, 0, \vec{k_i}]$

3. Analyze $S_\mu(f) = \{\vec{k'_i}\}$ and put them on the head of the list.
$\varphi(f) = [\vec{k'_i}, \vec{j_i}, 0, \vec{k_i}]$

4. Simplify the local strategies.

$$1. \begin{cases} \varphi(+) = [1, 2, 0] \\ \varphi(\times) = [1, 2, 0] \\ \varphi(s) = [1] \\ \varphi(\mathit{double}) = [1, 0] \end{cases} \Rightarrow 2. \begin{cases} \varphi(+) = [1, 0] \\ \varphi(\times) = [1, 0] \\ \varphi(s) = [1] \\ \varphi(\mathit{double}) = [0] \end{cases} \Rightarrow 3. \begin{cases} \varphi(+) = [2, 1, 0] \\ \varphi(\times) = [1, 0] \\ \varphi(s) = [1] \\ \varphi(\mathit{double}) = [1, 0] \end{cases}$$

$$\Rightarrow 4. \begin{cases} \varphi(+) = [2, 1, 0] \\ \varphi(\times) = [1, 0] \\ \varphi(s) = [1] \\ \varphi(\mathit{double}) = [1, 0] \end{cases}$$

Since the above methods can be easily implemented, we propose that we adopt local strategies obtained by each of these methods as default strategies. A default strategy is a local strategy which the system defines automatically when a user omits writing it. It is easier to define a replacement set than a local strategy list. So we define a new function symbol's attribute `replacement`. We write after a replacement set of the function symbol `replacement`. For example, we consider the following module NATLIST:

```
mod! LIST {
  [ Nat List ]
  op 0     : -> Nat           {replacement: {}    }
  op s_    : Nat -> Nat       {replacement: {1}   }
  op _+_   : Nat Nat -> Nat   {replacement: {1, 2}}
  op _*_   : Nat Nat -> Nat   {replacement: {1, 2}}
  op 2*_   : Nat -> Nat        {replacement: {1}   }

  op hd    : List -> Nat       {replacement: {1}   }
  op tl    : List -> List      {replacement: {1}   }
  op _::_  : Nat List -> List  {replacement: {1}   }
  op inf   : Nat -> List       {replacement: {1}   }

  vars N M : Nat
  var  L   : List

  eq N + 0 = N .
  eq N + s M = s(N + M) .

  eq N * 0 = 0 .
  eq N * s M = s(N + M) .

  eq hd(N :: L) = N .
  eq tl(N :: L) = L .
  eq inf(N) = N :: (inf (s N)).
}
```

Sets after `replacement` correspond to replacement sets. When we try to reduce a term in this module, the `reduce` command reduce the term according to the following local strategies:

```
  op 0     : -> Nat            {strat: ()     }
  op s_    : Nat -> Nat        {strat: (1)    }
  op _+_   : Nat Nat -> Nat    {strat: (2 1 0)}
  op _*_   : Nat Nat -> Nat    {strat: (1 0 2)}
  op 2*_   : Nat -> Nat        {strat: (1 0)  }

  op hd    : List -> Nat       {strat: (1 0)  }
  op tl    : List -> List      {strat: (1 0)  }
  op _::_  : Nat List -> List  {strat: (1)    }
  op inf   : Nat -> List       {strat: (0 1)  }
```

These local strategies are computed by the above first methods. If we assume input terms are always ground, we adopt the second method to compute local strategies.

```
  op 0     : -> Nat            {strat: ()     }
  op s_    : Nat -> Nat        {strat: (1)    }
  op _+_   : Nat Nat -> Nat    {strat: (2 1 0)}
  op _*_   : Nat Nat -> Nat    {strat: (1 0)  }
  op 2*_   : Nat -> Nat        {strat: (1 0)  }

  op hd    : List -> Nat       {strat: (1 0)  }
  op tl    : List -> List      {strat: (1 0)  }
  op _::_  : Nat List -> List  {strat: (1)    }
  op inf   : Nat -> List       {strat: (0)    }
```

