Efficient Rewriting Techniques

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op woensdag 1 april 2009 om 16.00

door

Muck Joost van Weerdenburg

geboren te 's-Hertogenbosch

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. J.F. Groote
en
prof.dr. M.G.J. van den Brand

Copromotor:
dr.ir. M.A. Reniers

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

IPA dissertation series 2009-06.

A catalogue record is available from the Eindhoven University of Technology Library.

ISBN 978-90-386-1671-1

Typeset using LaTeX.
Printed by Printservice Technische Universiteit Eindhoven.
Cover design by Muck van Weerdenburg. (And no, it's not a gallow.)

Contents

Chapter 1

Introduction

A rewrite system is a simple system consisting of a set of rules that determine how certain structures (terms) can be transformed. For example, one could have a rule $a \to b$ that states that any occurrence of a may be replaced by b. Thus, if one has a term $f(a, c)$ we can apply this rule to obtain $f(b, c)$.

The most typical practical application of rewrite systems is as implementation of (simple) functional languages. In such languages one often defines functions by giving transformations of patterns. For example, take the factorial function !. A functional way to define the factorial is as follows.

$$\begin{array}{lcl} 0! & = & 1 \\ (n+1)! & = & (n+1) \cdot n! \end{array}$$

By considering these equations as rewrite rules from left to right (i.e. replacing the $=$ by $\to$), one can calculate the value of a factorial (assuming there are also rules to rewrite terms of the form $n + m$ and $n \cdot m$). For instance, if we are interested in the value of $((0 + 1) + 1)!$ (which is a complex way of writing $2!$), we can apply rewrite rule $(n + 1)! \to (n + 1) \cdot n!$ to get $((0 + 1) + 1) \cdot (0 + 1)!$ and once more to get $((0 + 1) + 1) \cdot ((0 + 1) \cdot 0!)$. Applying rewrite rule $0! \to 1$ and standard arithmetic we get $2 \cdot 1 \cdot 1$ and eventually 2. The latter can usually not be rewritten any further and is therefore called a *normal form* (of $((0 + 1) + 1)!$). When using rewrite systems as implementation of a language, one's goal is to obtain a normal form of a given term as fast as possible.

Of course, in order to do so, the rewrite system itself still has to be implemented on an actual computer. Such an implementation is called a *rewriter*. As there are often many ways to rewrite a term – $3! + 4!$ can be rewritten to both $6 + 4!$ and $3! + 24$ – and not all ways are as efficient as others, this implementation follows some *strategy* to determine the order in which rewrite rules need to be applied. A rewrite system usually allows for a great deal of freedom with respect to choosing such a strategy.

Probably the most well-known applications of rewrite systems for the evaluation of functional programs are those of Haskell [Pey03, HHPW07, LS93] and Clean [Pla95]. In [Pey87] a detailed documentation for implementation of such systems is given. The main topics seem to be common for implementations of functional languages: translation to an intermediate language, combining patterns of rewrite rules into *match trees* [Pey87, Aug85, Sch88] and using an abstract machine to perform the actual rewriting (using the intermediate language). Note that in order to be able to explicitly handle sharing of subterms, these implementations often use the more general *graph rewriting* instead of term rewriting [BvEG+87].

Another useful application of (term) rewrite systems is in symbolic reasoning. For example, rewriters are used for program transformation [Vis05] (e.g. ASF/SDF [vdBvDH+01], Stratego/XT [Vis04]) and to support theorem proving [Nip89] and model checking (e.g. in μCRL [BFG+01], mCRL2 [GMvWU07, GMR+08]). In the latter settings rewriting is typically used to (in some sense) simplify expressions.

To illustrate, one of the most essential aspects in, for instance, generation of state spaces in the μCRL and mCRL2 toolsets is finding all solutions of a boolean expression. This is needed to obtain an explicit state space from the used compact symbolic representation. Here rewriting is used to simplify an expression as $n < 0$ (with n a natural number) to false such that it is easy to determine that there are no solutions to $n < 0$ without having to investigate all possible values of n.

An important difference between functional-program evaluation and symbolic reasoning is that for the former one typically only rewrites closed terms (i.e. terms without variables) while it is essential to rewrite open terms for the latter.

The origin of this thesis lies in the development of the mCRL2 toolset. As it is meant as a successor to the μCRL toolset, there are a lot of similarities between the two. Specifically, both have rewriters that use either an innermost strategy or just-in-time strategies [vdP01]. The just-in-time rewriters are an improvement over innermost rewriters in that they can avoid rewriting unneeded subterms while with an innermost strategy one always rewrites every subterm regardless of whether it is needed. Also, both toolsets make use of so-called *compiling* rewriters. These are rewriters that, on initialisation, create and compile code that is specialised for the given rewrite system. Instead of having one general piece of code that can rewrite with arbitrary rules, compiling rewriters have a specialised piece of code for each function symbol.

In [vW07] we evaluated the performance of the mCRL2 compiling rewriters by comparing them with various other rewriters or functional language evaluators. One of the observations made was that in some tests the implementation of the just-in-time rewriter was significantly slower than the innermost rewriter. This is somewhat peculiar as just-in-time rewriting never requires more rewrite steps than innermost rewriting. The reason that it was nevertheless slower is that an innermost rewriter can be (and was) implemented in such a way that it does not unnecessarily construct terms in cases where each subterm is relevant. This is possible because of the way an innermost rewriter traverses terms. When using the

just-in-time rewriter, which usually traverses terms differently, these unnecessary constructions were not avoided.

A similar observation was made in [HFA+96], where various implementations of functional languages were compared to each other using one specific benchmark. The implementations that employed *lazy evaluation* [FW76, HJ76] were, in general, slower than those that used *eager evaluation*.

Another observation that can be made based on the results from [vW07] is that taking the best of both mCRL2 rewriters gives results that, looking at execution times, come reasonably close to the other considered systems but still are a factor 2 or 3 behind. It is hard to determine precisely what the cause is due to the many differences between implementations. However, one likely cause is the use of lazy evaluation in the implementations of functional languages as Haskell and Clean versus a much more eager evaluation in the mCRL2 rewriters. Even though usage of just-in-time strategies allows one to avoid rewriting subterms, once you start rewriting one, you only stop when it is in normal form.

In this thesis we consider three aspects of implementing term rewriting. First of all, in Chapter 3, we look at matching: the process of finding the values of variables of a pattern such that it is equal to a given term. For example, when we have a rewrite rule $\mathrm{inc}(n) \to n + 1$ and we try to apply this rule to term $\mathrm{inc}(2)$, we need to match the pattern $\mathrm{inc}(n)$ to this term $\mathrm{inc}(2)$. Obviously, by taking n equal to 2 we have a match and can apply the rewrite rule to obtain $2 + 1$.

When there is overlap in the patterns of rewrite rules, as is the case for the definition of $+$ on "Peano numbers" below, we want to avoid repeating the same activities such as determining the value for n.

$$\begin{array}{lcl} n + 0 & \to & n \\ n + S(m) & \to & S(n + m) \end{array}$$

The typical approach to do this is combining a collection of rewrite rules (typically with the same head symbol in the pattern) into a simple structure called a *match tree*. Such a match tree can then be used to efficiently determine which of the original rewrite rules can be applied to a given term.

We find that definitions of match trees found in literature [Pey87, Aug85, Sch88, Mar92] are either ad-hoc or too complex. The complexity seems to be the result of giving a direct translation of a set of rewrite rules to a match tree. Instead of this "all-in-one" approach, we wish to establish a relatively simple formal definition of match trees by separating the construction of match trees from a rule and the combination of these match trees into one single match tree.

Another aspect we consider is temporary-term construction. During rewriting there usually are many intermediate results. For example, in the rewriting of 2! above, we had intermediate results such as $((0 + 1) + 1) \cdot (0 + 1)!$ and $2 \cdot 1 \cdot 1$. Now assume that we are rewriting term $\mathrm{inc}(2)$ and know that subterm 2 is in normal form. If we then apply rewrite rule $\mathrm{inc}(n) \to n + 1$ we obtain the temporary term

$2 + 1$, which we can rewrite further. However, we somehow wish to remember that 2 is already in normal form and avoid rewriting it again (as part of rewriting $2+1$). Rewriting normal forms can be quite costly even though effectively nothing is done.

Also, in some cases we know that specific parts of temporary terms are certainly going to be rewritten later on. For example, in the rewrite rules for $+$ above, we must always rewrite the second argument of $+$ to be able to determine whether or not it can be written as 0 or as $S(m)$, for some m. So instead of constructing a temporary term $n + (2 + 3)$, for example, we can immediately rewrite $2 + 3$ to 5 and construct $n + 5$.

In Chapter 4 we give two methods. One allows for efficient annotation of terms to keep track of normal forms. The other determines which parts of temporary terms will be rewritten later on in any case (similar to strictness analysis [PvE93]). By employing these methods, just-in-time rewriting requires significantly less term constructions and the gap between innermost and just-in-time rewriting has been closed in those cases where just-in-time rewriting was (significantly) slower than innermost rewriting in [vW07].

The final aspect we consider is rewrite strategies. As mentioned before, in order to implement a rewrite system one needs a strategy that determines what action to perform in what situation. One method of defining such strategies is by using *just-in-time* strategies [vdP01]. An essential characteristic of these just-in-time strategies is that they facilitate rewriting in such a manner that avoids rewriting certain subterms that are not relevant in obtaining a normal form. For example, the term $\mathit{if}(b, t, u)$ typically rewrites to t or u depending on whether b rewrites to true or false. In either case, however, only one of the terms t and u is relevant for the result. Just-in-time enables one to only rewrite either t or u, depending on which of them is going to be the result.

A downside of just-in-time strategies is that if a (sub)term is rewritten, it is always completely rewritten to normal form. There are cases where this is not necessary and possibly even results in infinite behaviour. Sometimes one only needs to know the head symbol of a subterm to continue rewriting. A typical example is a function to obtain the length of a list. For the calculation of this function it is not relevant what the elements of the lists are; only the structure of the list matters.

Our goal in Chapter 5 is to give an extension of just-in-time strategies that does not have this downside and a method that automatically generates strategies from rewrite rules using this extension. We do this by adding the possibility to rewrite a subterm to *stable-head form*, which is a form where the head symbol will not change with further rewriting. With this it is possible to rewrite only as much as is needed to perform matching. For example, to match $f(t)$ to a pattern $f(g(x))$, you only need to rewrite subterm t until you know that the head symbol of its normal form is g or not; t itself need not be in normal form.

For the same reason E-strategies [GWM+93, OF97], used in OBJ [GWJMJ00],

were extended to *on-demand* E-strategies [GWJMJ00, OF00]. These E-strategies are similar to just-in-time strategies only without control over which rewrite rules are tried and without restrictions on the contents of a strategy. The latter is due to the fact that E-strategies are meant to be used with terms that need not have a normal form, while just-in-time strategies are meant to ensure a normal form. We do not consider the on-demand extension, or E-strategies in general, sufficient because it lacks control over the application of rewrite rules as well as control over when terms are rewritten to, for example, stable-head form. It is only possible to mark an argument as a whole as "on-demand" and only during matching subterms are rewritten as needed.

To conclude, we investigate the performance of various methods we discussed in Chapter 6 and give some concluding remarks in Chapter 7. Note that all proofs of theorems and properties are given in the appendices.

Chapter 2

Preliminaries

2.1 Rewriting

We introduce the notations and concepts we use throughout this thesis.

Terms

A **signature** Σ consists of a set of **variables** $\mathbb{V}$ and a set of **function symbols** $\mathbb{F}$. With $x \in \mathbb{V}$ and $f \in \mathbb{F}$, **terms** $\mathbb{T}$ are defined as follows (with $t \in \mathbb{T}$).

$$t \;:=\; x \;\mid\; f(t, \ldots, t)$$

Given an term $f(t_1, \ldots, t_n)$ we call f the **head symbol** and term t_i the ith **argument**. The head symbol of a term t is denoted by $\mathrm{hs}(t)$ (where we leave the case $t \in \mathbb{V}$ undefined[1]). Each function symbol f has an **arity** $\mathrm{ar}(f)$ which indicates the number of arguments it takes. We write $\mathrm{var}(t)$ for the set of variables that occur in t. That is, $\mathrm{var}(x) = \{x\}$ and $\mathrm{var}(f(t_1, \ldots, t_n)) = \bigcup_{1 \leq i \leq n} \mathrm{var}(t_i)$.

Sometimes we also consider **applicative** terms. These have the following syntax.

$$t \;:=\; x \;\mid\; f \;\mid\; t(t)$$

We often write $f(t_1, \ldots, t_n)$ for $f(t_1)(t_2)\ldots(t_n)$.

Substitutions are functions $\sigma, \tau, \ldots : \mathbb{V} \to \mathbb{T}$. A substitution σ applied to term t, notation $t\sigma$, is defined as $\sigma(x)$ if $t = x$ and $f(t_1\sigma, \ldots, t_n\sigma)$ if $t = f(t_1, \ldots, t_n)$. We write $\sigma[x \mapsto t]$ for the substitution that maps x to t and $y \neq x$ to $\sigma(y)$. A term t matches a term u if, and only if, $\exists_\sigma(t = u\sigma)$. We also refer to u as a **pattern**.

To facilitate operations on subterms we inductively define **positions** ($\mathbb{P}$) as follows. A position is either ϵ (the empty position) or an **index** i (from $1, 2, \ldots$)

[1] With undefined we do *not* mean that a function is partial but that we simply do not know what the precise value is for some arguments.

combined with a position π, notation $i \cdot \pi$. We lift $\cdot$ to an associative operator on positions with ϵ as its unit element and often write just i for the position $i \cdot \epsilon$. We write the subterm of t at position π as $t|_\pi$ and we write term t with the subterm at position π replaced by u as $t[u]_\pi$. These operations are defined as follows.

$$\begin{array}{lcll} t|_\epsilon & = & t & \\ f(t_1,\ldots,t_n)|_{i\cdot\pi} & = & t_i|_\pi & \text{if } 1 \leq i \leq n \end{array}$$

$$\begin{array}{lcll} t[u]_\epsilon & = & u & \\ x[u]_{i\cdot\pi} & = & x & \\ f(t_1,\ldots,t_n)[u]_{i\cdot\pi} & = & f(t_1,\ldots,t_{i-1},t_i[u]_\pi,t_{i+1},\ldots,t_n) & \text{if } i \leq n \\ f(t_1,\ldots,t_n)[u]_{i\cdot\pi} & = & f(t_1,\ldots,t_n) & \text{if } i > n \end{array}$$

We also use a generalisation of $t[u]_\pi$ that takes a set of positions Π and a function mapping positions to terms. This generalisation is defined as follows. Note that it only applies a substitution at position $\pi \in \Pi$ if there is no (smaller) prefix of π in Π.

$$\begin{array}{lcll} t[\varphi]_\emptyset & = & t & \\ t[\varphi]_{\Pi\cup\{\pi,\pi\cdot\pi'\}} & = & t[\varphi]_{\Pi\cup\{\pi\}} & \\ t[\varphi]_{\Pi\cup\{\pi\}} & = & (t[\varphi(\pi)]_\pi)[\varphi]_{\Pi\setminus\{\pi\}} & \text{if } \neg\exists_{\pi'}(\pi\cdot\pi' \in \Pi) \\ t[\varphi]_\Pi & = & t[\varphi]_{\Pi\cap\mathrm{pos}(t)} & \end{array}$$

We write $\overline{\Pi}$ to denote the set of positions that have a prefix π in Π (i.e. $\overline{\Pi} = \{\pi\cdot\pi' \;:\; \pi \in \Pi\}$).

We write $\mathrm{pos}(t)$ for all the **valid** positions for term t. A position π is a valid position for t if $t|_\pi$ is well-defined. That is, $\mathrm{pos}(x) = \{\epsilon\}$ and $\mathrm{pos}(f(t_1,\ldots,t_n)) = \{\epsilon\} \cup \{i\cdot\pi \;:\; 1 \leq i \leq n \;\wedge\; \pi \in \mathrm{pos}(t_i)\}$. Also, we write $\mathrm{pos}_\mathrm{f}(t)$ for the set of valid positions of function application in t (i.e. $\mathrm{pos}_\mathrm{f}(t) = \{\pi \;:\; \pi \in \mathrm{pos}(t) \;\wedge\; t|_\pi \notin \mathbb{V}\}$) and $\mathrm{pos}_\mathrm{v}(t)$ for the set of valid positions of variables in t (i.e. $\mathrm{pos}_\mathrm{v}(t) = \{\pi \;:\; \pi \in \mathrm{pos}(t) \;\wedge\; t|_\pi \in \mathbb{V}\}$).

We define the set of positions that indicate the "difference" between a term t and a pattern u, notation $\partial(t,u)$ as follows. Note this notion does not include differences that are the result of multiple occurrences of the same variable in u (i.e. $\partial(f(a,b),f(x,x)) = \emptyset$).

$$\begin{array}{lcll} \partial(t,u) & = & \partial(t,u,\epsilon) & \\ & & & \\ \partial(t,x,\pi) & = & \emptyset & \\ \partial(x,f(t_1,\ldots,t_n),\pi) & = & \{\pi\} & \\ \partial(f(t_1,\ldots,t_n),f(u_1,\ldots,u_n),\pi) & = & \bigcup_{1\leq i\leq n}\partial(t_i,u_i,\pi\cdot i) & \\ \partial(f(t_1,\ldots,t_n),g(u_1,\ldots,u_m),\pi) & = & \{\pi\} & \text{if } f \neq g \end{array}$$

Instead of a pattern u we often use an object that has a (single) pattern (e.g. a rewrite rule as defined below). For example, we write $\partial(t, o)$ for $\partial(t, u)$ if u is the pattern of o.

Rewriting

Rewrite rules are of the form $t \to u$ **if** c, where $t \notin \mathbb{V}$ and $\mathrm{var}(u) \cup \mathrm{var}(c) \subseteq \mathrm{var}(t)$. Term c is the condition of a rewrite rule indicating whether or not the rule may be applied. Often we omit this condition in the case c is syntactically equal to *true*. We often refer to t as the pattern of the rewrite rule. The set of rewrite rules is denoted by $\mathbb{R}$.

We write $\langle \Sigma, \to, \eta \rangle$ for a signature Σ, set of rewrite rules $\to \subseteq \mathbb{T}(\Sigma) \times \mathbb{T}(\Sigma) \times \mathbb{T}(\Sigma)$ and **condition-evaluation function** $\eta : \mathbb{T}(\Sigma) \to \mathbb{B}$ (where $\mathbb{B}$ is the set of booleans) to denote a Term Rewrite System (**TRS**) [DJ90]. The condition-evaluation function η is use to determine whether or not a condition is satisfied. Typically $\eta(c)$, for some condition c, is true when the specific implementation of a rewrite system can rewrite c to true (i.e. an instantiation of type III of [BK86]).

If R is a set of rewrite rules, we often write R_f to denote the subset of R containing all rules from R where f is the head symbol of the left-hand side. If R_f is empty, we call f a **constructor function** (with respect to R).