6.7 On-demand E-strategy

In CafeOBJ, the on-demand E-strategy is defined by local strategies with negative integers. For negative integers $-i$, the i-th arguments are not evaluated until so forced. Such an argument is forced to evaluate if it is involved in matching. The subterm `inf(s 0)` of the term `0 :: inf(s 0)` is not evaluated until so forced if `{strat: (-1 -2)}` is given to `::`. Trying to match the term `2nd(0 :: inf(s 0))` to the pattern `2nd(N :: (M :: L))` to rewrite the term, `inf(s 0)` must be rewritten and otherwise the matching fails. On the other hand, on the term `2nd(0 :: s 0 :: inf(s s 0))`, the term `inf(s s 0)` does not have to be rewritten because it is not involved in the matching. The matching succeeds and the whole term `2nd(0 :: s 0 :: inf (s s 0))` is rewritten into `0`. In the above chapter, we formalized the on-demand E-strategy by a pair of an E-strategy map and an on-demand map which stand for order of reduction and order of matching respectively. Hence we must prepare a new attribute for declare an on-demand E-strategy maps. For example, we write `{strats:` l_1 l_2`}` as $\varphi(f) = l_1$ and $o(f) = l_2$.

```
mod! LIST2 { protecting(NAT+)
  [ List ]
  op hd    : List -> Nat       {strats: (1 0) (1)}
  op tl    : List -> List      {strats: (1 0) (1)}
  op 2nd   : List -> Nat       {strats: (1 0) (1)}
  op _::_  : Nat List -> List  {strats: (1)   (2)}
  op inf   : -> List           {strats: (0)   () }
  vars N M : Nat
  var L    : List
  eq hd(N :: L) = N .
  eq tl(N :: L) = L .
  eq 2nd(N :: (M :: L)) = M .
  eq inf(N) = N :: (inf (s N)).
}
```

The symbol `2nd` returns the second element of an input list. Under this E-strategy we can also show some equation where `inf` must be reduced, for example, `hd(tl(inf(N))) == 2nd(inf(N))`. Unfortunately, there is no implementation with the on-demand E-strategy. If it is implemented according to our formalization, the system will output the following:

```
-- reduce in LIST : hd(tl(inf(N))) == 2nd(inf(N))  .
true : Bool
```

Chapter 7

Conclusion

In this thesis we investigated evaluation strategies for term rewriting systems.

Safety condition We generalized the condition, called the safety condition, on which some existing research results of the E-strategy holds. We showed that on the condition the E-strategy satisfied the fundamental property: if a term is a result of evaluating by the E-strategy, it cannot be reduced more, i.e. $red(t) = red(red(t))$ (Theorem 3.1.2). In the latter of this chapter, we proposed how to define local strategies with the safety condition. It is trivial that an E-strategy is safe if the condition proposed in the existing researches holds: each local strategy list ends in 0. We gave a weaker condition which is sufficient for the safety condition by analyzing occurrences of variables in the left-hand sides of the rules of a given TRS (Theorem 3.4.4). Since the condition is weaker than that of some existing results of the E-strategy, it means that we had extended the scopes of the existing research results depending on it, such as evaluated flags (Theorem 3.2.5), simplifying local strategies (Theorem 3.3.1 and Theorem 3.3.2) and the normalizability of the E-strategy [Nag99].

Relation with context-sensitive rewriting In Section 4.2 we showed that context-sensitive rewriting suited for analyzing some properties of the E-strategy. We first showed the method to obtain context-sensitive rewriting from a given E-strategy. The definition of this method is a very simple because a local strategy is given as a natural numbers list and a replacement set is given as a positive numbers set. We just forget the order and the element 0 of a given list to obtain the corresponding replacement set. We showed that reduction of an E-strategy is covered with the corresponding context-sensitive rewriting (Theorem 4.2.3) and reduction of an E-strategy terminates if so does the context-sensitive rewriting (Theorem 4.2.4).