Let R be a set of rewrite rules. The rewrite relation $\to_R$ on terms is defined as follows.

$$\forall_{t,u}(t \to_R u \;\Leftrightarrow\; \exists_{\pi,\sigma,l\to r \text{ if } c\in R}(t|_\pi = l\sigma \;\wedge\; u = t[r\sigma]_\pi \;\wedge\; \eta(c\sigma)))$$

We also write $\to$ instead of $\to_R$ if no confusion can occur. We write $\to_R^*$ for the reflexive and transitive closure of $\to_R$ and $t \not\to_R$ if there is no u such that $t \to_R u$.

A **normal form** of t is a term u such that $t \to_R^* u$ and $u \not\to_R$. We also refer to u as a *full* normal form. We denote the set of all normal forms of a term t, $\{u : t \to_R^* u \;\wedge\; u \not\to_R\}$, by $\mathrm{nf}_R(t)$ (or just $\mathrm{nf}(t)$).

A **stable-head form** of t is a term u such that $t \to^* u$ and, if $u \notin \mathbb{V}$, there is no v with $u \to_R^* v$ and $\mathrm{hs}(u) \neq \mathrm{hs}(v)$. We denote the set of all stable-head forms of a term t, $\{u : t \to^* u \wedge (u \notin \mathbb{V} \Rightarrow \neg\exists_v(u \to^* v \wedge \mathrm{hs}(u) \neq \mathrm{hs}(v))\}$, by $\mathrm{shf}_R(t)$ (or just $\mathrm{shf}(t)$).

A **head normal form** of t is a term u such that $t \to_R^* u$ and there are no term v, rewrite rule $l \to r$ **if** $c \in R$ and substitution σ such that $u \to_R^* v$, $v = l\sigma$ and $\eta(c\sigma)$. We denote the set of all head normal forms of a term t, $\{u : t \to_R^* u \;\wedge\; \neg\exists_{v,l\to r \text{ if } c\in R,\sigma}(u \to_R^* v \;\wedge\; v = l\sigma \;\wedge\; \eta(c\sigma))\}$, by $\mathrm{hnf}_R(t)$ (or just $\mathrm{hnf}(t)$).

Theorem 2.1.1 *We have that* $t|_\pi \in \mathrm{hnf}(t|_\pi)$ *for all* $\pi \in \mathrm{pos}(t)$ *if, and only if,* $t \in \mathrm{nf}(t)$.

We call a sequence $t_1, t_2, \ldots$ of terms such that $t_i \to_R t_{i+1}$ a **rewrite sequence** of t_1. A term t is **strongly normalising** if all rewrite sequences of t are finite.

We write $t \to_R^\omega$ if t is *not* strongly normalising (i.e. if there is an infinite rewrite sequence of t).

We define the **essential positions** of a rewrite rule ρ, notation esspos(ρ), to be those positions of the left-hand side l of ρ that put restrictions on the terms to which ρ can be applied. That is, all positions of function applications of l are essential as well as positions of variables that either occur more than once in l or occur in the condition of ρ. More formally, esspos is defined as follows.

$$\begin{aligned}\text{esspos}(l \to r \textbf{ if } c) = \quad & \text{pos}_\text{f}(l) \cup \\ & \{\pi \cdot \pi' \ : \ \pi \in \text{pos}_\text{v}(l) \ \wedge \ (\exists_{\pi'' \in \text{pos}_\text{v}(l) \setminus \{\pi\}} (l|_{\pi''} = l|_\pi) \vee \\ & \qquad\qquad l|_\pi \in \text{var}(c))\}\end{aligned}$$

Note that one can argue that variables that occur in a condition do not necessarily play an essential role in matching. For example, in the condition $b \vee \text{true}$ the value of b does not influence the value of the condition itself (assuming that η adheres to standard logic). Although this is a valid argument, we use the above over-approximation for practical purposes.

The following theorem expresses the use of essential positions; replacing subterms at non-essential positions of a rewrite rule ρ does not affect the applicability of ρ.

Theorem 2.1.2

$$\begin{aligned}\forall_{l \to r \textbf{ if } c, t, \Pi} (\Pi \cap \text{esspos}(l \to r \textbf{ if } c) = \emptyset \ \Rightarrow \\ \forall_\varphi (\exists_\sigma (t = l\sigma \ \wedge \ \eta(c\sigma)) \Leftrightarrow \exists_\tau (t[\varphi]_\Pi = l\tau \ \wedge \ \eta(c\tau))))\end{aligned}$$

In this thesis we consider only a specific form of rewriting strategies. These strategies are essentially top-down in that they examine the head symbol for every term that needs to be rewritten and a continuation is determined by the unique strategy that is given for that specific head symbol. Examples of such strategies are innermost and just-in-time.

Sequential Strategy Rewriting

A **sequential strategy** is a list of integer sets and rule sets (i.e. sets with (names of) rewrite rules) that describes the way a rewriter must rewrite a term with a specific head symbol. This notion comes from [vW07] and is based on the notion of strategies in [vdP01].

For example, $[\{1,3\}, \{\alpha, \beta\}, \{2\}]$ is a strategy for some function symbol f that states that first arguments 1 and 3 must be rewritten, then rewrite rules α and β must be tried and finally, if both α and β do not match, the second argument must be rewritten. Such strategies are usually required to be *full* and *in-time* [vdP01]. A strategy for f is full if all argument indices $(1, \ldots, \text{ar}(f))$ and rewrite rules of f occur in it (at least once). It is in-time if each argument is rewritten before it is required for trying a rewrite rule. An argument t_i is required for trying a rule

$\rho = f(t_1, \ldots, t_n) \to r$ **if** c if $i \in \text{esspos}(\rho)$. We denote the strategy for function symbol f by $\varsigma(f)$.

To illustrate, let function *if* have the following typical rewrite rules.

$$\begin{array}{lllll} \alpha: & \mathit{if}(\mathit{true}, x, y) & \to & x \\ \beta: & \mathit{if}(\mathit{false}, x, y) & \to & y \\ \gamma: & \mathit{if}(b, x, x) & \to & x \end{array}$$

The strategy for innermost rewriting would be $[\{1,2,3\}, \{\alpha, \beta, \gamma\}]$. That is, first rewrite all arguments and then try to apply one of the rewrite rules. A JITty rewriter [vdP02], on the other hand, would typically use the (full and in-time) strategy $[\{1\}, \{\alpha\}, \{\beta\}, \{2\}, \{3\}, \{\gamma\}]$.

We typically only consider the first element of a strategy before we look at the rest of the strategy. We therefore write $I \rhd s$ for the strategy s prepended with index set I. Similarly, we write $R \rhd s$ for s prepended with set of rewrite rules R. We write $\mathbb{S}$ for the set of all sequential strategies.

The semantics of sequential strategies is given as follows. Here $\text{rewr}_s : \mathbb{T} \to \mathcal{P}(\mathbb{T})$ and $\text{eval}_s : \mathbb{S} \times \mathbb{T} \to \mathcal{P}(\mathbb{T})$. To cope with infinite reductions we use a fixed-point definition (see Appendix A). We write $\text{hs}^\perp$ for hs where variables are mapped to some unique element $\perp$ and we write $\varsigma^\perp$ for ς extended to include $\perp$ such that $\varsigma^\perp(\perp) = []$. Also, we write $\text{app}_s(R,t)$ for $\{r\sigma : l \to r \textbf{ if } c \in R \wedge t = l\sigma \wedge \text{true} \in \text{rewr}_s(c\sigma)\}$ and $\text{rewrf}_s(t, I)$ for $\{\varphi : \forall_{i \in I}(i \in \text{pos}(t) \Rightarrow \varphi(i) \in \text{rewr}_s(t|_i))\}$.

$$\begin{array}{lcll} \text{rewr}_s(t) & = & \text{eval}_s(\varsigma^\perp(\text{hs}^\perp(t)), t) & \\ & & & \\ \text{eval}_s([], t) & =_\mu & \{t\} & \\ \text{eval}_s(I \rhd s, t) & =_\mu & \bigcup_{\psi \in \text{rewrf}_s(t,I)} \text{eval}_s(s, t[\psi]_I) & \\ \text{eval}_s(R \rhd s, t) & =_\mu & \bigcup_{u \in \text{app}_s(R,t)} \text{rewr}_s(u) & \text{if } \text{app}(R,t) \neq \emptyset \\ \text{eval}_s(R \rhd s, t) & =_\mu & \text{eval}_s(s, t) & \text{if } \text{app}_s(R,t) = \emptyset \end{array}$$

The following corollaries[2] express that rewr_s rewrites and, if the used strategies are full and in-time, results in normal forms.

Corollary 2.1.3 *For all terms t and u such that $u \in \text{rewr}_s(t)$ we have that $t \to^* u$.*

Corollary 2.1.4 *If, for sequential strategy function ς, it holds that $\varsigma(f)$ is full and in-time for all function symbols f, then we have that $\text{rewr}_s(t) \subseteq \text{nf}(t)$ for all terms t.*

Corollary 2.1.5 *If, for sequential strategy function ς, it holds that $\varsigma(f)$ is full and in-time for all function symbols f, then we have that $\text{rewr}_s(t) = \emptyset$ implies that $t \to^\omega$ for all terms t.*

[2] They follow from Theorems in Chapter 5.

Note that this definition of rewr_s corresponds to the one of *norm* in [vdP01] if one restricts it to the same setting (i.e. no conditionals, only one element in each I and R and no infinite rewrite sequences).

Sequential Strategy Generation

Because one might not want to burden users with supplying strategies themselves, we want to generate reasonable strategies from a given set of rewrite rules (i.e. one strategy per function symbol). This is done by observing which arguments need to be rewritten to be able to match a given rule. An argument that is needed for matching by *most* of the rules is added to the strategy, indicating that it needs to be rewritten first. In the case that all arguments of a rule that are essential for matching are rewritten, this rule is added to the strategy. This process continues until all rules and arguments are in the strategy.

More formally, let $\text{dep}(\rho)$ be a function that returns the indices of the arguments that need to be rewritten before matching rule ρ, i.e.

$$\text{dep}(f(t_1,\ldots,t_n) \to r \textbf{ if } c) = \text{esspos}(f(t_1,\ldots,t_n) \to r \textbf{ if } c) \cap \{i \ : \ 1 \leq i \leq n\}$$

Also, let $\text{occ}(i, R)$ be a function that returns the number of rules of a set R that require argument i:

$$\text{occ}(i, R) = \#\{\rho \in R \ : \ i \in \text{dep}(\rho)\}$$

We denote the empty strategy with $[]$ and a set S of argument indices or rewrite rules prepended to a strategy l by $S \rhd_c l$. Here, $\rhd_c$ only adds S to l if S is not empty (i.e. $\emptyset \rhd_c l = l$). A strategy for a finite set of rules R_f is generated with $\text{strat}(R_f, \{1,\ldots,\text{ar}(f)\})$, where $\text{strat}(R, I)$ is defined as follows, for any set of rules $R \subseteq R_f$ and set of indices I (with I the set of argument indices not yet added to the strategy so far and $\uparrow$ the maximum quantifier):

$$\begin{array}{lcl}
\text{strat}(\emptyset, I) & = & I \rhd_c [] \\
\text{strat}(R, I) & = & T \rhd_c J \rhd_c \text{strat}(R \setminus T, I \setminus J) \qquad \text{if } R \neq \emptyset \\
 & & \textbf{where} \quad T = \{\rho \in R \ : \ \text{dep}(\rho) \cap I = \emptyset\}, \\
 & & \qquad\qquad J = \{i \ : \ i \in I \ \wedge \ \text{occ}(i, R \setminus T) = \uparrow_{j \in I} \text{occ}(j, R \setminus T)\}
\end{array}$$

Theorem 2.1.6 *For all sets R_f of rewrite rules for symbol f, we have that $\text{strat}(R_f, \{1,\ldots,\text{ar}(f)\})$ is full and in-time.*

Note that this function can be improved by making J contain at most one element. This avoids rewriting too much in rewrite systems like $\{f(c, x) \to t, f(x, d) \to u\}$ which otherwise results in a strategy that first rewrites both arguments and then tries to apply rules. We have chosen to take the definition as it is given because this is the way it was given in [vW07] and for the evaluation in Chapter 6 the change showed no differences.

For the *if* above we can now calculate strat($\{\alpha, \beta, \gamma\}, \{1, 2, 3\}$). As all rules depend on at least one argument, no rules will be added in the first step. And, as both α and β depend (solely) on the first argument, this argument will be added first. Thus we get $\emptyset \rhd_c \{1\} \rhd_c \text{strat}(\{\alpha, \beta, \gamma\}, \{2, 3\})$. Then, as the first argument is now in the strategy, we can add α and β. Doing so means that there is only one rule left (γ) and it needs both remaining arguments, which we therefore add. This gives us $\emptyset \rhd_c \{1\} \rhd_c \{\alpha, \beta\} \rhd_c \{2, 3\} \rhd_c \text{strat}(\{\gamma\}, \emptyset)$. As only γ remains to be added we get $\emptyset \rhd_c \{1\} \rhd_c \{\alpha, \beta\} \rhd_c \{2, 3\} \rhd_c \{\gamma\} \rhd_c \emptyset \rhd_c \emptyset \rhd_c []$, which is $[\{1\}, \{\alpha, \beta\}, \{2, 3\}, \{\gamma\}]$.

Our approach deviates from the *just-in-time* strategy as defined in [vdP01] and used in [vdP02] in two ways. First of all, we do not require arguments to be rewritten in order. This way we basically get the same strategy as before when we permute the arguments of the *if*. We also do not preserve in any way the order in which rules were specified by the user while *just-in-time* strategies would (as far as a strategy allows this; i.e. where we have $[\ldots, \{\rho_1, \rho_2\}, \ldots]$, with just-in-time strategies we would have $[\ldots, \{\rho_1\}, \{\rho_2\}, \ldots]$ if ρ_1 is specified before ρ_2). The possibility of using sets is mentioned in [vdP01], but not used due to the choice to let *norm* be a function of terms to terms (instead of term to set of terms).

2.2 Compiling Rewriters

A straightforward implementation of a rewriter typically consist of several generic methods. For example, one might have a method that uses some (simple) form of a database to look up what strategy should be applied to a given term, a method that executes a given strategy and a term, a method to match a given pattern to a given term, etc. Due to this generic nature such implementations are typically not very efficient.

The problem with these implementations is that a significant amount of the running time is spent interpreting strategies, patterns etc. To overcome this, one can use so called **compiling rewriters**[3] (as opposed to the **interpreting rewriters** mentioned above). These compiling rewriters have specialised methods for each operation they might need to perform. For example, a compiling rewriter typically has a separate method for each head symbol that executes the strategy for that symbol. Also matching is hard-coded because in such specialised functions you have very specific moments where a particular rewrite rule is matched and possibly applied.

One technique that is useful in implementing (not necessarily compiling) rewriters is that of using **implicit substitutions**. In certain fields (e.g. state-space generation as discussed in Chapter 1) one often has the need to both apply a substitution to a term and then rewriting the result to a normal form. Because both application of substitutions and rewriting require traversal of the given term, it is

[3] The term "compiling" is due to the fact that these rewriters typically generate and compile code as initialisation and then use the generated/compiled code to do the actual rewriting.

hardly efficient to do this twice. With implicit substitutions you call the rewriter with the original term and the substitution and as soon as the rewriter encounters a variable, it applies the substitution.

Note, however, that the use of implicit substitutions puts some requirements on the rewriter. Because a substitution can map a variable to another (or even the same) variable, the rewriter must not apply the implicit substitution on a term to which it has already been applied. For example, if $\sigma(x) = y$ and $\sigma(y) = t$, then $x\sigma = y$ but $x\sigma\sigma = t$. To avoid this, we require that the rewriter never rewrites terms it has rewritten before. In Chapter 4 we discuss how to do the latter.

In this thesis we assume the following structure of a (compiling) rewriter. For each function symbol f, there is one **specialised rewrite function**, rewrite_f, that takes $\text{ar}(f)$ arguments $t_1, \ldots, t_{\text{ar}(f)}$ and returns the normal form of $f(t_1, \ldots, t_{\text{ar}(f)})$. Besides the specialised rewrite functions, there is also one **main rewrite function**, rewrite, that takes a term and calls the appropriate specialised rewrite function for that term. In pseudo-code, this function looks as follows (where we write $\text{arg}_i(f(t_1, \ldots, t_n))$ for t_i, with $1 \leq i \leq n$):

```
function rewrite(t : 𝕋)
    if t ∈ 𝕍 then
        return t
    else
        var f : 𝔽 = hs(t)
        return rewrite_f(arg_1(t), ..., arg_ar(f)(t))
```

A possible implementation to efficiently obtain the function rewrite_f for some f is to use an array of specialised rewrite functions with function symbols as indices. Accessing an element of such an array usually only requires one or two machine instructions.

If one uses applicative terms, we assume that for each function symbol f there are function symbols f_i in the implementation for $0 \leq i \leq \text{ar}(f)$. In this case the only change needed to the code above is to add an additional **if**-statement to check whether the head of the term is a variable or not. If it is, then the return value would be $x(\text{rewrite}(\text{arg}_1(t)), \text{rewrite}(\text{arg}_2(t)), \ldots)$. Also, in this case the initial **if**-statement is no longer required as the **else**-part also works for the case that $t \in \mathbb{V}$.

If one uses implicit substitutions, the case where $t \in \mathbb{V}$ will have an additional **if**-statement to determine whether a value should be substituted for t (and do so if this is the case). In the case that both implicit substitutions and applicative terms are used, the case added for the applicative terms will contain another **if**-statement to take into account the possibly substituted value for the head variable.

The following piece of pseudo-code corresponds to implementation of the main rewrite function if both implicit substitutions and applicative terms are used. Here we assume that σ is the (global) implicit substitution which acts as the identity

for variables that do not have a value assigned to them and assume that variables used as head in a function application have an arity and can be obtained with hs (i.e. $\mathrm{ar}(x)$ is defined and $\mathrm{hs}(x(t)) = x$).

function rewrite($t : \mathbb{T}$)
  var $h : \mathbb{F} \cup \mathbb{V} \;=\; \mathrm{hs}(t)$
  if $h \in \mathbb{V}$ **then**
    var $u : \mathbb{T} = \sigma(h)$
    var $h' : \mathbb{F} \cup \mathbb{V} \;=\; \mathrm{hs}(u)$
    if $h' \in \mathbb{V}$ **then**
      return $h'(\mathrm{arg}_1(u), \ldots, \mathrm{arg}_{\mathrm{ar}(h')}(u),$
        $\mathrm{rewrite}(\mathrm{arg}_1(t)), \ldots, \mathrm{rewrite}(\mathrm{arg}_{\mathrm{ar}(h)}(t)))$
    else
      return $\mathrm{rewrite}_{h'}(\mathrm{arg}_1(u), \ldots, \mathrm{arg}_{\mathrm{ar}(h')}(u),$
        $\mathrm{arg}_1(t), \ldots, \mathrm{arg}_{\mathrm{ar}(h)}(t))$
  else
    return $\mathrm{rewrite}_h(\mathrm{arg}_1(t), \ldots, \mathrm{arg}_{\mathrm{ar}(h)}(t))$

The implementation of a specialised rewrite function of symbol f consist of code that corresponds to the strategy for f as well as code to perform the matching and application of the rewrite rules occurring in that strategy. An example of such an implementation is the following function for the addition with an innermost strategy:

function $\mathrm{rewrite}_+(n, m : \mathbb{T})$
  $n \;:=\; \mathrm{rewrite}(n)$
  $m \;:=\; \mathrm{rewrite}(m)$
  if $m = 0$ **then**
    return n
  else if $\exists_{m' \in \mathbb{T}}(m = S(m'))$ **then**
    return $\mathrm{rewrite}(S(n + m'))$ **where** $S(m') = m$
  else
    return $n + m$

For an innermost rewriter there is a "trick" to make rewriting significantly more efficient. This trick avoids having to construct the term $S(n + m')$ and doing the consequent rewrites on the subterm n and m' which are already in normal form. The trick is to ensure that the arguments of the specialised rewrite function are always in normal form (instead of rewriting them at the beginning of the function). This means that the main rewrite function needs to ensure this by not passing $\mathrm{arg}_i(t)$ as argument but $\mathrm{rewrite}(\mathrm{arg}_i(t))$. It also means that instead of writing $\mathrm{rewrite}(S(n + m'))$ we can write $\mathrm{rewrite}_S(n, m')$.

For non-innermost strategies this trick does not work as an essential part of such strategies is that some arguments need never to be rewritten. In Chapter 4 we

discuss how we can avoid unnecessary term construction and rewriting of rewritten terms in general.

Chapter 3

Match Trees

3.1 Introduction

During term rewriting one of the most substantial activities is matching a term against a pattern. Matching is therefore one of the main areas to focus on in the pursuit of efficiency.

A straightforward implementation of matching tries the available patterns one by one. Consider, for example, the following rewrite rules.

$$\begin{array}{lcl} n+0 & \rightarrow & n \\ n+S(m) & \rightarrow & S(n+m) \end{array}$$

Here a term t could first be matched against $n+0$ and, if this match fails, afterwards against $n+S(m)$. If this term t is of the form $0+S(0)$ then this typically results in the following steps.

1. Try pattern $n+0$
   (a) Check that the head symbol of t is a $+$. This is the case.
   (b) Remember the first argument of this $+$ (i.e. 0) as n.
   (c) Check that the second argument is 0. This is not the case; match of pattern $n+0$ failed.
2. Try pattern $n+S(m)$
   (a) Check that the head symbol of t is a $+$. This is the case.
   (b) Remember the first argument of this $+$ (i.e. 0) as n.
   (c) Check that the head symbol of the second argument is an S. This is the case.
   (d) Remember the argument of this S (i.e. 0) as m.
   (e) Match of pattern $n+S(m)$ succeeded with 0 for n and m.

It is clear that steps 2a and 2b are redundant; the same activities have already been done in steps 1a and 1b. This is the result of the overlap of the patterns $n + 0$ and $n + S(m)$ and it indicates some room for improvement. Note that the names of the variables are not relevant; $n + 0$ and $n' + S(m)$ essentially have the same overlap.

Another (but minor) drawback is that term t is inspected twice. Depending on the implementation details this might also be costly.

In order to take advantage of overlap in patterns and to ensure that a term is only inspected once, some implementations (e.g. [vW07, vdBHKO02, Vit96]) use some form of *match trees*. These match trees are automaton-like structures that dictate how matching should precede. In the example above such a match tree could be as follows.

- Check that the head symbol of t is a $+$.
  - If so, remember the first argument of this $+$ as n and check whether the second argument is 0.
    - If so, pattern $n + 0$ matched successfully.
    - If not, check that the head symbol of the second argument is an S.
      - If so, remember the argument of this s as m; pattern $n + S(m)$ matched successfully.
      - If not, no pattern matches.
  - If not, no pattern matches.

With this tree we perform every step only once.

For this simple example the effect is perhaps not very significant, but functions such as equality benefit significantly more from the use of match trees. Assume, for example, a sort S containing the (constructor) elements s_1, s_2, s_3 and s_4. To define an equality function on S one needs 16 rules (for every pair in $S \times S$).[1] More generally, if sort S has n constructors, one needs n^2 rules. By combining these rules into a specific tree structure, we can test for a match in the order of n.

Of course, we must also take into account the fact that these match trees must be constructed from the rewrite rules. However, in typical settings the set of rewrite rules is fixed. This means that construction of match trees can be done once (as preprocessing) and its efficiency is therefore not so much relevant during rewriting.

The goal of this chapter is to give a relatively simple formal definition of match trees that are suitable for efficient matching in a wide range of settings. To this end we introduce one functions to construct match trees and one to match using these match trees. We consider rules with nonlinear left-hand sides (i.e. left-hand sides in which variables may occur more than once), conditional rules, applicative

[1] Note that many languages allow for more compact notations by assuming an order on rules. Such features are in general not safe when rewriting with open terms. See Section 3.3.3 for more details.

rules (i.e. rules on terms with partial applications; e.g. $f(x)$ and $f(x, y)$ are both allowed for one symbol f) and ordered rules.

This method of using match trees that we consider here is similar to the ones used in the ASF+SDF [vdBvDH+01] rewriters [vdBHKO02] and ELAN [Vit96]. For rules with linear left-hand sides (i.e. left-hand sides in which variables occur at most once), algorithms to create such trees can be found in [Pey87, Aug85, Sch88, Mar92]. The one in [Sch88] is more similar to our approach in that it introduces an intermediate language with constructs specifically tailored for matching while the others are more ad-hoc translations to the final implementation language. The construction of the match trees in [Sch88] is less intuitive than ours in our opinion. This is due to the fact that they construct match trees directly from a set of rewrite rules while we construct trees per rewrite rule and combine them afterwards.

Our match trees also allow nonlinear rules (at no additional runtime cost for linear rules), conditional rewrite rules (also in [Sch88] and ASF+SDF), applicative terms and priorities (often also available in a limited form in the others). Note that in ASF+SDF nonlinear rules are also allowed, but converted to linear rules, which requires additional side conditions to rules to express the nonlinear requirements. This effectively means that matching (of the linear pattern) is performed before checking the nonlinear requirements, instead of doing the latter during matching. Which method performs better greatly depends on what is being matched (as well as the precise implementation of matching).

3.2 Match Trees

We wish to introduce match trees as an alternative for matching rewrite rules separately. The idea is that there is a function γ to construct a match tree from a set of rewrite rules R and a function μ that matches t using a match tree T. Match tree T can be seen as a "program" that is executed by μ. With the division construction and usage we can compute all relevant match trees once and repeatedly use them during rewriting. As such, this construction only contributes a constant amount to the performance of rewriting and is typically negligible.

To give a more precise definition of our goal, we consider the following specification. We write $\mathbb{M}$ for the set of **match trees**, the concrete elements of which are defined as we go along. Note that we only consider finite sets of rewrite rules (i.e. $\mathcal{F}(\mathbb{R})$), which is sufficient in practice.

Specification μ, γ.

$$\gamma : \mathcal{F}(\mathbb{R}) \to \mathbb{M}$$
$$\mu : \mathbb{M} \times \mathbb{T} \to \mathcal{P}(\mathbb{T})$$

$$\forall_{R \subseteq \mathbb{R}, t \in \mathbb{T}}(\mu(\gamma(R), t) = \{r\sigma \ : \ l \to r \in R \ \wedge \ t = l\sigma\})$$

That is, γ applied to a set of rewrite rules R is a match tree. Using such a match tree as argument of μ together with a term t gives the set of all results

of possible applications of rules from R on t. Of course, in practice, we are only interested in one result. What this single result should be depends on the specific setting, however. For example, when imposing an order on rewrite rules (as in Section 3.3.3) one typically only wants the "greatest" rules to be applied while without such an order it is common to be satisfied with any arbitrary choice. We therefore do not consider this choice at this moment.

Single Rule Match Trees

To get an idea of what kind of elements should make up match trees we look at the straightforward matching of a single rewrite rule. Consider the following rewrite rule:

$$\text{remove}(x, x \rhd l) \to \text{remove}(x, l)$$

An attempt to apply this rule to a term t would typically follow the process depicted in Fig. 3.1.

Note that this process uses a left-most depth-first traversal through the term t. A characteristic of such a search is efficiently implementable with a stack containing the information for resuming work on a term after considering one of its subterms. This will therefore also be a core element of our matching function. Of course, we could also use positions as in Fig. 3.1. In practice, however, the stack solution seems to be a more obvious choice.

Instead of the left-most depth-first traversal one might also desire a different kind of traversal. We do not consider such cases here, but the strategy trees of Chapter 5 suggest that this might be useful. Also, if one desires a depth-first traversal where the traversal order of the arguments can be determined solely on the function symbol, then it is quite straightforward to adapt the method given here.

So, reflecting the left-most depth-first traversal, we implement match function μ using a stack containing terms that will be matched using a match tree. Initially this stack will only contain the term t to be matched. We use the same notation for stacks as for lists. That is, $[t_1, \ldots, t_n]$ is the stack containing terms $t_1, \ldots, t_n$ where t_1 is at the top and t_n at the bottom. We write $t \rhd s$ for the stack s with term t added on top. We refer to the top of the stack as the **current term**. The set of **stacks** of terms is denoted by $\mathbb{S}(\mathbb{T})$.

Besides the stack of terms, we also need a substitution to remember which variables of the pattern are instantiated with which terms. Then, considering the process in Fig. 3.1, we define the following (initial) match tree constructors. The superscript 1 of S and M is to differentiate them from the final versions introduced later.

- $\mathsf{F}(f, T, U)$, with $f \in \mathbb{F}$ and $T, U \in \mathbb{M}$: check that the current term has head symbol f. If this is the case, replace the top of the stack (i.e. the current term) by its arguments (first argument on top) and continue with T. Otherwise, continue with U (without changing the stack).

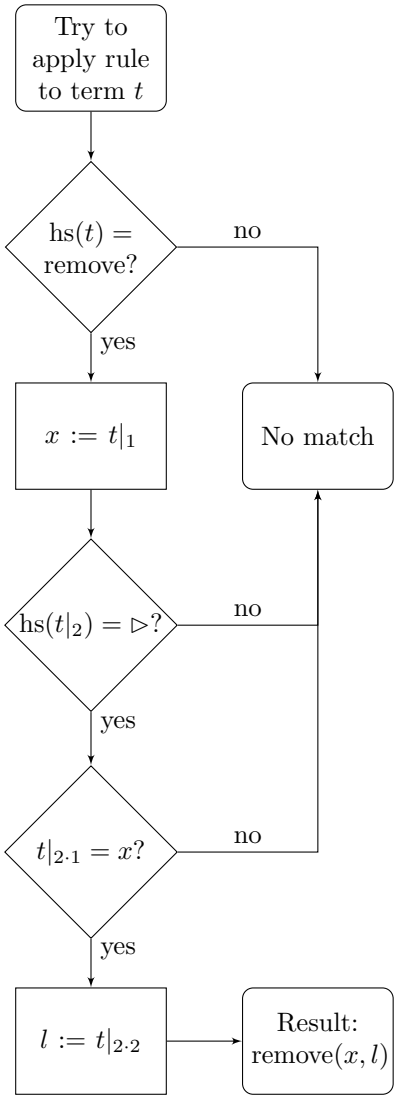


Figure 3.1: Process of trying to apply $\mathrm{remove}(x, x \triangleright l) \to \mathrm{remove}(x, l)$ to a term

- $\mathsf{S}^1(x, T)$, with $x \in \mathbb{V}$ and $T \in \mathbb{M}$: assign the current term to variable x, remove it from the stack and continue with T.

- $\mathsf{M}^1(x, T, U)$, with $x \in \mathbb{V}$ and $T, U \in \mathbb{M}$: check that the current term is the same as the value (previously) assigned to variable x. If this is the case,

remove it from the stack and continue with T. Otherwise, continue with U (without changing the stack).

- $\mathsf{R}(t)$, with $t \in \mathbb{T}$: return term t as the result of a successful application.
- X: no match.

Note that at this moment the third arguments of F and M^1 are not really necessary as they will always be X when matching a single rule. This will change when we look at trees combining multiple rewrite rules.

This gives us the following initial definition of μ. Note that the result is a set per the above specification. In this definition we use auxiliary function $\mu' : \mathbb{M} \times \mathbb{S}(\mathbb{T}) \times (\mathbb{V} \to \mathbb{T}) \to \mathcal{P}(\mathbb{T})$ that takes a match tree, a stack of terms and a substitution (as explained above) and "executes" the match tree.

Definition μ. Let τ be an arbitrary (but fixed) substitution (i.e. its value is not relevant). The function μ is defined as follows.

$$
\begin{array}{lcll}
\mu(T, t) & = & \mu'(T, [t], \tau) & \\
 & & & \\
\mu'(\mathsf{F}(f, T, U), [], \sigma) & = & \emptyset & \\
\mu'(\mathsf{F}(f, T, U), x \rhd s, \sigma) & = & \mu'(U, x \rhd s, \sigma) & \\
\mu'(\mathsf{F}(f, T, U), f(t_1, \ldots, t_n) \rhd s, \sigma) & = & \mu'(T, t_1 \rhd \ldots \rhd t_n \rhd s, \sigma) & \\
\mu'(\mathsf{F}(f, T, U), g(t_1, \ldots, t_n) \rhd s, \sigma) & = & \mu'(U, g(t_1, \ldots, t_n) \rhd s, \sigma) & \text{if } f \neq g \\
\mu'(\mathsf{S}^1(x, T), [], \sigma) & = & \emptyset & \\
\mu'(\mathsf{S}^1(x, T), t \rhd s, \sigma) & = & \mu'(T, s, \sigma[x \mapsto t]) & \\
\mu'(\mathsf{M}^1(x, T, U), [], \sigma) & = & \emptyset & \\
\mu'(\mathsf{M}^1(x, T, U), t \rhd s, \sigma) & = & \mu'(T, s, \sigma) & \text{if } \sigma(x) = t \\
\mu'(\mathsf{M}^1(x, T, U), t \rhd s, \sigma) & = & \mu'(U, t \rhd s, \sigma) & \text{if } \sigma(x) \neq t \\
\mu'(\mathsf{R}(t), [], \sigma) & = & \{t\sigma\} & \\
\mu'(\mathsf{R}(t), t \rhd s, \sigma) & = & \emptyset & \\
\mu'(\mathsf{X}, s, \sigma) & = & \emptyset &
\end{array}
$$

The construction of match trees from rewrite rules is done by first constructing the trees for each rule separately and then combining these trees. Constructing a match tree for a single rule is quite straightforward and is done with function $\gamma_1 : \mathbb{R} \to \mathbb{M}$ which is defined as follows. Note that, similar to the definition of μ above, we use an auxiliary function $\gamma_1' : \mathbb{S}(\mathbb{T}) \times \mathbb{T} \times \mathcal{P}(\mathbb{V}) \to \mathbb{M}$. Its first argument is a stack of patterns (like the stack of terms above) and its second argument the right-hand side of the original rewrite rule. As third argument it takes a set of variables indicating the variables to which a value will have been assigned when matching with this tree.

Definition γ_1**.** The function γ_1 is defined as follows.

$$
\begin{array}{lcll}
\gamma_1(l \to r) & = & \gamma_1'([l], r, \emptyset) & \\
 & & & \\
\gamma_1'([], r, V) & = & \mathsf{R}(r) & \\
\gamma_1'(x \rhd s, r, V) & = & \mathsf{S}^1(x, \gamma_1'(s, r, V \cup \{x\})) & \text{if } x \notin V \\
\gamma_1'(x \rhd s, r, V) & = & \mathsf{M}^1(x, \gamma_1'(s, r, V), \mathsf{X}) & \text{if } x \in V \\
\gamma_1'(f(p_1, \ldots, p_n) \rhd s, r, V) & = & \mathsf{F}(f, \gamma_1'(p_1 \rhd \ldots \rhd p_n \rhd s, r, V), \mathsf{X}) &
\end{array}
$$

To illustrate the use of γ_1 and μ we look at the following examples.

Example 3.2.1 We consider the rewrite rule $\text{remove}(x, x \rhd l) \to \text{remove}(x, l)$ once more. With γ_1 we get the following derivation to a match tree for this rule.

$$
\begin{array}{ll}
 & \gamma_1(\text{remove}(x, x \rhd l) \to \text{remove}(x, l)) \\
= & \\
 & \gamma_1'([\text{remove}(x, x \rhd l)], \text{remove}(x, l), \emptyset) \\
= & \\
 & \mathsf{F}(\text{remove}, \gamma_1'([x, x \rhd l], \text{remove}(x, l), \emptyset), \mathsf{X}) \\
= & \\
 & \mathsf{F}(\text{remove}, \mathsf{S}^1(x, \gamma_1'([x \rhd l], \text{remove}(x, l), \{x\}), \mathsf{X}) \\
= & \\
 & \mathsf{F}(\text{remove}, \mathsf{S}^1(x, \mathsf{F}(\rhd, \gamma_1'([x, l], \text{remove}(x, l), \{x\}), \mathsf{X})), \mathsf{X}) \\
= & \\
 & \mathsf{F}(\text{remove}, \mathsf{S}^1(x, \mathsf{F}(\rhd, \mathsf{M}^1(x, \gamma_1'([l], \text{remove}(x, l), \{x\}), \mathsf{X}), \mathsf{X})), \mathsf{X}) \\
= & \\
 & \mathsf{F}(\text{remove}, \mathsf{S}^1(x, \mathsf{F}(\rhd, \mathsf{M}^1(x, \mathsf{S}^1(l, \gamma_1'([], \text{remove}(x, l), \{x, l\})), \mathsf{X}), \mathsf{X})), \mathsf{X}) \\
= & \\
 & \mathsf{F}(\text{remove}, \mathsf{S}^1(x, \mathsf{F}(\rhd, \mathsf{M}^1(x, \mathsf{S}^1(l, \mathsf{R}(\text{remove}(x, l))), \mathsf{X}), \mathsf{X})), \mathsf{X})
\end{array}
$$

This match tree is depicted in Fig. 3.2. Note the resemblance with Fig. 3.1.