Correctness However some literatures had proposed conditions on which each evaluated term are always in normal form, i.e. the E-strategy is correct, these conditions are too strong to treat some example, such as the specification of infinite lists. If we take a suitable context-sensitive rewriting, we can treat such examples by the corresponding E-strategy obtained by our method. In Section 4.3, we generalized the correctness of the E-strategy. We proposed the μ-correctness of the E-strategy as each evaluated term are always a μ-normal form. In order to obtain μ-correct E-strategies, we analyzed which arguments are unnecessary for a redex. We proved that such a variable argument can be evaluated later with keeping μ-correctness (Theorem 4.3.9). The latter of this section we introduced and proposed some conditions about the shape of μ-normal forms for a redex (Lemma 4.3.11), a root-stable form (Proposition 4.3.15) and a normal from (Theorem 4.3.21). By combining these conditions with the condition of the μ-correctness we can obtain conditions on which evaluated terms are in sets of terms which have some useful properties. Moreover we proposed two useful conditions especially for a normal form. One was for order-sorted terms. In the CafeOBJ specification language, signatures have sorts with an order and constructing terms is restricted by the sort information. We proposed a hierarchical order on terms for a given order-sorted TRS and showed a useful property that ignoring arguments having some sorts has no effect for rewriting (Theorem 4.3.39). This means that such arguments can be ignored with keeping the correctness of the E-strategy. The other was for ground terms. We proposed ground totality of function symbols and of replacement maps. We showed that if a replacement map is ground total, a set of μ-normal forms coincides with that of normal forms Hence we also showed that the E-strategy is correct even if only such arguments are evaluated (Theorem 4.3.29).

Strictness In Section 4.4, we analyzed an argument such that eager evaluation of the argument does not change the termination behavior, called a strict argument. In order to obtain strict arguments of a function symbol, we analyzed behavior of variable arguments: which variable arguments disappear by applying rewrite rules, and defined a function which takes a function symbol and returns strict arguments. We proved that an argument obtained by our strictness analysis function is strict, i.e. does not change the termination behavior (Theorem 4.4.7). By the result of correctness and strictness analyses, we gave methods to obtain a suitable E-strategy for a given context-sensitive rewriting in Section 6.6. Note that analyzing variable arguments, ground totality and strict arguments by our methods is very easy to implement and the above theorems hold for any TRS. Hence the methods we proposed in Section 6.6 can be used to compute a default strategy, which the system (such as CafeOBJ) gives us automatically when we omit to write local strategies.

On-demand E-strategy Finally we discussed the on-demand E-strategy in Chapter 5. The on-demand E-strategy has been proposed to express true lazy evaluation [NSF98], since we cannot do it with the ordinary E-strategy. In this paper we formalized the on-demand E-strategy by a pair of an E-strategy map and an on-demand map, which stand for an order of reduction and an order of matching. We defined a reduction function of the on-demand E-strategy as an extension of the definition of the ordinary E-strategy by adding new function *match*. For the on-demand E-strategy an order of matching is important because a target term may be changed while doing on-demand matching. We showed some examples that we can treat well owing to an on-demand map. Moreover we showed a useful property of the on-demand E-strategy about the shape of evaluated terms: each evaluated term is always a root-stable form. For obtain this property we defined an o-normal term for an on-demand map o by generalizing a left-normal term. We also defined a meta function RED to obtain a correctness E-strategy by the on-demand E-strategy. We compared the on-demand E-strategy to the functional strategy which is adopted by some lazy functional languages.

Bibliography

[BN98] F. Baader and T. Nipkow, "Term rewriting and all that," Cambridge University Press, 1998.

[DF98] R. Diaconescu and K. Futatsugi, "CafeOBJ report," World Scientific, 1998.

[Dol01] E. Dolstra, *Functional stratego,* In E. Visser, editor, Proceedings of the Second Stratego Users Day, 10-17, 2001.

[Eke98] S. Eker,*Term rewriting with operator evaluation strategies,* In C. Kirchner and H. Kirchner, editors, Proceedings of the 2rd International Workshop on Rewriting Logic and its Applications, WRLA'98, Electronic Notes in Theoretical Computer Science, 1-20, 1998.

[FGJM85] K. Futatsugi, J.A. Goguen, J.-P. Jouannaud and J. Meseguer, *Principles of OBJ2*, In Proceedings of the 12th ACM Symposium on Principles of Programming Languages, 52-66, 1985.

[FN97] K. Futatsugi, A. Nakagawa. *An Overview of* CAFE *Specification Environment - An algebraic approack for creating, verifying, and maintaining formal specification over networks* -, In Proceedings of 1st International Conference on Formal Engineering Methods, 1985.