Example 3.2.2 Taking the match tree from the previous example, we consider applying rule $\text{remove}(x, x \rhd l) \to \text{remove}(x, l)$ to the terms $\text{remove}(t, [])$ and $\text{remove}(t, [t])$. First $\text{remove}(t, [])$:

$$
\begin{array}{ll}
 & \mu(\mathsf{F}(\text{remove}, \mathsf{S}^1(x, \mathsf{F}(\rhd, \mathsf{M}^1(x, \mathsf{S}^1(l, \mathsf{R}(\text{remove}(x, l))), \mathsf{X}), \mathsf{X})), \mathsf{X}), \\
 & \quad \text{remove}(t, [])) \\
= &
\end{array}
$$

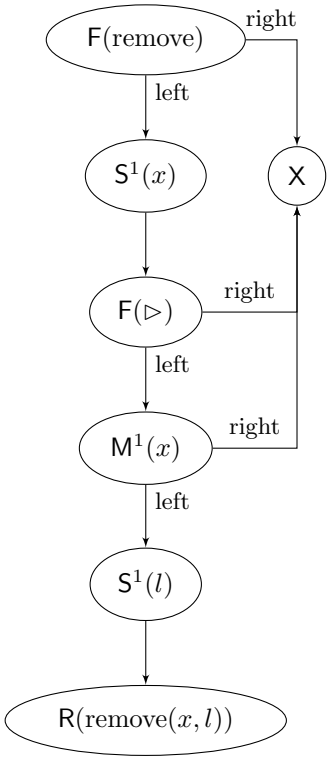


Figure 3.2: Match tree for rule $\text{remove}(x, x \rhd l) \to \text{remove}(x, l)$

$$
\begin{aligned}
& \mu'(\mathsf{F}(\text{remove}, \mathsf{S}^1(x, \mathsf{F}(\rhd, \mathsf{M}^1(x, \mathsf{S}^1(l, \mathsf{R}(\text{remove}(x, l))), \mathsf{X}), \mathsf{X})), \mathsf{X}), \\
& \quad [\text{remove}(t, [])], \tau) \\
= \\
& \mu'(\mathsf{S}^1(x, \mathsf{F}(\rhd, \mathsf{M}^1(x, \mathsf{S}^1(l, \mathsf{R}(\text{remove}(x, l))), \mathsf{X}), \mathsf{X})), [t, []], \tau) \\
= \\
& \mu'(\mathsf{F}(\rhd, \mathsf{M}^1(x, \mathsf{S}^1(l, \mathsf{R}(\text{remove}(x, l))), \mathsf{X}), \mathsf{X}), [[]], \tau[x \mapsto t]) \\
= \\
& \mu'(\mathsf{X}, [[]], \tau[x \mapsto t]) \\
= \\
& \emptyset
\end{aligned}
$$

Next remove$(t, [t])$:

$$\begin{aligned}
&\mu(\mathsf{F}(\text{remove}, \mathsf{S}^1(x, \mathsf{F}(\triangleright, \mathsf{M}^1(x, \mathsf{S}^1(l, \mathsf{R}(\text{remove}(x, l))), \mathsf{X}), \mathsf{X})), \mathsf{X}),\\
&\quad \text{remove}(t, [t]))\\
=\\
&\mu'(\mathsf{F}(\text{remove}, \mathsf{S}^1(x, \mathsf{F}(\triangleright, \mathsf{M}^1(x, \mathsf{S}^1(l, \mathsf{R}(\text{remove}(x, l))), \mathsf{X}), \mathsf{X})), \mathsf{X}),\\
&\quad [\text{remove}(t, [t])], \tau)\\
=\\
&\mu'(\mathsf{S}^1(x, \mathsf{F}(\triangleright, \mathsf{M}^1(x, \mathsf{S}^1(l, \mathsf{R}(\text{remove}(x, l))), \mathsf{X}), \mathsf{X})), [t, [t]], \tau)\\
=\\
&\mu'(\mathsf{F}(\triangleright, \mathsf{M}^1(x, \mathsf{S}^1(l, \mathsf{R}(\text{remove}(x, l))), \mathsf{X}), \mathsf{X}), [[t]], \tau[x \mapsto t])\\
=\\
&\mu'(\mathsf{M}^1(x, \mathsf{S}^1(l, \mathsf{R}(\text{remove}(x, l))), \mathsf{X}), [t, []], \tau[x \mapsto t])\\
=\\
&\mu'(\mathsf{S}^1(l, \mathsf{R}(\text{remove}(x, l))), [[]], \tau[x \mapsto t])\\
=\\
&\mu'(\mathsf{R}(\text{remove}(x, l)), [], \tau[x \mapsto t][l \mapsto []])\\
=\\
&\{\text{remove}(x, l)(\tau[x \mapsto t][l \mapsto []])\}\\
=\\
&\{\text{remove}(t, [])\}
\end{aligned}$$

Of course, we need to be sure that these definitions really make sense.

Theorem 3.2.3

$$\mu(\gamma_1(l \to r), t) = \{r\sigma \ : \ t = l\sigma\}$$

Combining Match Trees

The next thing we need to do is combine match trees. We do this with the $\|$ operator. The goal of this operator, as expressed in the following specification, is to construct a single match tree that can be used to simultaneously match a term with both argument match trees. That is, if you consider a match tree as a representation of a set of rewrite rules, the $\|$ operator on match trees corresponds to the union on sets of rewrite rules.

Specification $\|$.

$$\| : \mathbb{M} \times \mathbb{M} \to \mathbb{M}$$

$$\mu(T \,\|\, U, t) = \mu(T, t) \cup \mu(U, t)$$

With this operator we can define γ as follows.

Definition γ. Let ι be fixed choice function on sets of rewrite rules.

$$\begin{array}{lcl} \gamma(\emptyset) & = & \mathsf{X} \\ \gamma(R) & = & \gamma(R \setminus \{\iota(R)\}) \parallel \gamma_1(\rho) \end{array}$$

Corollary 3.2.4 *The above definition of γ satisfies its specification.*

Because we are now considering the case that a match tree represents more than one rewrite rule, we must extend our match trees with some additional constructs. First of all we now have to facilitate the situation where multiple rules can be applied. We therefore extend R to also take sets of terms as its argument. Thus, $\mathsf{R}(R)$ means that each $r \in R$ is a result of a possible application of a rewrite rule.

Also, because different rules might need to do different things for a certain argument, we can no longer remove the top of the stack as soon as we have executed an S^1 or M^1. We therefore introduce S and M that only differ from S^1 and M^1 in that they leave the stack unaltered. Because S and M do not remove the top of the stack, we also need an extra construct N that does precisely this.

We then get the following *extension* of the definition of μ.

Extension μ. The extension of μ with S, M, N and R (on sets) is as follows.

$$\begin{array}{lcll} \mu'(\mathsf{S}(x,T),[],\sigma) & = & \emptyset & \\ \mu'(\mathsf{S}(x,T),t \rhd s,\sigma) & = & \mu'(T,t \rhd s,\sigma[x \mapsto t]) & \\ \mu'(\mathsf{M}(x,T,U),[],\sigma) & = & \emptyset & \\ \mu'(\mathsf{M}(x,T,U),t \rhd s,\sigma) & = & \mu'(T,t \rhd s,\sigma) & \text{if } \sigma(x) = t \\ \mu'(\mathsf{M}(x,T,U),t \rhd s,\sigma) & = & \mu'(U,t \rhd s,\sigma) & \text{if } \sigma(x) \neq t \\ \mu'(\mathsf{N}(T),[],\sigma) & = & \emptyset & \\ \mu'(\mathsf{N}(T),t \rhd s,\sigma) & = & \mu'(T,s,\sigma) & \\ \mu'(\mathsf{R}(R),[],\sigma) & = & \{t\sigma \ : \ t \in R\} & \\ \mu'(\mathsf{R}(R),t \rhd s,\sigma) & = & \emptyset & \end{array}$$

It is clear that S^1, M^1 and R (on a single term) are strongly related to these new elements. To express this we first need to define equality on match trees such that we can replace subtrees with equivalent trees. Note that this definition uses μ' instead of μ as we want equivalent trees to be equivalent regardless of the context.

Definition $=_\mu$.

$$T =_\mu U \ \Leftrightarrow \ \forall_{s,\sigma}(\mu'(T,s,\sigma) = \mu'(U,s,\sigma))$$

With this equality we can now give the following properties that allow us to remove every occurrence of the elements S^1, M^1 or R (on a single term) if desired (e.g. in

implementations or proofs).

Property 3.2.5

(S^{1}S)	$\mathsf{S}^1(x,T)$	$=_\mu$	$\mathsf{S}(x,\mathsf{N}(T))$
(M^{1}M)	$\mathsf{M}^1(x,T,U)$	$=_\mu$	$\mathsf{M}(x,\mathsf{N}(T),U)$
(rR)	$\mathsf{R}(t)$	$=_\mu$	$\mathsf{R}(\{t\})$

When combining match trees there are several restrictions on the trees we consider as arguments of $\|$. Instead of considering arbitrary match trees as argument we restrict one of the arguments to the structures that are the result of γ_1 (modulo $=_\mu$). This mainly means that trees are not allowed to have nodes with three arguments (M, F) of which the third argument is not X. We refer to the set of all such trees as $\mathbb{M}_1$. This obviously is sufficient for our needs (given the definition of γ).

The reason we do this is because some cases are needlessly complex and will not occur in practice. For example consider the combination of two F-trees (with $f \neq g$):

$$\mathsf{F}(f,T,U) \parallel \mathsf{F}(g,T',U')$$

Let us assume the head symbol of the current term is an f. This means that the first tree wants to remove the current term and put its direct subterms on the stack. However, the second tree wants no such thing; it wants to continue with the current term and tree U'. In addition, it might be possible that, for example, $T' = \mathsf{F}(f,T'',U'')$, which means we cannot just take a F of f first and within it combine $\mathsf{F}(g,T',U')$ with T and U. This would make T'' and/or U'' unreachable as, being a subtree of another F of f we already know whether f is the head symbol.

It is possible to solve this, but this is needlessly complicated for the current setting. Here we have that if a $\mathsf{F}(g,T',U')$ is (part of) the result of γ_1, then $U' = \mathsf{X}$. We use this information to our advantage.

Also, we assume that the sets of variables used in both arguments of $\|$ are disjoint. Of course, it is straightforward to establish this as equality of match trees is preserved by α-conversion. (Note that S acts as a binder in match trees.)

Note that we only define $\|$ for the relevant cases. Situations such a $\mathsf{R}(R)\|\mathsf{S}(x,T)$ should not occur. Also, we ensure that a certain "local" order is maintained to facilitate the work in Section 3.4. This means that M comes before S, S before F.

Definition $\|$. Assuming that $\mathrm{var}(T) \cap \mathrm{var}(U) = \emptyset$ for all $T \parallel U$, the definition of $\|$ is as follows.

$$\begin{array}{lcl}
\mathsf{X} \parallel T & = & T \\
T \parallel \mathsf{X} & = & T \\
\mathsf{R}(R) \parallel \mathsf{R}(R') & = & \mathsf{R}(R \cup R') \\
\mathsf{S}(x,T) \parallel \mathsf{S}(y,U) & = & \mathsf{S}(x, T \parallel \mathsf{S}(y,U))
\end{array}$$

(continued on next page)

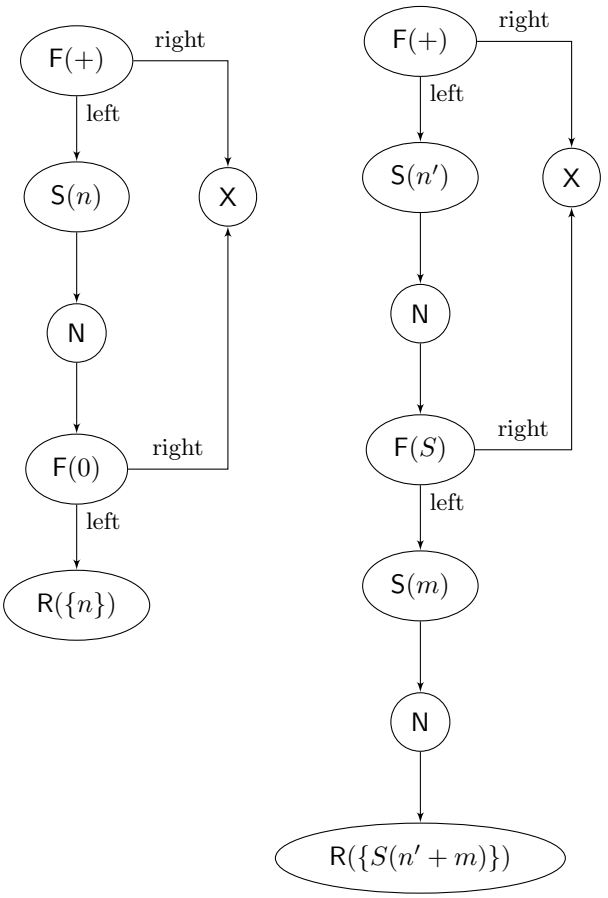
$$
\begin{array}{lcll}
\mathsf{S}(x,T) \parallel \mathsf{M}(y,U,\mathsf{X}) & = & \mathsf{M}(y,\mathsf{S}(x,T) \parallel U,\mathsf{S}(x,T)) & \\
\mathsf{S}(x,T) \parallel \mathsf{F}(f,U,\mathsf{X}) & = & \mathsf{S}(x,T \parallel \mathsf{F}(f,U,\mathsf{X})) & \\
\mathsf{S}(x,T) \parallel \mathsf{N}(U) & = & \mathsf{S}(x,T \parallel \mathsf{N}(U)) & \\
\mathsf{M}(x,T,T') \parallel \mathsf{S}(y,U) & = & \mathsf{M}(x,T \parallel \mathsf{S}(y,U),T' \parallel \mathsf{S}(y,U)) & \\
\mathsf{M}(x,T,T') \parallel \mathsf{M}(y,U,\mathsf{X}) & = & \mathsf{M}(x,T \parallel \mathsf{M}(y,U,\mathsf{X}),T' \parallel \mathsf{M}(y,U,\mathsf{X})) & \\
\mathsf{M}(x,T,T') \parallel \mathsf{F}(f,U,\mathsf{X}) & = & \mathsf{M}(x,T \parallel \mathsf{F}(f,U,\mathsf{X}),T' \parallel \mathsf{F}(f,U,\mathsf{X})) & \\
\mathsf{M}(x,T,T') \parallel \mathsf{N}(U) & = & \mathsf{M}(x,T \parallel \mathsf{N}(U),T' \parallel \mathsf{N}(U)) & \\
\mathsf{F}(f,T,T') \parallel \mathsf{S}(x,U) & = & \mathsf{S}(x,F(f,T,T') \parallel U) & \\
\mathsf{F}(f,T,T') \parallel \mathsf{M}(x,U,\mathsf{X}) & = & \mathsf{M}(x,F(f,T,T') \parallel U,F(f,T,T')) & \\
\mathsf{F}(f,T,T') \parallel \mathsf{F}(f,U,\mathsf{X}) & = & \mathsf{F}(f,T \parallel U,T') & \\
\mathsf{F}(f,T,T') \parallel \mathsf{F}(g,U,\mathsf{X}) & = & \mathsf{F}(f,T,T' \parallel F(g,U,\mathsf{X})) & \text{if } f \neq g \\
\mathsf{F}(f,T,T') \parallel \mathsf{N}(U) & = & \mathsf{F}(f,T \parallel \mathsf{N}^{\mathrm{ar}(f)}(U),T' \parallel \mathsf{N}(U)) & \\
\mathsf{N}(T) \parallel \mathsf{S}(x,U) & = & \mathsf{S}(x,\mathsf{N}(T) \parallel U) & \\
\mathsf{N}(T) \parallel \mathsf{M}(x,U,U') & = & \mathsf{M}(x,\mathsf{N}(T) \parallel U,\mathsf{N}(T) \parallel U') & \\
\mathsf{N}(T) \parallel \mathsf{F}(f,U,\mathsf{X}) & = & \mathsf{F}(f,\mathsf{N}^{\mathrm{ar}(f)}(T) \parallel U,\mathsf{N}(T)) & \\
\mathsf{N}(T) \parallel \mathsf{N}(U) & = & \mathsf{N}(T \parallel U) &
\end{array}
$$

Example 3.2.6 Lets look at the rewrite rules for the $+$. We recall the following rules.

$$
\begin{array}{lcll}
\rho_1 & n+0 & \to & n \\
\rho_2 & n+S(m) & \to & S(n+m)
\end{array}
$$

The match trees corresponding to $\gamma^1(\rho_1)$ and $\gamma^1(\rho_2)$ are $\mathsf{F}(+,\mathsf{S}^1(n,\mathsf{F}(0,\mathsf{R}(n),\mathsf{X})),\mathsf{X})$ and $\mathsf{F}(+,\mathsf{S}^1(n,\mathsf{F}(S,\mathsf{S}^1(m,\mathsf{R}(S(n{+}m))),\mathsf{X})),\mathsf{X})$, respectively. With the above properties (and α-equivalence) we have that these match trees are equivalent to the match trees $\mathsf{F}(+,\mathsf{S}(n,\mathsf{N}(\mathsf{F}(0,\mathsf{R}(\{n\}),\mathsf{X}))),\mathsf{X})$ (as depicted in Fig. 3.3a) respectively $\mathsf{F}(+,\mathsf{S}(n',\mathsf{N}(\mathsf{F}(S,\mathsf{S}(m,\mathsf{N}(\mathsf{R}(\{S(n'+m)\}))),\mathsf{X}))),\mathsf{X})$ (as depicted in Fig. 3.3b). We combine these match trees to get a single match tree for both rules.

$$
\begin{array}{l}
\quad \mathsf{F}(+,\mathsf{S}(n,\mathsf{N}(\mathsf{F}(0,\mathsf{R}(\{n\}),\mathsf{X}))),\mathsf{X}) \parallel \\
\quad \mathsf{F}(+,\mathsf{S}(n',\mathsf{N}(\mathsf{F}(S,\mathsf{S}(m,\mathsf{N}(\mathsf{R}(\{S(n'+m)\}))),\mathsf{X}))),\mathsf{X}) \\
= \\
\quad \mathsf{F}(+, \\
\qquad \mathsf{S}(n,\mathsf{N}(\mathsf{F}(0,\mathsf{R}(\{n\}),\mathsf{X}))) \parallel \\
\qquad \mathsf{S}(n',\mathsf{N}(\mathsf{F}(S,\mathsf{S}(m,\mathsf{N}(\mathsf{R}(\{S(n'+m)\}))),\mathsf{X}))), \\
\quad \mathsf{X}) \\
=
\end{array}
$$



(a) $\gamma^1(\rho_1)$ (b) $\gamma^1(\rho_2)$

Figure 3.3: Match trees for ρ_1 and ρ_2

$$
\begin{aligned}
&\mathsf{F}(+, \mathsf{S}(n, \mathsf{S}(n', \\
&\quad \mathsf{N}(\mathsf{F}(0, \mathsf{R}(\{n\}), \mathsf{X})) \parallel \\
&\quad \mathsf{N}(\mathsf{F}(S, \mathsf{S}(m, \mathsf{N}(\mathsf{R}(\{S(n' + m)\}))), \mathsf{X})) \\
&\quad)), \mathsf{X}) \\
=
\end{aligned}
$$

$$
\begin{array}{l}
\quad \mathsf{F}(+, \mathsf{S}(n, \mathsf{S}(n', \mathsf{N}(\\
\qquad \mathsf{F}(0, \mathsf{R}(\{n\}), \mathsf{X}) \parallel \\
\qquad \mathsf{F}(S, \mathsf{S}(m, \mathsf{N}(\mathsf{R}(\{S(n'+m)\}))), \mathsf{X}) \\
\quad))), \mathsf{X}) \\
= \\
\quad \mathsf{F}(+, \mathsf{S}(n, \mathsf{S}(n', \mathsf{N}(\\
\qquad \mathsf{F}(0, \mathsf{R}(\{n\}), \\
\qquad\quad \mathsf{X} \parallel \\
\qquad\quad \mathsf{F}(S, \mathsf{S}(m, \mathsf{N}(\mathsf{R}(\{S(n'+m)\}))), \mathsf{X}) \\
\qquad) \\
\quad))), \mathsf{X}) \\
= \\
\quad \mathsf{F}(+, \mathsf{S}(n, \mathsf{S}(n', \mathsf{N}(\mathsf{F}(0, \mathsf{R}(n), \mathsf{F}(S, \mathsf{S}(m, \mathsf{N}(\mathsf{R}(\{S(n'+m)\}))), \mathsf{X}))))), \mathsf{X})
\end{array}
$$

This tree is depicted in Fig. 3.4

Note that the result of $\parallel$ is not optimal. The variables n and n' are always assigned the same value. In Section 3.4 we discuss optimising match trees.

Again, we must make sure the $\parallel$ function is actually sound with respect to μ.

Theorem 3.2.7

$$\forall_{T \in \mathbb{M}, U \in \mathbb{M}_1, t \in \mathbb{T}} (\mu(T \parallel U, t) = \mu(T, t) \cup \mu(U, t))$$

Note that in Appendix C.4 we actually prove this theorem including the extension that follow (in particular the one in Section 3.3.2). It is easy to check that with these extensions all possible combinations are taken care and preserve the fact that the right-hand side of $\parallel$ should be in $\mathbb{M}_1$. This means that $\parallel$ is well-defined over the domain $\mathbb{M} \times \mathbb{M}_1$.

Also note that this means that we do not really know that combining two match trees with the constructors introduced so far results in a match tree that also only consists of those nodes. We would be surprised if this is not the case, but we have not investigated this at this time.

3.3 Extensions

In this section we consider several extensions to match trees. First we make an extension for conditional rewrite rules, then we look at matching applicative terms and finally we consider priorities on rewrite rules.

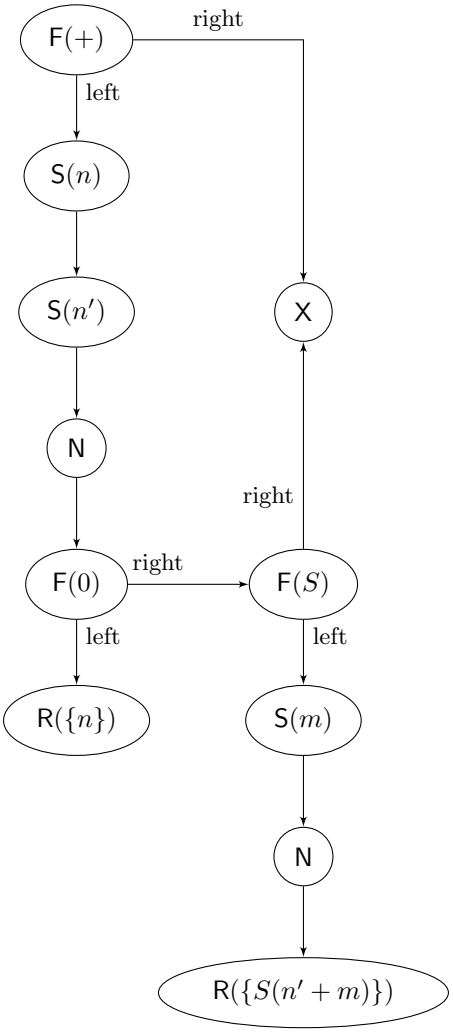


Figure 3.4: Match tree for $\{\rho_1, \rho_2\}$

3.3.1 Conditional Rewrite Rules

To also allow conditional rewrite rules we extend the specification of μ and γ as follows. Note that we assume the condition evaluation function η is given.

Specification μ, γ.

$\mu(\gamma(R), t) = \{r\sigma \ : \ l \to r \in R \wedge t = l\sigma\} \cup \{r\sigma \ : \ l \to r \ \textbf{if} \ c \in R \wedge t = l\sigma \wedge \eta(c\sigma)\}$

To reflect this addition in the match trees we need a way to establish whether $\eta(t)$ holds for terms t. We do this by introducing a construct $\mathsf{C}(t, T, U)$ that is defined as follows.

Extension μ.

$$\begin{array}{lcll} \mu'(\mathsf{C}(t,T,U),s,\sigma) & = & \mu'(T,s,\sigma) & \text{if } \eta(t\sigma) \\ \mu'(\mathsf{C}(t,T,U),s,\sigma) & = & \mu'(U,s,\sigma) & \text{if } \neg\,\eta(t\sigma) \end{array}$$

The extension of γ_1 is also quite straightforward. We only need to add a parameter to the auxiliary function γ_1' for the condition of a rewrite rule and add a C before returning a result.

Extension γ_1. The extension of γ_1 with conditional rewrite rules is as follows.

$$\begin{array}{lcll} \gamma_1(l \to r \textbf{ if } c) & = & \gamma_1'([l], r, c, \emptyset) & \\ & & & \\ \gamma_1'([], r, c, V) & = & \mathsf{C}(c, \mathsf{R}(r), \mathsf{X}) & \\ \gamma_1'(x \rhd s, r, c, V) & = & \mathsf{S}^1(x, \gamma_1'(s, r, c, V \cup \{x\})) & \text{if } x \notin V \\ \gamma_1'(x \rhd s, r, c, V) & = & \mathsf{M}^1(x, \gamma_1'(s, r, c, V), \mathsf{X}) & \text{if } x \in V \\ \gamma_1'(f(p_1, \ldots, p_n) \rhd s, r, c, V) & = & \mathsf{F}(f, \gamma_1'(p_1 \rhd \ldots \rhd p_n \rhd s, r, c, V), \mathsf{X}) & \end{array}$$

To show that this extension is sound we have the following theorem.

Theorem 3.3.1

$$\mu(\gamma_1(l \to r \textbf{ if } c), t) = \{r\sigma \;:\; t = l\sigma \;\wedge\; \eta(c\sigma)\}$$

Finally we must extend $\|$ as well. This means we must also reprove Theorem 3.2.7.

Extension $\|$. Assuming that $\mathrm{var}(T) \cap \mathrm{var}(U) = \emptyset$ for all $T \,\|\, U$, the definition of $\|$ is as follows.

$$\begin{array}{lcll} \mathsf{C}(t,T,T') \,\|\, U & = & \mathsf{C}(t, T \,\|\, U, T' \,\|\, U) & \\ T \,\|\, \mathsf{C}(t,U,\mathsf{X}) & = & \mathsf{C}(t, T \,\|\, U, T) & \text{if } \neg\exists_{u,T',T''}(T = \mathsf{C}(u,T',T'')) \end{array}$$

3.3.2 Adding Applicative Terms

In higher order settings one typically allows *applicative* terms: function symbols are regarded as terms and an application operator allows terms to be applied to

terms. A term $f(x, y)$ in this setting is actually the term $f(x)(y)$. We keep using the notation $f(x, y)$ though. The only difference with before is that the number of arguments that is applied to a function symbol can be less than its arity.

Note that with applicative terms it is allowed to write $x(t)$, where x is a variable. For sake of simplicity we do not allow such terms in the left-hand side of rewrite rules. We have not found this to be a limitation in practice.

To keep the notion of arity sensible we assume that it is possible to give a non-recursive type system for our applicative terms such that the left-hand and right-hand sides of each rewrite rules have the same type. This effectively means that it is not possible to have a rewrite rule such as $g(f(x, y)) \to g(f(x))$.

Due to the fact that the head symbol of a left-hand side of a rewrite rule might have a greater arity then its number of actual arguments, we have a slight difference in the notion of matching. Before we said a pattern p matches a term t if there is a substitution σ such that $t = p\sigma$. However, with applicative terms it might be that case that pattern p has a different number of arguments than term t. For example, with rewrite rule $f \to g$ we wish that term $f(t)$ rewrites to $g(t)$. We therefore extend the notion of matching as follows. A pattern p matches a term t if, and only if, there is a sequence of variables $x_1, \ldots, x_n$ and a substitution σ such that $t = p(x_1, \ldots, x_n)\sigma$.

Finally, we must consider how to represent the change in terms in the match trees. Recall that in the definition of $\|$ we have an equation $\mathsf{F}(f, T, T') \parallel \mathsf{N}(U) = \mathsf{F}(f, T \parallel \mathsf{N}^{\mathrm{ar}(f)}(U), T' \parallel \mathsf{N}(U))$. Of course, this no longer suffices because the arity of a function symbol no longer corresponds to the number of arguments such a function has in a term. One option is to extend F to include a number indicating the expected number of arguments. However, for sake of simplicity, we choose another solution. For each F node that is the result of γ'_1 we can easy get the expected number of arguments by looking at the head of the first arguments of γ'_1. So we can trivially annotate such nodes with this number and use it when combining trees. Due to our typing assumption we know that it is not possible that we need $\mathsf{F}(f, T, U) \parallel \mathsf{F}(f, T', U')$ where both F nodes are annotated with different numbers. That is, except for the very first node of γ'_1, but there we never encounter $\mathsf{F}(f, T, U) \parallel \mathsf{N}(T')$ as we do not allow rules of the form $x \to t$. Note that we do not explicitly write down the annotations in the examples.

One has to adapt the μ function slightly such that an annotated function symbol will match with an unannotated function symbol. We do not explicitly do this here. Also, we will assume in the rest of this chapter that sets of rewrite rules have been adapted to include annotations.

With all this we get the following specification for μ_{a} and γ.

Specification μ_{a}, γ.

$$\mu_{\mathrm{a}} : \mathbb{M} \times \mathbb{T} \to \mathcal{P}(\mathbb{T})$$

$$\mu_{\mathrm{a}}(\gamma(R), t) = \{r(x_1, \ldots, x_n)\sigma \ : \ l \to r \in R \ \wedge \ t = l(x_1, \ldots, x_n)\sigma\}$$

We can now derive the following.

$$
\begin{aligned}
& \mu_{\mathrm{a}}(\gamma(R), t) \\
= \quad & \{ \text{ Specification } \mu_{\mathrm{a}}, \gamma \ \} \\
& \{r(x_1, \ldots, x_n)\sigma \ : \ l \to r \in R \ \wedge \ t = l(x_1, \ldots, x_n)\sigma\} \\
= \quad & \{ \text{ Extend } R \ \} \\
& \{r(x_1, \ldots, x_n)\sigma \ : \ l(x_1, \ldots, x_n) \to r(x_1, \ldots, x_n) \in \\
& \qquad \{l(y_1, \ldots, y_n) \to r(y_1, \ldots, y_n) \ : \ l \to r \in R\} \ \wedge \ t = l(x_1, \ldots, x_n)\sigma\} \\
= \quad & \{ \text{ Calculus } \} \\
& \{r'\sigma \ : \ l' \to r' \in \{l(y_1, \ldots, y_n) \to r(y_1, \ldots, y_n) \ : \ l \to r \in R\} \ \wedge \ t = l'\sigma\} \\
= \quad & \{ \text{ Specification } \mu, \gamma \ \} \\
& \mu(\gamma(\{l(y_1, \ldots, y_n) \to r(y_1, \ldots, y_n) \ : \ l \to r \in R\}), t)
\end{aligned}
$$

In other words, we don't have to change much about our match trees to be able to match applicative terms; extending the input is sufficient for the match function μ. The only thing that we must take care of is combining match trees that expect a different amount of arguments.

Note that in general the number of variables that need to be added is bound by the arity of the function symbol. That is, if you add more variables to a pattern than its head symbol can take as arguments, then the pattern will never match. Also note that the extension of the input with conditional rules follows the same lines.

Before we give the required extensions to match applicative terms, we look at the precise problem we are facing here. Consider the following rewrite rules:

$$
\begin{array}{lllll}
f_1: & f & \to & g \\
f_2: & f(x) & \to & g(x)
\end{array}
$$

We get the following match trees for these rules:

$$
\begin{array}{ll}
m_{f_1}: & \mathsf{F}(f, \mathsf{R}(g), \mathsf{X}) \\
m_{f_2}: & \mathsf{F}(f, \mathsf{S}(x, \mathsf{N}(\mathsf{R}(g(x)))), \mathsf{X})
\end{array}
$$

When calculating $m_{f_1} \| m_{f_2}$ one encounters the term $\mathsf{R}(g) \| \mathsf{S}(x, \mathsf{R}(g(x)))$. Currently, $\|$ is not defined on this case. The reason is that R expects the stack to be empty while S expects the stack to contain at least one item.

In previous settings this situation could not occur due to the fact that all trees matching a specific function would expect the same number of arguments. Now that this is no longer the case we must extend the match trees with another construct. We introduce E with two argument: one for the case that the stack is not empty and one for the case that the stack is empty. Note that an equally valid choice would be to extend the nodes we already had with an additional argument. We add E only because it has less impact on the work that has already been done

and we believe it to be conceptually nicer. This gives us the following extension to μ.

Extension μ.

$$\begin{array}{lcl} \mu'(\mathsf{E}(T,U),[],\sigma) & = & \mu'(U,[],\sigma) \\ \mu'(\mathsf{E}(T,U),t \rhd s,\sigma) & = & \mu'(T,t \rhd s,\sigma) \end{array}$$

We then get the following extension to $\|$.

Extension $\|$.

$$\begin{array}{lcl} \mathsf{R}(R) \parallel \mathsf{F}(f,T,\mathsf{X}) & = & \mathsf{E}(\mathsf{F}(f,T,\mathsf{X}),\mathsf{R}(R)) \\ \mathsf{R}(R) \parallel \mathsf{S}(x,T) & = & \mathsf{E}(\mathsf{S}(x,T),\mathsf{R}(R)) \\ \mathsf{R}(R) \parallel \mathsf{M}(x,T,\mathsf{X}) & = & \mathsf{E}(\mathsf{M}(x,T,\mathsf{X}),\mathsf{R}(R)) \\ \mathsf{R}(R) \parallel \mathsf{N}(T) & = & \mathsf{E}(\mathsf{N}(T),\mathsf{R}(R)) \\ \mathsf{F}(f,T,T') \parallel \mathsf{R}(R) & = & \mathsf{E}(\mathsf{F}(f,T,T'),\mathsf{R}(R)) \\ \mathsf{S}(x,T) \parallel \mathsf{R}(R) & = & \mathsf{E}(\mathsf{S}(x,T),\mathsf{R}(R)) \\ \mathsf{M}(x,T,T') \parallel \mathsf{R}(R) & = & \mathsf{E}(\mathsf{M}(x,T,T'),\mathsf{R}(R)) \\ \mathsf{N}(T) \parallel \mathsf{R}(R) & = & \mathsf{E}(\mathsf{N}(T),\mathsf{R}(R)) \\ \mathsf{E}(T,T') \parallel \mathsf{R}(R) & = & \mathsf{E}(T,T' \parallel \mathsf{R}(R)) \\ \mathsf{E}(T,T') \parallel \mathsf{F}(f,U,\mathsf{X}) & = & \mathsf{E}(T \parallel \mathsf{F}(f,U,\mathsf{X}),T') \\ \mathsf{E}(T,T') \parallel \mathsf{S}(x,U) & = & \mathsf{E}(T \parallel \mathsf{S}(x,U),T') \\ \mathsf{E}(T,T') \parallel \mathsf{M}(y,U,\mathsf{X}) & = & \mathsf{E}(T \parallel \mathsf{M}(y,U,\mathsf{X}),T') \\ \mathsf{E}(T,T') \parallel \mathsf{N}(U) & = & \mathsf{E}(T \parallel \mathsf{N}(U),T') \end{array}$$

By extending $\|$ we need to reprove Theorem 3.2.7.

Example 3.3.2 Consider the following specification of $+$ (where id is the identity function).

$$\begin{array}{lcl} +(0) & \to & id \\ S(n)+m & \to & S(n+m) \end{array}$$

This gives us the match trees $\mathsf{F}(+,\mathsf{F}(0,\mathsf{R}(\{\mathrm{id}\}),\mathsf{X}),\mathsf{X})$ for the first rule as it is, $\mathsf{F}(+,\mathsf{F}(0,\mathsf{S}(n',\mathsf{N}(\mathsf{R}(\{\mathrm{id}(n')\}))),\mathsf{X}),\mathsf{X})$ for the first rule with an extra argument and $\mathsf{F}(+,\mathsf{F}(S,\mathsf{S}(n,\mathsf{N}(\mathsf{S}(m,\mathsf{N}(\mathsf{R}(\{S(n+m)\}))))),\mathsf{X}),\mathsf{X})$ for the last rule. Combining them gives the tree as depicted in Fig. 3.5.

3.3.3 Adding Priority

In settings where only closed terms are rewritten, it is common to impose an order on rewrite rules. For example, one would have the following two rules for equality.

$$\begin{array}{llcl} \mathrm{eq}_1 & x=x & \to & \mathrm{true} \\ \mathrm{eq}_2 & x=y & \to & \mathrm{false} \end{array}$$

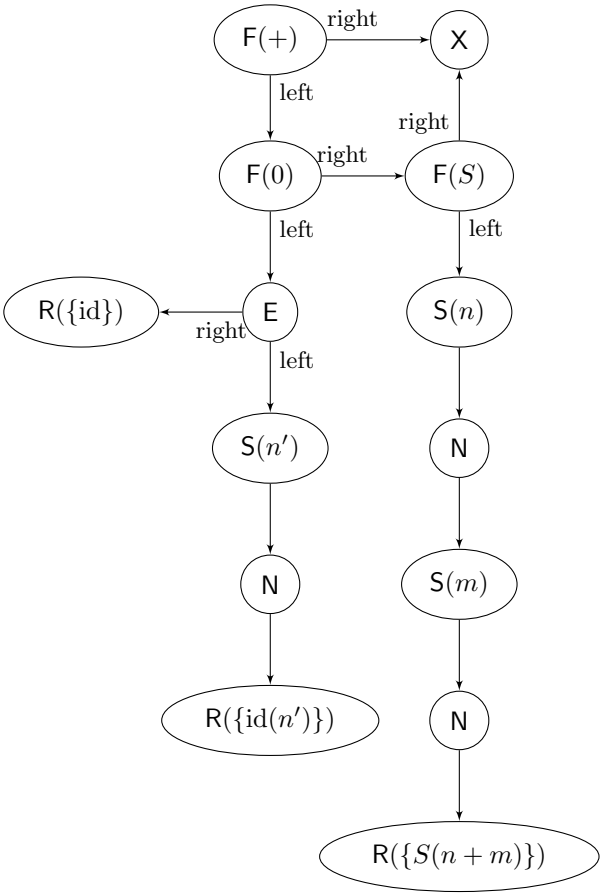


Figure 3.5: Match tree for applicative +

Without order, the second rule (eq_2) would lead to unexpected results as terms of the form $t = t$ fit the pattern $x = y$ as well as the pattern $x = x$. By imposing the order $\mathrm{eq}_2 < \mathrm{eq}_1$, one enforces the application of rule eq_1 for terms of the form $t = t$.

Priority is an effective way to reduce the number of rewrite rules and improve the efficiency of rewriting. Compare for example the rules above with rules without order; for a sort with n constructors one typically needs n^2 rules. With match trees we can combine these rules in a structure that allows $O(n)$ matching, but this is still a factor n bigger than with the two rules above. Even if n is small, repeatedly matching the same rules (as is often the case in rewriting) still means that this factor n has an significant effect on rewriting terms in general.