[FR99] M. C. F. Ferreira and A. L. Ribeiro, *Context-Sensitive AC-rewriting*, In Proceedings of the 10th International Conference on Rewriting Techniques and Applications, Trento, LNCS 1631, 286-300, 1999.

[GM92] J.A. Goguen and J. Meseuer, *Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations,* Theoretical Computer Science, 105(2), 217-273, 1992.

[GM99] J. Giesl and A. Middeldorp, *Transforming Context-Sensitive Rewrite Systems*, In Proceedings of the 10th International Conference on Rewriting Techniques and Applications, Trento, LNCS 1631, 271-285, 1999.

[GM97] J.A. Goguen and G. Malcolm, *Algebraic Semantics of Imperative Programs,* MIT Press, 1997.

[GWMFJ00] J.A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi and J. P. Jouannaud, *Introducig OBJ,* Software Engineering with OBJ: algebraic specification in action, edited by Goguen and Malcolm, Kluwer, 2000.

[HL91] G. Huet and J.-J. Lévy, *Computations in orthogonal rewriting systems, I and II,* in computational logic, essays in honor of Alan Robinson, eds. J.-L. Lassez and G. Plotkin, MIT Press, 396-443, 1991.

[JK86] J.-P. Jouannaud and E. Kounalis. *Automatic proofs by induction in equational theories without constructors,* In Proceedings, Symposium on Logic in Computer Science, Cambridge, Massachusetts, IEEE Computer Society, 358-366, 1986.

[KNZ97] D. Kapur, P. Narendran and H. Zhang, *On sufficient completeness and related properties of term rewriting systems,* Acta Informatica, 24(4): 395-415, 1987.

[Luc96] S. Lucas, *Termination of context-sensitive rewriting by rewriting,* In Proceeding of the 23rd international colloquium on automata, languages and programming, LNCS 1099, 122-133, 1996.

[Luc98] S. Lucas, *Context-sensitive computations in functional and functional logic programs,* Journal of Functional and Logic Programming, 1998(1), 1-61, 1998.

[Luc00] S. Lucas, *Context-sensitive rewriting strategies,* Technical Report DSIC-II/7/00, 56 pages, Departamento de Sistemas Informaticos y Computacion, UPV, 2000.

[Luc01a] S. Lucas, *Context-Sensitive Rewriting, Lazy Rewriting, and On-demand Rewriting,* Proc. of WFLP'01, 197-210, Technical Report 2017, Christian-Albrecht-Universität zu Kiel, 2001.

[Luc01b] S. Lucas, *Termination of on-demand rewriting and termination of OBJ programs,* Proc. of 3rd International Conference on Principles and Practice of Declarative Programming, PPDP'01, 82-93, 2001.

[Mid97] A. Middeldorp, *Call by need computations to root-stable form,* In Conference Record of the 24th Annual ACM symposium on Principles of Programming Languages, POPL'97, 94-105, 2001.

[MTHM97] R. Milner, M. Tofte, R. Harper and D. MacQueen, *The Definition of Standard ML (Revised),* MIT Press, 1997.

[Nag99] T. Nagaya, "Reduction strategy for term rewriting system," Ph.D. thesis, Japan Advanced Institute of Science and Technology, 1999.

[HWA+90] P. Hudak, P. L. Wadeler, Arvind, B. Boutel, J. Fairbairn, J. Fasel, K. Hammond, J. Hughes, T. Johnsson, R. Kieburtz, R. S. Nikhil, S. L. Peyton Jones, M. Reeve, D. Wise and J. Young, *Report on the Functional Programming Language Haskell,* Technical report, Department of Computer Science, Glasgow University, 1990.

[NMOF98] T. Nagaya, M .Matsumoto, K. Ogata, and K. Futatsugi, *How to give local strategies to function symbols for equality of two implementations of the E-strategy with and without evaluated flags,* Proceedings of Asian Symposium on Computer Mathematics, 71-81, 1998.

[NSF98] A. Nakagawa, T. Sawada and K. Futatsugi, CafeOBJ user's manual – ver.1.4.–, http://www.ldl.jaist.ac.jp/cafeobj/, 1998.