As mentioned before, in the setting of open terms rewriting the notion of priority

is not so useful. A partially instantiated term like $x = 0$ would be rewritten to false regardless of the possibility that x is latter (outside of rewriting) instantiated with 0. Of course, it is possible to circumvent this by adapting the matching and application process. We are not aware of such an implementation.

Note that rewriting with priority as described here is not quite the same as *priority rewriting* [BBK87]; priority rewriting prohibits the use of a rule if a greater rule can be applied on the top level (possibly by first rewriting subterms). The notion of priority we consider here is that only when multiple rules can be applied at the same position (in this thesis the top level), then we use the order to determine which rule is applied.

More formally (and generally) we have the following. Let φ be a function on sets of rewrite rules such that $\varphi(R) \subseteq R$ and $\varphi(R) = \emptyset$ if, and only if, $R = \emptyset$. This means that for every set of rewrite rules, φ defines the set of rules that have the highest priority. Then we have the following specification.

Specification μ_{p}.

$$\mu_{\mathrm{p}} : \mathbb{M} \times \mathbb{T} \to \mathcal{P}(\mathbb{T})$$
$$\overline{\gamma} : \mathcal{F}(\mathbb{R}) \to \mathbb{M}$$

$$\mu_{\mathrm{p}}(\overline{\gamma}(R), \varphi, t) = \{r\sigma \ : \ t = l\sigma \ \wedge \ l \to r \in \varphi(\{l' \to r' \in R \ : \ t = l'\sigma\})\}$$

We observe that this specification does not really require any significant changes to the match trees; only when returning a result it is necessary to only return those results that are from the application of a rewrite rule with the highest priority. In other words, if we can, in some way, apply φ to the R nodes of a match tree, then we do not need a specific matching function for these rewrite systems.

There is only one small problem: the R nodes only contain right-hand sides of rewrite rules and this is not sufficient to determine the right-hand sides of the same rules after φ is applied. For example, consider the node $\mathsf{R}(\{r, r'\})$ and assume that $\varphi(\{l_1 \to r, l' \to r'\}) = \{l_1 \to r\}$ and $\varphi(\{l_2 \to r, l' \to r'\}) = \{l' \to r'\}$. Depending on the origin of the r in the argument of R we either want $\mathsf{R}(\{r\})$ or $\mathsf{R}(\{r'\})$.

To overcome this problem we introduce a new node R' that takes a set of rewrite rules instead of just right-hand sides of rewrite rules. The extension of μ with this node is as follows:

Extension μ.

$$\begin{array}{lcl} \mu'(\mathsf{R}'(R), [], \sigma) & = & \{r\sigma \ : \ l \to r \in R\} \\ \mu'(\mathsf{R}'(R), t \rhd s, \sigma) & = & \emptyset \end{array}$$

Which trivially gives us the following property.

Property 3.3.3

$$\mathsf{R}'(R) =_{\mu} \mathsf{R}(\{r \ : \ l \to r \in R\})$$

We then define $\overline{\gamma}$, $\overline{\gamma}_1$ and $\overline{\gamma}'_1$ in the same way as γ, γ_1 and γ'_1 with the exception that the second of argument of γ'_1 is a rewrite rule, $\overline{\gamma}_1(l \to r) = \overline{\gamma}'_1([l], l \to r, \emptyset)$ and $\overline{\gamma}'_1([], \rho, V) = \mathsf{R}'(\{\rho\})$. Also we extend $\|$ by adding the same rules for R' as for R. Note that this means that we trivially have that $\overline{\gamma}(R) =_\mu \gamma(R)$.

What remains is a function prior to apply φ to match trees, which is defined as follows.

Definition prior**.**

$$\begin{array}{lcl}
\mathrm{prior}(\mathsf{X}, \varphi) & = & \mathsf{X} \\
\mathrm{prior}(\mathsf{F}(f, T, U), \varphi) & = & \mathsf{F}(f, \mathrm{prior}(T, \varphi), \mathrm{prior}(U, \varphi)) \\
\mathrm{prior}(\mathsf{S}(x, T), \varphi) & = & \mathsf{S}(x, \mathrm{prior}(T, \varphi)) \\
\mathrm{prior}(\mathsf{M}(x, T, U), \varphi) & = & \mathsf{M}(x, \mathrm{prior}(T, \varphi), \mathrm{prior}(U, \varphi)) \\
\mathrm{prior}(\mathsf{C}(t, T, U), \varphi) & = & \mathsf{C}(t, \mathrm{prior}(T, \varphi), \mathrm{prior}(U, \varphi)) \\
\mathrm{prior}(\mathsf{N}(T), \varphi) & = & \mathsf{N}(\mathrm{prior}(T, \varphi)) \\
\mathrm{prior}(\mathsf{R}'(R), \varphi) & = & \mathsf{R}(\{r \ : \ l \to r \in \varphi(R)\}) \\
\mathrm{prior}(\mathsf{E}(T, U), \varphi) & = & \mathsf{E}(\mathrm{prior}(T, \varphi), \mathrm{prior}(U, \varphi))
\end{array}$$

To show that this indeed does what we want, we have the following theorem.

Theorem 3.3.4

$$\mu_{\mathrm{p}}(\overline{\gamma}(R), \varphi, t) = \mu(\mathrm{prior}(\overline{\gamma}(R), \varphi), t)$$

In other words, adding priority with a function φ can be done by simply applying prior to the match tree and this φ.

Example 3.3.5 We look at the example of the equality. We recall the following rules:

$$\begin{array}{lcl}
\mathrm{eq}_1 \quad x = x & \to & \mathrm{true} \\
\mathrm{eq}_2 \quad x = y & \to & \mathrm{false}
\end{array}$$

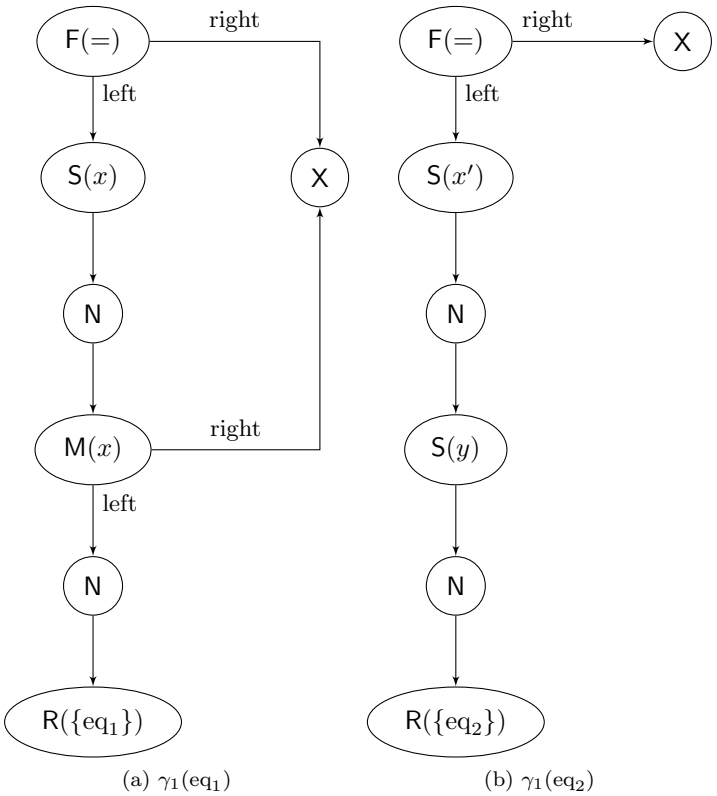
These rules have the following match trees (as depicted in Fig. 3.6):

$$\begin{array}{ll}
m_{\mathrm{eq}_1} & \mathsf{F}(=, \mathsf{S}(x, \mathsf{N}(\mathsf{M}(x, \mathsf{N}(\mathsf{R}(\{\mathrm{eq}_1\})), \mathsf{X}))), \mathsf{X}) \\
m_{\mathrm{eq}_2} & \mathsf{F}(=, \mathsf{S}(x', \mathsf{N}(\mathsf{S}(y, \mathsf{N}(\mathsf{R}(\{\mathrm{eq}_2\}))))), \mathsf{X})
\end{array}$$

Our priority function is defined by $\varphi(S) = S$ if $\{\mathrm{eq}_1, \mathrm{eq}_2\} \not\subseteq S$ and $\varphi(\{\mathrm{eq}_1, \mathrm{eq}_2\} \cup S) = \{\mathrm{eq}_1\} \cup (S \setminus \{\mathrm{eq}_2\})$. After combining both these trees we get the tree $T = \mathsf{F}(=, \mathsf{S}(x, \mathsf{S}(x', \mathsf{N}(\mathsf{S}(y, \mathsf{M}(x, \mathsf{N}(\mathsf{R}(\{\mathrm{eq}_1, \mathrm{eq}_2\})), \mathsf{N}(\mathsf{R}(\{\mathrm{eq}_2\}))))))), \mathsf{X})$ (Fig. 3.7a). Then $\mathrm{prior}(T, \varphi) = \mathsf{F}(=, \mathsf{S}(x, \mathsf{S}(x', \mathsf{N}(\mathsf{S}(y, \mathsf{M}(x, \mathsf{N}(\mathsf{R}(\{\mathrm{true}\})), \mathsf{N}(\mathsf{R}(\{\mathrm{false}\}))))))), \mathsf{X})$ (Fig. 3.7b).

3.4 Optimisation

As we have observed in the previous section, the match trees generated by γ are typically not optimal. They assign one term to multiple variables, check certain



(a) $\gamma_1(\mathrm{eq}_1)$ (b) $\gamma_1(\mathrm{eq}_2)$

Figure 3.6: Match trees for eq_1 and eq_2

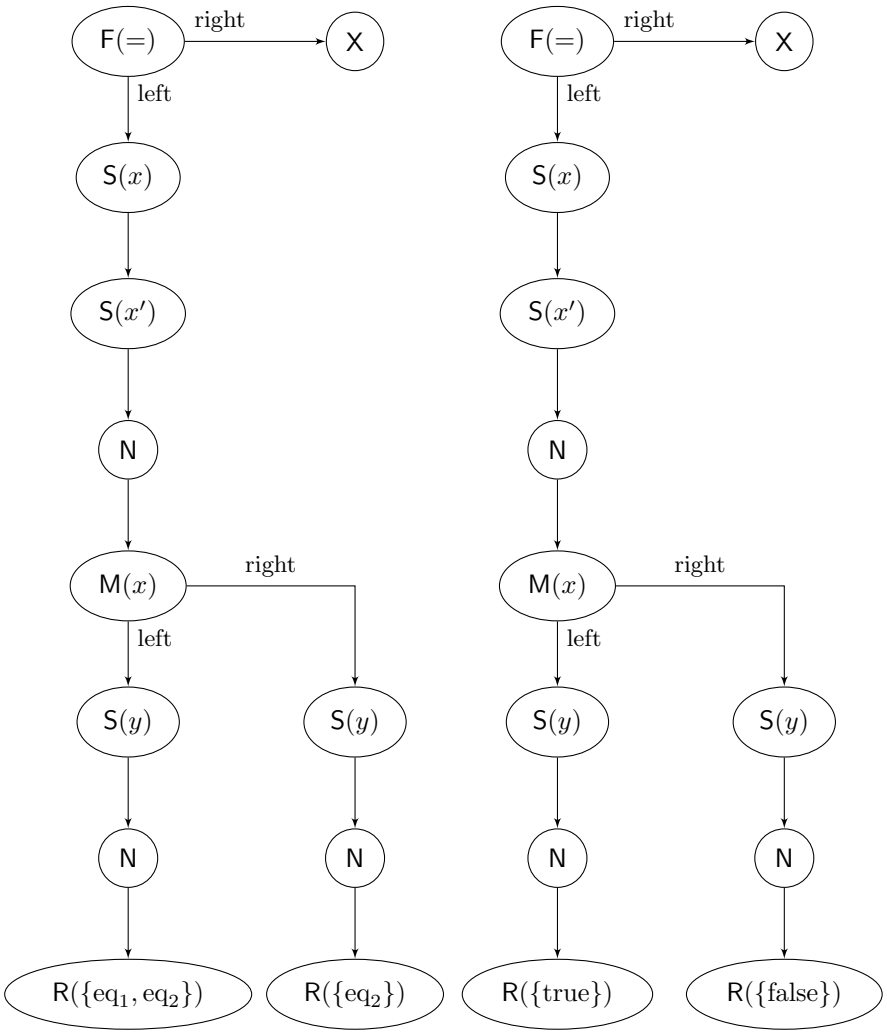
things multiple times etc. In this section we focus on manipulation of match trees in order to get better performance.

The most desirable path would be to establish a notion of optimality and give a method to convert any tree to its optimal equivalent. However, their might be more than one optimal equivalent match tree. Consider, for example, the following trees.

$$\mathsf{M}(x, \mathsf{M}(y, T, U), \mathsf{M}(y, V, W))$$
$$\mathsf{M}(y, \mathsf{M}(x, T, V), \mathsf{M}(x, U, W))$$

Virtually any (sensible) measure on match trees will equate both trees. Only if the order in which values of variables are accessed is significant and taken into account (e.g. because they are stored in linked lists for some reason), it is possible to say that one is better than the other.

Of course, it is fairly simple to impose an (arbitrary) order on the variables (similar to what is done with Binary Decision Diagrams). One does have to consider



(a) $\gamma(\{\mathrm{eq}_1, \mathrm{eq}_2\})$ (b) $\mathrm{prior}(\gamma(\{\mathrm{eq}_1, \mathrm{eq}_2\}), \varphi)$

Figure 3.7: Match tree for $\{\mathrm{eq}_1, \mathrm{eq}_2\}$

when such an order should be used. In the above solution the choice might not effect the performance, but if one imposes this order on all M nodes it might have undesirable consequences. This can clearly be seen by the following equivalent trees, where by changing x to y and vice versa one can change which one is the

optimal tree.

$$\mathsf{M}(x, \mathsf{M}(y, T, U), V))$$
$$\mathsf{M}(y, \mathsf{M}(x, T, V), \mathsf{M}(x, U, V))$$

In most situations one would expect the first tree to be preferred above the second one.

Besides there not necessarily being a unique optimal representation, which is in itself not directly a problem, there are some more troublesome issues with defining a good measure on match trees. One of them is how to value a C node. Because executing a C typically requires rewriting a term, it is very hard to estimate the precise cost (time-wise). Typically, rewriting is very expensive which suggests to postpone a C as long as possible. However, it might very well be the case that an early C can actually avoid unnecessary computation. An extreme example would be the following:

$$\begin{array}{lcll} x & \to & 0 & \text{if true} \\ f(g(\ldots), h(k(\ldots), \ldots), \ldots) & \to & t & \text{if false} \end{array}$$

The same holds for M to some extent as well if establishing (in)equality of terms is expensive. However, M nodes have significantly less freedom of "movement".

Finally, consider the following two (equivalent) match trees.

$$\mathsf{S}(x, \mathsf{N}(\mathsf{S}(y, \mathsf{M}(x, \mathsf{N}(\mathsf{R}(\{f(x,x)\})), \mathsf{N}(\mathsf{C}(g(x,y), \mathsf{R}(\{f(y,x)\}), \mathsf{R}(\{f(x,y)\}))))))))$$
$$\mathsf{S}(x, \mathsf{N}(\mathsf{S}(y, \mathsf{N}(\mathsf{C}(g(x,y), \mathsf{R}(\{f(y,x)\}), \mathsf{R}(\{f(x,y)\}))))))$$

The second tree has one less M, but at the cost of always having to execute the C node. Even if one has a suitable order on these trees, the task of obtaining one out of the other is far from trivial. This would involve adding temporary nodes and non trivial substitutions (e.g. changing $f(x,x)$ to $f(x,y)[y/x]$).

Instead of searching for optimal results, we focus on eliminating the most obvious inefficiencies. In order to do so, we first investigate the structure of the result of γ. We can make the following observations:

1. Trees consist of segments; each segment takes care of a specific subterm. The borders are indicated by N nodes or the first argument of F nodes (i.e. those places which result in a change on the stack and thus a change in the current term).

2. Each segment consists of C, E M, S and F nodes (in that order and at most one E) and finally an X, N or R. The latter node can only occur first, as argument of C or as right argument of E.

For example, in the tree $\mathsf{S}(x, \mathsf{F}(f, \mathsf{M}(x, \mathsf{N}(\mathsf{R}(\{t\})), \mathsf{X}), \mathsf{F}(g, \mathsf{R}(\{u\})), \mathsf{X}))$ we have segments $\mathsf{S}(x, \mathsf{F}(f, \ , \mathsf{F}(g, \ , \mathsf{X})))$, $\mathsf{M}(x, \mathsf{N}(\ ), \mathsf{X})$, $\mathsf{R}(\{t\})$ and $\mathsf{R}(\{u\})$. Next we observe some equivalences between trees:

1. Multiple S nodes within a segment are not needed; each variable is assigned the same value.
2. An $\mathsf{F}(f)$ in the right alternative (and same segment) of another $\mathsf{F}(f)$ will result in the former always choosing its right alternative. This means that such an F can be eliminated.
3. The same holds for $\mathsf{M}(x)$ nodes within $\mathsf{M}(x)$ nodes.
4. Nodes M and C with the same tree as left and right argument can be eliminated.
5. $\mathsf{S}(x)$ nodes can be eliminated if its subtree does not contain the variable x (unbound).
6. An E node can be eliminated if one of its subtrees is X

Taking these observations in to account, we want to convert each tree into an equivalent tree that adheres to the following.

1. An S node has no S node below it within the same segment.
2. An $\mathsf{M}(x)$ node has no $\mathsf{M}(x)$ node below it within the same segment.
3. An $\mathsf{M}(f)$ node has no $\mathsf{M}(f)$ node below it within the same segment.
4. Nodes M and C do not have the same tree as both left and right argument.
5. Each $\mathsf{S}(x)$ is useful. That is, x occurs (unbound) in its subtree.
6. No E node has X as a subtree.

The first three points are taken care of by the function reduce. The last three by function clean. These functions are defined as follows. Here reduce itself just traverses a tree and calls the $\text{reduce}_{\mathsf{F}}$, $\text{reduce}_{\mathsf{S}}$ and $\text{reduce}_{\mathsf{M}}$ to take care of the blocks of F, S and M nodes, respectively. The second parameter of $\text{reduce}_{\mathsf{F}}$ is to keep track of the function symbols we have seen (and are not the head symbol of the "current term"). In $\text{reduce}_{\mathsf{S}}$ the second argument is a set of variables for which we have encountered an S node and which will all be renamed to one particular variable in the end (with $[x/V]$ as defined below). The two extra arguments of $\text{reduce}_{\mathsf{M}}$ keep track of the variables that have been checked (against the "current term") and according to the result of this check (i.e. M_t for variables that were equal and M_f for those that were not).

Definition reduce**.** The definition of reduce is as follows.

$$
\begin{array}{lcll}
\mathrm{reduce}(\mathsf{X}) & = & \mathsf{X} & \\
\mathrm{reduce}(\mathsf{F}(f,T,U)) & = & \mathrm{reduce}_{\mathsf{F}}(\mathsf{F}(f,T,U),\emptyset) & \\
\mathrm{reduce}(\mathsf{S}(x,T)) & = & \mathrm{reduce}_{\mathsf{S}}(\mathsf{S}(x,T),\emptyset) & \\
\mathrm{reduce}(\mathsf{M}(x,T,U)) & = & \mathrm{reduce}_{\mathsf{M}}(\mathsf{M}(x,T,U),\emptyset,\emptyset) & \\
\mathrm{reduce}(\mathsf{C}(t,T,U)) & = & \mathsf{C}(t,\mathrm{reduce}(T),\mathrm{reduce}(U)) & \\
\mathrm{reduce}(\mathsf{N}(T)) & = & \mathsf{N}(\mathrm{reduce}(T)) & \\
\mathrm{reduce}(\mathsf{E}(T,U)) & = & \mathsf{E}(\mathrm{reduce}(T),\mathrm{reduce}(U)) & \\
\mathrm{reduce}(\mathsf{R}(R)) & = & \mathsf{R}(R) & \\
 & & & \\
\mathrm{reduce}_{\mathsf{F}}(\mathsf{X},F) & = & \mathrm{reduce}(\mathsf{X}) & \\
\mathrm{reduce}_{\mathsf{F}}(\mathsf{F}(f,T,U),F) & = & \mathrm{reduce}_{\mathsf{F}}(U,F) & \text{if } f \in F \\
\mathrm{reduce}_{\mathsf{F}}(\mathsf{F}(f,T,U),F) & = & \mathsf{F}(f,\mathrm{reduce}(T), & \\
 & & \quad \mathrm{reduce}_{\mathsf{F}}(U,F\cup\{f\})) & \text{if } f \notin F \\
\mathrm{reduce}_{\mathsf{F}}(\mathsf{N}(T),F) & = & \mathrm{reduce}(\mathsf{N}(T)) & \\
 & & & \\
\mathrm{reduce}_{\mathsf{S}}(\mathsf{X},\emptyset) & = & \mathrm{reduce}(\mathsf{X}) & \\
\mathrm{reduce}_{\mathsf{S}}(\mathsf{X},\{x\}\cup V) & = & \mathsf{S}(x,\mathrm{reduce}(\mathsf{X}[x/V],\emptyset)) & \\
\mathrm{reduce}_{\mathsf{S}}(\mathsf{F}(f,T,U),\emptyset) & = & \mathrm{reduce}_{\mathsf{F}}(\mathsf{F}(f,T,U),\emptyset) & \\
\mathrm{reduce}_{\mathsf{S}}(\mathsf{F}(f,T,U),\{x\}\cup V) & = & \mathsf{S}(x,\mathrm{reduce}_{\mathsf{F}}(\mathsf{F}(f,T,U)[x/V],\emptyset)) & \\
\mathrm{reduce}_{\mathsf{S}}(\mathsf{S}(x,T),V) & = & \mathrm{reduce}_{\mathsf{S}}(T,V\cup\{x\}) & \\
\mathrm{reduce}_{\mathsf{S}}(\mathsf{N}(T),\emptyset) & = & \mathrm{reduce}(\mathsf{N}(T)) & \\
\mathrm{reduce}_{\mathsf{S}}(\mathsf{N}(T),\{x\}\cup V) & = & \mathsf{S}(x,\mathrm{reduce}(\mathsf{N}(T)[x/V])) & \\
 & & & \\
\mathrm{reduce}_{\mathsf{M}}(\mathsf{X},S,M_t,M_f) & = & \mathrm{reduce}(\mathsf{X}) & \\
\mathrm{reduce}_{\mathsf{M}}(\mathsf{F}(f,T,U),M_t,M_f) & = & \mathrm{reduce}_{\mathsf{F}}(\mathsf{F}(f,T,U),\emptyset) & \\
\mathrm{reduce}_{\mathsf{M}}(\mathsf{S}(x,T),M_t,M_f) & = & \mathrm{reduce}_{\mathsf{S}}(\mathsf{S}(x,T),\emptyset) & \\
\mathrm{reduce}_{\mathsf{M}}(\mathsf{M}(x,T,U),M_t,M_f) & = & \mathrm{reduce}_{\mathsf{M}}(T,M_t,M_f) & \text{if } x \in M_t \\
\mathrm{reduce}_{\mathsf{M}}(\mathsf{M}(x,T,U),M_t,M_f) & = & \mathrm{reduce}_{\mathsf{M}}(U,M_t,M_f) & \text{if } x \in M_f \\
\mathrm{reduce}_{\mathsf{M}}(\mathsf{M}(x,T,U),M_t,M_f) & = & \mathsf{M}(x,\mathrm{reduce}_{\mathsf{M}}(T,M_t\cup\{x\},M_f) & \text{if } x \notin M_t \\
 & & \quad \mathrm{reduce}_{\mathsf{M}}(U,M_t,M_f\cup\{x\})) & \wedge\ x \notin M_f \\
\mathrm{reduce}_{\mathsf{M}}(\mathsf{N}(T),M_t,M_f) & = & \mathrm{reduce}(\mathsf{N}(T)) & \\
 & & & \\
\mathsf{X}[x/V] & = & \mathsf{X} & \\
\mathsf{F}(f,T,U)[x/V] & = & \mathsf{F}(f,T[x/V],U[x/V]) & \\
\mathsf{S}(x,T)[y/V] & = & \mathsf{S}(x,T[y/(V\setminus\{x\})]) & \\
\mathsf{M}(x,T,U)[y/V] & = & \mathsf{M}(y,T[y/V],U[y/V]) & \text{if } x \in V \\
\mathsf{M}(x,T,U)[y/V] & = & \mathsf{M}(x,T[y/V],U[y/V]) & \text{if } x \notin V \\
\mathsf{C}(t,T,U)[x/V] & = & \mathsf{C}(t[x/y\ :\ y\in V], & \\
 & & \quad T[x/V],U[x/V]) & \\
\mathsf{N}(T)[x/V] & = & \mathsf{N}(T[x/V]) & \\
\mathsf{R}(R)[x/V] & = & \mathsf{R}(R[x/y\ :\ y\in V]) &
\end{array}
$$

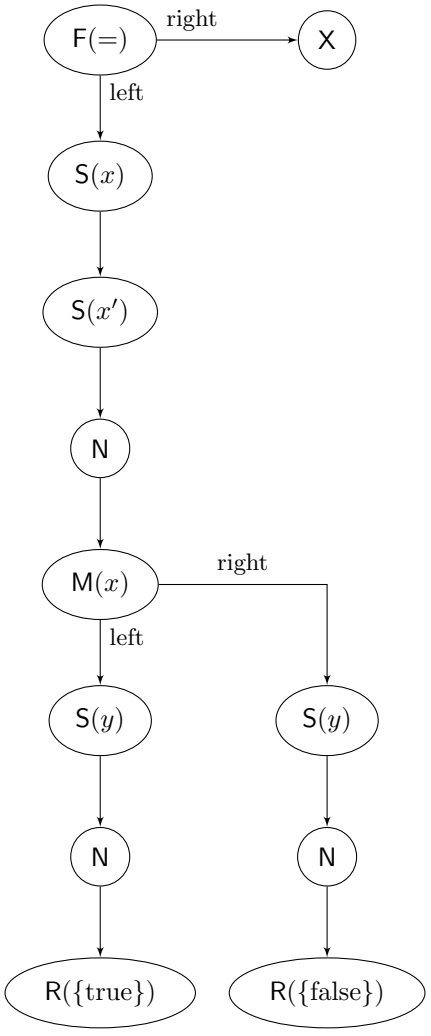
The clean function uses an auxiliary function clean' that returns a pair of the "cleaned-up" tree and a set of the unbound variables in this tree. The latter is used to remove unnecessary S nodes.

Definition clean**.** The definition of clean is as follows.

$$
\begin{array}{lcll}
\text{clean}(T) & = & T' & \text{if } \langle T', V\rangle = \text{clean}'(T) \\
 & & & \\
\text{clean}'(\mathsf{X}) & = & \langle \mathsf{X}, \emptyset\rangle & \\
\text{clean}'(\mathsf{F}(f,T,U)) & = & \langle \mathsf{F}(f,T',U'), V \cup W\rangle & \text{if } \langle T', V\rangle = \text{clean}'(T) \wedge \\
 & & & \quad \langle U', W\rangle = \text{clean}'(U) \\
\text{clean}'(\mathsf{S}(x,T)) & = & \langle \mathsf{S}(x,T'), V \setminus \{x\}\rangle & \text{if } \langle T', V\rangle = \text{clean}'(T) \wedge \\
 & & & \quad x \in V \\
\text{clean}'(\mathsf{S}(x,T)) & = & \langle T', V\rangle & \text{if } \langle T', V\rangle = \text{clean}'(T) \wedge \\
 & & & \quad x \notin V \\
\text{clean}'(\mathsf{M}(x,T,U)) & = & \langle T', V\rangle & \text{if } \langle T', V\rangle = \text{clean}'(T) \wedge \\
 & & & \quad \langle U', W\rangle = \text{clean}'(U) \wedge \\
 & & & \quad T' = U' \\
\text{clean}'(\mathsf{M}(x,T,U)) & = & \langle \mathsf{M}(x,T',U'), V \cup W \cup \{x\}\rangle & \text{if } \langle T', V\rangle = \text{clean}'(T) \wedge \\
 & & & \quad \langle U', W\rangle = \text{clean}'(U) \wedge \\
 & & & \quad T' \neq U' \\
\text{clean}'(\mathsf{C}(t,T,U)) & = & \langle T', V\rangle & \text{if } \langle T', V\rangle = \text{clean}'(T) \wedge \\
 & & & \quad \langle U', W\rangle = \text{clean}'(U) \wedge \\
 & & & \quad T' = U' \\
\text{clean}'(\mathsf{C}(t,T,U)) & = & \langle \mathsf{C}(t,T',U'), & \\
 & & \quad V \cup W \cup \text{var}(t)\rangle & \text{if } \langle T', V\rangle = \text{clean}'(T) \wedge \\
 & & & \quad \langle U', W\rangle = \text{clean}'(U) \wedge \\
 & & & \quad T' \neq U' \\
\text{clean}'(\mathsf{N}(T)) & = & \langle \mathsf{N}(T'), V\rangle & \text{if } \langle T', V\rangle = \text{clean}'(T) \\
\text{clean}'(\mathsf{E}(T,U)) & = & \langle T', V\rangle & \text{if } \langle T', V\rangle = \text{clean}'(T) \wedge \\
 & & & \quad \langle \mathsf{X}, W\rangle = \text{clean}'(U) \\
\text{clean}'(\mathsf{E}(T,U)) & = & \langle U', W\rangle & \text{if } \langle \mathsf{X}, V\rangle = \text{clean}'(T) \wedge \\
 & & & \quad \langle U', W\rangle = \text{clean}'(U) \\
\text{clean}'(\mathsf{E}(T,U)) & = & \langle \mathsf{E}(T',U'), V \cup W\rangle & \text{if } \langle T', V\rangle = \text{clean}'(T) \wedge \\
 & & & \quad \langle U', W\rangle = \text{clean}'(U) \wedge \\
 & & & \quad T' \neq \mathsf{X} \wedge U' \neq \mathsf{X} \\
\text{clean}'(\mathsf{R}(R)) & = & \langle \mathsf{R}(R), \text{var}(R)\rangle &
\end{array}
$$

To illustrate the use of these functions we look at the following example.

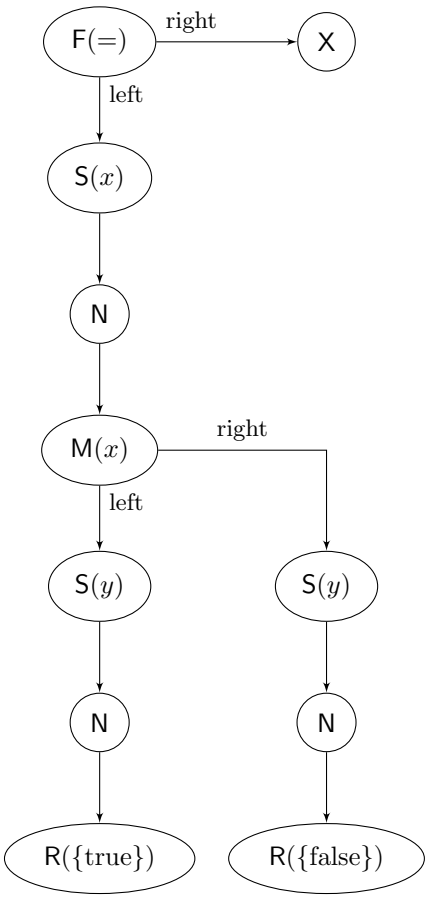
Example 3.4.1 Consider the rules of Example 3.3.5. In Fig. 3.8 the process of applying reduce and clean is depicted. Figure 3.8a depicts the match tree before reductions, Figure 3.8b depicts the same tree after application of reduce and in Figure 3.8c also clean has been applied.



(a) prior($\gamma(\{\mathrm{eq}_1, \mathrm{eq}_2\}), f$)

Figure 3.8: Optimisations for $\{\mathrm{eq}_1, \mathrm{eq}_2\}$

The function reduce and clean are sound, which is stated in the following theorems.



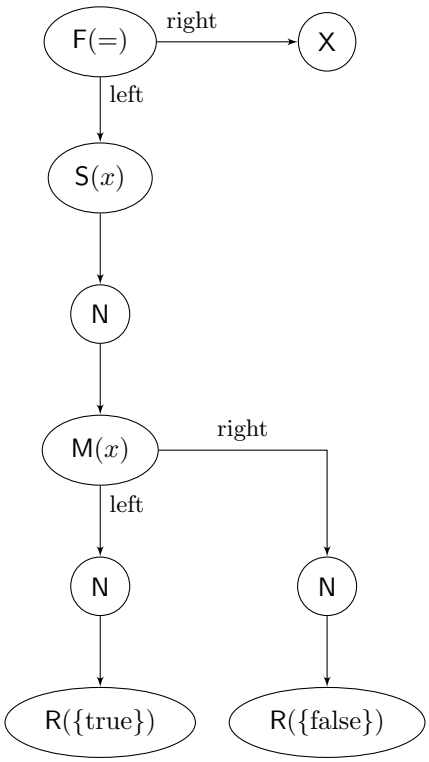
(b) reduce(prior($\gamma(\{\mathrm{eq}_1, \mathrm{eq}_2\}), f$))

Figure 3.8: Optimisations for $\{\mathrm{eq}_1, \mathrm{eq}_2\}$

Theorem 3.4.1. $\mathrm{reduce}(T) =_\mu T$

Theorem 3.4.2. $\mathrm{clean}(T) =_\mu T$

Of course, there are many more improvements that can be made. For example, it might be useful to introduce a node $\mathsf{N}(n, T)$ that is equivalent to $\mathsf{N}^n(T)$. Also, it is possible to remove S nodes that occur in left arguments of $\mathsf{M}(x)$ nodes; in this left argument one knows that the value assigned to x is equal to the current term,



(c) clean(reduce(prior($\gamma(\{\mathrm{eq}_1, \mathrm{eq}_2\}), f$)))

Figure 3.8: Optimisations for $\{\mathrm{eq}_1, \mathrm{eq}_2\}$

so there is no need to assign this value to another variable.

To avoid long sequences of F nodes that try a large number of function symbols, one can introduce a node $\mathsf{F}(\varphi)$, where φ is a function mapping function symbols to match trees. In practice we expect this to be mostly useful at the top level. This effectively means that one groups all rewrite rules by the head symbol of their left-hand side and has a quick lookup (e.g. an array) to get the match tree for a given function symbol. At other levels we expect this method to be too expensive

(in terms of memory) in general.

Note that we have done very little with C nodes (i.e. we only remove such a node if its subtrees are the same). These nodes can easily be "moved" around the tree (as long as they do not cross a capturing S). Also, there are a number of transformations that can be done based on the condition itself (e.g. adding/eliminating negation by swapping subtrees and splitting/merging C nodes using conjunctions and disjunctions in the conditions). As discussed before, it is not always clear what is most desirable.

Chapter 4

Temporary-Term Construction

4.1 Introduction

Besides using efficient strategies and matching, there is another important aspect of rewriting where significant gains in performance can be made. With each application of a rewrite rule one needs to construct a new term based on the right-hand side of the applied rule. We call these terms **temporary terms**. In constructing these terms, we can apply various optimisations. A simple example of such an optimisation is that instead of constructing a term $f(t)$ and then calling the rewriter on this term, we can also construct just t and directly call the specialised rewriter for function symbol f with t as its input.

One of the most significant optimisations is to add information to terms to be able to avoid rewriting terms that are already rewritten before. To illustrate, we consider again the addition (+) with the following rewrite rules.

$$\begin{array}{lcl} n+0 & \rightarrow & n \\ n+S(m) & \rightarrow & S(n+m) \end{array}$$

It is clear from these rewrite rules that both arguments of the + will always be (directly or indirectly) rewritten to normal form. So let us assume a strategy for + that first rewrites both arguments to normal form and then tries to apply the above rules. If the second rule is applied, we get a term of the form $S(n+m)$ where we know that n and m are in normal form. However, the first thing that happens when rewriting the subterm $n+m$ is that both n and m are rewritten to normal form. Even though they are already in normal form, we still have to reestablish this as this information is not explicitly available. By adding annotations to terms we can make such information explicit.

Another optimisation is to avoid constructing temporary terms of which one

knows that they will be rewritten later on. In such a case it might be advantageous to rewrite immediately without constructing the temporary terms. Take, for example, the following rewrite rules for a function symbol f: $f(0) \to c$ and $f(S(n)) \to h(n)$. A typical strategy will always rewrite the argument of f before trying to apply a rewrite rule. This means that if we are constructing some temporary term $f(k(x))$, we already know that the first thing that will happen to such a term is that $k(x)$ will be rewritten. Thus, instead of constructing this temporary (sub)term, we can just as well immediately rewrite $k(x)$ to its normal form t and then rewrite $f(t)$. As we can directly call a specialised rewrite function for head symbol k, supplying x as argument, there is no need for the explicit construction of the term $k(x)$.

Finally, we also use the fact that some functions have no rewrite rules at all and, if applied to normal forms, lead to a normal form themselves. Consider, for example, the rewrite rule $f(g(x)) \to h(k(x))$ and assume that before application of this rule the rewriter will always rewrite the argument of f first. In this situation, the right-hand side of the rewrite rule, $h(k(x))$, would be used to create a temporary term which is annotated such that it is known that the value substituted for x is in normal form. However, if k has no rewrite rules, we can actually annotate the temporary term to indicate that the whole argument of h is in normal form.

An important aspect of these optimisations is that they can influence which normal form(s) the rewriter returns for a given term. By adding annotations to terms, it is possible (and often useful) to make rewrite strategies in such a way that they take into account these annotations. For example, if you have the term $if(t, u, v)$ annotated in such a way that we know that u and v are in normal form, then it is useful to first try to apply the rules $if(b, x, x) \to x$ instead of first rewriting t to normal form and then trying the rules for if. Doing so, however, means that due to the annotations different rules are applied and thus, possibly, other normal forms are obtained. Note that using knowledge about constructor functions and rewriting subterms earlier both influence the annotation in the eventual temporary term (and thus the outcome of rewriting).