[NF01] M. Nakamura and K. Futatsugi *Completeness and strictness analysis for the evaluation strategy,* The international workshop on rewriting in proof and computation, RPC'01, 80-89, 2001.

[NO00] M. Nakamura and K. Ogata *The evaluation strategy for head normal form with and without on-demand flags,* In K. Futatsugi, editor, Proceedings of the 3rd International Workshop on Rewriting Logic and its Applications, WRLA2000, Electronic Notes in Theoretical Computer Science, 211-227, 2000.

[O'Do77] M.J. O'Donnell, *Computing in systems described by equations,*Springer-Verlag, Lecture note in computer science 58, 1977.

[OF97] T. Ogata and K. Futatsugi, *Implementation of term rewritings with the evaluation strategy,* In H. Glaser and P. Hartel, editors, Proceedings of the 9th International Symposium on Programming Languages, Implementations, Logics and Programs, PLILP'97, LNCS 1292, 225-239, 1997.

[OF00] T. Ogata and K. Futatsugi, *Operational semantics of rewriting with the on-demand evaluation strategy,* Proceedings of the 2000 ACM Symposium on Applied Computing, 756-763, 2000.

[PE93] R. Plasmeijer and M. Eekelen, "Functional programming and parallel Graph rewriting," ADDISON-WESLEY, 1993.

[Pol01] J. Pol, *Just-in-time: On Strategy Annotations,* Technical Report SEN-R0105, CWI, Amsterdam.

[Vis99a] E. Visser, *The Stratego Library,* Institute of Information and Computing Sciences, Utrecht University, 0.5 edition, 1999-2001.

[Vis99b] E. Visser, *The Stratego Reference Manual,* Institute of Information and Computing Sciences, Utrecht University, 0.5 edition, 1999-2001.

[Vis99c] E. Visser, *The Stratego Tutorial,* Institute of Information and Computing Sciences, Utrecht University, 0.5 edition, 1999-2001.

[Vis99] E. Visser, *The Stratego Compiler,* Institute of Information and Computing Sciences, Utrecht University, 0.5 edition, Technical Documentation. Latest version available at `http://www.stratego-language.org`, 1999-2001.

[Vis01] E. Visser, *A survey of rewriting strategies in program transformation systems,* In B. Gramlich and S. Lucas, editors, Workshop on Reduction Strategies in Rewriting and Programming, volume 57 of Electronic Notes in Theoretical Computer Science, Utrecht, The Netherlands, Elsevier Science Publishers, 2001.

[Zan97] H. Zantema, *Termination of context-sensitive rewriting,* In Proceedings of the 8th international conference on rewriting techniques and applications, LNCS 1232, 172-186, 1997.

Publications

[1] M. Nakamura and K. Futatsugi *Completeness and strictness analysis for the evaluation strategy,* The international workshop on rewriting in proof and computation, RPC'01, 80-89, 2001.

[2] M. Nakamura and K. Futatsugi *Completeness and strictness analysis for the evaluation strategy,* Workshop on Foundation of Software Engineering 2001, FOSE2001, 23-34, 2001.(in Japanese)

[3] M. Nakamura and K. Futatsugi *The on-demand context-sensitive rewriting,* Technical Report of IEICE, SS2000-33, 17-24, 2001.(in Japanese)

[4] M. Nakamura and K. Ogata *The evaluation strategy for head normal form with and without on-demand flags,* In K. Futatsugi, editor, Proceedings of the 3rd International Workshop on Rewriting Logic and its Applications, WRLA2000, Electronic Notes in Theoretical Computer Science, 211-227, 2000.

[5] K. Kusakari, M. Nakamura and Y. Toyama *Argument filtering transformation,* Proceedings of international conference on principles and practice of declarative programming '99, Lecture notes in computer science 1702, 47-61, 1999.

[6] M. Nakamura, K. Kusakari and Y. Toyama *On proving termination by general dummy elimination* IEICE transactions on information and systems, Vol. J82-D-I, No.10, 1225-1231, 1999.(in Japanese),

[7] M. Nakamura and Y. Toyama, *On proving termination by general dummy elimination* Technical report of IEICE, COMP98-58(1998-11), 57–64, 1998.(in Japanese)