Note that these techniques are mainly useful for non-innermost strategies. Essential argument detection is trivial as all arguments are always rewritten and normal form annotations can be avoided by using the "trick" discussed in Section 2.2.

In this chapter we look at how we can implement the above optimisations for sequential strategies. First we introduce function annotations for normal form information in Section 4.2. Then, in Section 4.3, we give a construction algorithm that is suitable for all of the above optimisations. Finally, we give a method to determine which arguments of a function will always be rewritten in Section 4.4.

4.2 Annotations

An **annotation** is a set of indices that occurs as superscript of a function symbol. We add such annotations in terms to indicate which (sub)terms are known to be in normal form. For example, we will use $f^{\{2\}}(t, u)$ for the term $f(t, u)$ if we know that u is in normal form. Note that we only annotate those function symbols that are themselves not part of a normal form; this information is already implied by annotations of the term that contains this normal form as subterm.

Of course, the input and output of a rewriter must still be unannotated terms. The way we assure this is to implement the rewriter in the following way. When an unannotated term needs to be rewritten, we do not – and can not – assume anything about this term. The only thing to do with an unannotated term is to simply rewrite it. However, during this rewriting we construct various **temporary** terms (i.e. terms that do not necessarily occur in the result). For example, when rewriting the term $n + S(m)$ we do not immediately return the normal form of this term but we apply one rewrite rule to get the temporary term $S(n + m)$ and then rewrite this temporary term. In the construction of these temporary terms we introduce the function annotations for all function symbols in the temporary term that are not known to be part of a subterm that is in normal form. By leaving the known normal forms unannotated we trivially get that the result will also be unannotated. Note that this means that matching will still be performed on unannotated terms when using in-time strategies.

Now, when the rewriter encounters an annotated term, it can execute a strategy that takes into account that certain subterms are already in normal form. For example, if we look at the if function, we can use a sequential strategy $[\{\gamma\}, \{1\}, \{\alpha, \beta\}]$ for terms with head symbol $\text{if}^{\{2,3\}}$ (instead of the normal strategy $[\{1\}, \{\alpha, \beta\}, \{2, 3\}, \{\gamma\}]$ from Section 2.1).

Another way to add this normal form information to terms is to add a marker to terms that are known to be in normal form. This approach is taken in [vdP02] were $\nu(t)$ is written to indicate that t is in normal form. The reason we prefer the function annotations is that with it we can avoid additional term manipulations – in particular term construction. With the marker we need to construct the additional ν around normal forms and, when it is requested to rewrite such a term to normal form, we must remove it again. With function annotation we do not need to construct an additional symbol; we just use a different symbol as head symbol. Also, because this normal-form information is in the head symbol itself and each annotated head symbol has its own rewrite strategy, we will avoid ever requesting that a known normal form is rewritten (avoiding a call to the rewriter and the removal of a marker). Note that making the marker part of the head symbol of a normal form does not avoid the extra term manipulations; the output should still be an unmarked term and thus these marked head symbols have to be unmarked at some point.

The process of making strategies for the annotated head symbols is quite straightforward. For example, for the sequential strategies this just means using the an-

notation as input of the generation process (e.g. strat from Section 2.1) instead of the initially empty set. In the next section we have a look at the construction of temporary terms using these function annotations.

4.3 Construction

In this section we give an algorithm for the construction of temporary terms using function annotations. This algorithm takes a term t and a substitution σ (the result of a successful match) and constructs an expression e – consisting of annotated terms and calls to rewrite functions – which, when evaluated, gives the normal form of t. Of course, we try to do this in such a way that there are sufficient annotations to avoid rewriting normal forms and that we use as much information about normal forms as is available to us.

We observe two different sources of information about normal forms. Our primary source comes from the substitution that is the result of matching. For each concrete application of a rewrite, we can use annotations of the input term and the executed rewriting strategy to establish which variables the substitution certainly maps to a normal form. For example, for a term of the form $f^{\{1\}}(t,u)$ and a strategy $\varsigma(f^{\{1\}}) = [\{2\}, \{f(x,y) \to r\}]$, we know that matching $f^{\{1\}}(t,u)$ to $f(x,y)$ results in a substitution σ where both x and y are mapped to normal forms. For this reason we have an extra parameter $N \subseteq \mathbb{V}$ for the algorithm such that $\sigma(x) \in \mathrm{nf}(\sigma(x))$ for all $x \in N$.

Secondly, we might have function symbols that have no rewrite rules. If we know that terms $t_1, \ldots, t_n$ are in normal form and there are no rewrite rules for function symbol f, then we also know that $f(t_1, \ldots, t_n)$ is a normal form.

The **term construction function** $\varphi^N_\sigma(t,b)$ returns a tuple $\langle u, c\rangle$ such that u is the term constructed from t with substitution σ (where $\sigma(x) \in \mathrm{nf}(\sigma(x))$ for all $x \in N$), c indicates whether or not u is in normal form and b specifies whether u *must* be in normal form. Its definition is as follows. We write $\{\vec{b_i}\}$ for $\{i \,:\, 1 \le i \le n \,\wedge\, b_i\}$ and $P(b)$ for $\forall_{1\le i\le n}(\langle t'_i, b_i\rangle = \varphi^N_\sigma(t_i, b))$.

$$
\begin{array}{lcll}
\varphi^N_\sigma(x, \mathrm{false}) & = & \langle \sigma(x), x \in N\rangle & \\
\varphi^N_\sigma(x, \mathrm{true}) & = & \langle \sigma(x), \mathrm{true}\rangle & \text{if } x \in N \\
\varphi^N_\sigma(x, \mathrm{true}) & = & \langle \mathrm{rewrite}(\sigma(x)), \mathrm{true}\rangle & \text{if } x \notin N \\
\varphi^N_\sigma(f(t_1,\ldots,t_n), \mathrm{false}) & = & \langle f(t'_1,\ldots,t'_n), \mathrm{true}\rangle & \text{if } R_f = \emptyset \wedge \\
 & & & \quad \forall_{1\le i\le n}(b_i) \wedge \\
 & & & \quad P(\mathrm{false}) \\
\varphi^N_\sigma(f(t_1,\ldots,t_n), \mathrm{false}) & = & \langle f^{\{\vec{b_i}\}}(t'_1,\ldots,t'_n), \mathrm{false}\rangle & \text{if } \neg(R_f = \emptyset \wedge \\
 & & & \quad \forall_{1\le i\le n}(b_i)) \wedge \\
 & & & \quad P(\mathrm{false})
\end{array}
$$

$$\begin{array}{lcll} \varphi^N_\sigma(f(t_1,\ldots,t_n),\mathrm{true}) & = & \langle f(t'_1,\ldots,t'_n),\mathrm{true}\rangle & \text{if } R_f = \emptyset \wedge P(\mathrm{true}) \\ \varphi^N_\sigma(f(t_1,\ldots,t_n),\mathrm{true}) & = & \langle \mathrm{rewrite}_{f\{\vec{b_i}\}}(t'_1,\ldots,t'_n),\mathrm{true}\rangle & \text{if } R_f \neq \emptyset \wedge P(\mathrm{false}) \end{array}$$

Of course we require this function to be sound. That is, $t\sigma$ must rewrite to u and if b or c holds then u must really be in normal form. This is expressed in the following theorems. Note that we assume that the rewrite functions result in a normal form related to the input.

Theorem 4.3.1 *For all terms t and u, substitutions σ, sets of variables N and booleans b and c we have that if $\varphi^N_\sigma(t,b) = \langle u,c\rangle$ then $t\sigma \to^* u$.*

Theorem 4.3.2 *For all terms t and u, substitutions σ, sets of variables N and booleans b and c we have that if $\varphi^N_\sigma(t,b) = \langle u,c\rangle$ and $b \vee c$, then $u \in \mathrm{nf}(t\sigma)$.*

Besides the definition being sound, we also wish that it preserves as much information about normal forms as possible. Given a substitution σ and set of variables N, we say that a term $t\sigma$ is known to be in normal form if $\varphi^N_\sigma(t,\mathrm{false}) = \langle t\sigma,\mathrm{true}\rangle$ (i.e. without using actual rewriting, the construction of term $t\sigma$ results in a normal form). We wish that $t\sigma$ is known to be in normal form if none of the function symbols in t have rewrite rules and all variables of t are in N. This is expressed by Theorem 4.3.3. Besides this, we also want that if an argument of a function f is in normal form according to φ, that the annotation of f indicates this. It is easy to see that this is the case by inspection of the definition of φ.

Theorem 4.3.3 *Let t be a term, σ a substitution and N a set of variables. We have that $\varphi^N_\sigma(t,\mathrm{false}) = \langle u,\mathrm{true}\rangle$ for some u if, and only if, t is either a variable x such that $x \in N$, or $t = f(t_1,\ldots,t_n)$ with $R_f = \emptyset$ and $\varphi^N_\sigma(t_i,\mathrm{false}) = \langle u_i,\mathrm{true}\rangle$ for some terms u_i (with $1 \leq i \leq n$). Furthermore we have that if $\varphi^N_\sigma(t,\mathrm{false}) = \langle u,\mathrm{true}\rangle$ for some term u, then $u = t\sigma$.*

Note that we do not take into account that if substitution σ maps a variable to another (or the same) variable, the latter is in normal form (as all variables are in normal form). Adapting φ to take this into account is quite straightforward, but we doubt this would have a positive impact on performance. Determining whether a substitution results in a variable requires an additional check every time a rewrite rule is applied and for each variable occurrence in the right-hand side (and possibly the condition) of the rule. On the other hand, rewriting a variable to normal form is relatively cheap and is only needed when the result of the substition is actually a variable. Further investigation is needed to establish the actual impact of this choice.

It is worth observing that in the case one uses sequential strategies that effectively rewrite innermost, temporary terms consist solely of direct calls to the

rewriter and normal forms. For example, a temporary term for right-hand side $f(g(x))$ where x is matched with t would be $\text{rewrite}_{f\{1\}}(\text{rewrite}_{g\{1\}}(t))$. This corresponds with the optimisation as discussed in Section 2.2.

Also, there is the possibility to obtain a bit more information about function symbols. Assume that we have a term $t = f(t_1, \ldots, t_n)$ where terms t_i, for i with $1 \leq i \leq n$, are in normal form. At the moment we determine whether t is also in normal form by checking whether f has rewrite rules or not. Instead, one can check whether there is a rewrite rule of f that might match t (using unification). If this is not the case, we know that the term is in normal form as well.

4.4 Essential-Argument Detection

As an improvement to the term construction function φ of the previous section, we can determine beforehand whether an argument of a function that is to be rewritten will be rewritten as well in any case. We call such arguments **essential**.

Note that this notion is strongly related to the notion of *strictness* [PvE93]. The reason we do not use the term strictness is because it is a notion that has very much become connected to lazy evaluation as used in mainstream functional languages (e.g. Haskell, Clean). It is often also not very clear what the precise context is; for example, the way in which non-determinism is taken into account is hardly ever made explicit. Also, because the semantics of lazy functional languages is often defined by the used evaluation technique, it is not clear whether strictness should be considered a semantic or implementation notion. We use essential arguments as an implementation notion, meaning we do not check whether an argument is rewritten for all possible rewrite sequences but only for those that are actually the result of the use rewriting strategy.

Another closely related notion is that of *call-by-need* [HL91]. This method determines the subterms of a term t that need to be rewritten in any reduction (i.e. are *needed*) to a normal form. Only needed subterms are rewritten. The difference with our setting comes from the fact that call-by-need is defined purely with respect to the rewrite system and limits the allowed strategies to rewriting only needed terms. We do not restrict strategies in any such way and base our form of neededness (i.e. essentiality) on the strategies instead of the rewrite system. Of course, any practical strategy will rewrite to normal form (w.r.t. the rewrite system) and must therefore rewrite all needed terms. It may, however, also rewrite terms that are not needed. One could call such strategies approximations of call-by-need.

Assume we have a function ea that, given a function symbol f and argument index i, states whether or not argument i of f is an essential argument. That is, let $\text{ea}(f, i)$ hold if, and only if, the rewriting of a term of the form $f(t_1, \ldots, t_n)$ always results in the rewriting of term t_i.

With this we can change the last case of the definition of φ of the previous section

by replacing $P(\text{false})$ by $\forall_{1\leq i\leq n}(\langle t'_i, b_i\rangle = \varphi^N_\sigma(t_i, \mathbf{ea(f,i)}))$ (with the change in bold). It is straightforward to see that this adaptation does not influence the theorems of Section 4.3 by inspecting the proofs in Appendix D.

This change expresses that if a term will always be rewritten, we might as well do this immediately instead of constructing a temporary term that will need to be rewritten later on anyway. This can greatly reduce the number of temporary terms that are created and as such the time needed for rewriting. Temporary terms cost time to construct and rewriting them requires additional calls to the main rewrite function. Note, however, that rewriting subterms earlier can influence the outcome of rewriting (as discussed in Section 4.1).

To determine the function $\text{ea}(f, i)$ we must define what it means for an argument to be rewritten in any case. Because this function is not easily calculated due to recursive dependencies, we do this by defining a boolean equation system [Mad97]. This boolean equation system consists of variables $X_{f,i}$ corresponding to $\text{ea}(f, i)$, for all function symbols f and indices i. The value of such a variable $X_{f,i}$ depends on the strategy $\varsigma(f)$. We define a function $\psi(s, i)$ that determines the expression indicating whether or not execution of sequential strategy s will always result in the rewriting of argument i. For this function ψ we must be able to determine whether the specific rewrite rules from a strategy rewrite the given argument. Therefore we define the function $\xi(r, i)$ that gives the expression stating that rewrite rule r will result in rewriting argument i. Finally, for function ξ we must be able to determine whether the right-hand side of a rewrite rule has this effect. This is determined with function $\chi(t, \pi)$ that takes a term and a position and gives an expression indicating whether $t|_\pi$ will be rewritten.

We start with the definition of χ.

$$\begin{aligned}\chi(t, \epsilon) &= \text{true}\\ \chi(f(t_1, \ldots, t_n), i \cdot \pi) &= X_{f,i} \wedge \chi(t_i, \pi)\end{aligned}$$

So the subterm x of $f(g, h(x, k))$ is rewritten if both the second argument of f and the first argument of h are always rewritten (i.e. if $X_{f,2} \wedge X_{h,1}$).

With χ we can define ξ.

$$\xi(f(t_1, \ldots, t_n) \to u, i) = \bigvee_{\pi\in pos(t_i) \wedge u|_\pi = t_i} \chi(u, \pi)$$

That is, $\xi(f(t_1, \ldots, t_n) \to u, i)$ is true if argument t_i of f occurs as a subterm of u and that subterm will always be rewritten.

The function ψ on strategies then becomes as follows.

$\psi([], i) = \text{false}$; the empty strategy does nothing, which includes not rewriting the ith argument.

$\psi(I \rhd s, i) = i \in I \vee \psi(s, i)$; this strategy first rewrites the arguments from I and then continues with strategy ς, so argument i is rewritten if it is in I or if it is rewritten due to ς (or both).

$\psi(R \rhd s, i) = \bigwedge_{r \in R} \xi(r, i) \wedge \psi(s, i)$; this strategy tries to apply the rules from R and if none match it continues with s. We only know for sure that argument i is rewritten if it is rewritten in all possible courses of action.

To illustrate, the strategy of the *if* ($\varsigma(\mathit{if}) = [\{1\}, \{\alpha, \beta\}, \{2, 3\}, \{\gamma\}]$) results in $\psi(\varsigma(\mathit{if}), i) = i \in \{1\} \vee (\xi(\alpha, i) \wedge \xi(\beta, i) \wedge (i \in \{2, 3\} \vee (\xi(\gamma, i) \wedge \text{false})))$. That is, argument 1 while always be rewritten and arguments 2 and 3 will be written if $\xi(\alpha, 2) \wedge \xi(\beta, 2)$ and $\xi(\alpha, 3) \wedge \xi(\beta, 3)$, respectively.

For each variable $X_{f,i}$, indicating whether we know for sure that argument i of f will always be rewritten, we get the following (greatest) fixed-point equation.

$$\nu\, X_{f,i} \doteq \psi(\varsigma(f), i)$$

The greatest fixed-point operator ν means that if there are solutions where $X_{f,i}$ is true and solutions where it is false, then we are interested in the ones where it is true. The example at the end of this section illustrates this.

Note that in the definition of ξ arguments t_i need not be variables; they can actually be complex terms. As ξ is used to determine whether an argument t_i is used or not, this means that we assume that if a complex argument occurs in the right-hand side of a rewrite rule, this knowledge is used to avoid rewriting that subterm. For example, when applying rewrite rule $f(g(x)) \to h(g(x))$ to a term $f(g(t))$, this would mean that you do not construct the right-hand side by taking the value for x (i.e. t) and applying g and h to it, but by taking $g(t)$ from the original term and apply h to it.

However, for strategies that are in-time we know that complex arguments are rewritten before trying to apply a rule. We therefore know, by the definition of ψ that such an occurrence of ξ is a subterm of an expression of the form $i \in I \vee \ldots$ where $i \in I$ is true. Therefore this assumption does not play a role with such strategies. It is quite straightforward to adapt the definition of ξ if non-in-time strategies are used and the assumption does not hold.

Solving a boolean equation system can be done in various ways (see, for example, [Mad97, Mat03, GK04]). Here we only consider a small typical example to illustrate how we use boolean equation system to define ea. Consider the functions *add* and *mult* on natural numbers (defined by 0 and successor function S).

$$\begin{array}{lll}
\alpha_1: & \mathit{add}(0, y) & \to \quad y \\
\alpha_2: & \mathit{add}(S(x), y) & \to \quad S(\mathit{add}(x, y))
\end{array}$$

$$\begin{array}{lll}
\mu_1: & \mathit{mult}(0, y) & \to \quad 0 \\
\mu_2: & \mathit{mult}(S(x), y) & \to \quad \mathit{add}(\mathit{mult}(x, y), y)
\end{array}$$

Let $\varsigma_{add} = [\{1\}, \{\alpha_1, \alpha_2\}, \{2\}]$ and $\varsigma_{mult} = [\{1\}, \{\mu_1, \mu_2\}, \{2\}]$. We then get the following boolean equation system. Note that the strategy for S is $[\{1\}]$ (and for 0 it is $[]$).

$$\begin{array}{lcl}
\nu\, X_{add,1} & \doteq & 1 \in \{1\} \vee (\text{false} \wedge \text{false} \wedge (1 \in \{2\} \vee \text{false})) \\
\nu\, X_{add,2} & \doteq & 2 \in \{1\} \vee (\text{true} \wedge (X_{S,1} \wedge X_{add,2} \wedge \text{true}) \wedge (2 \in \{2\} \vee \text{false})) \\
\nu\, X_{mult,1} & \doteq & 1 \in \{1\} \vee (\text{true} \wedge \text{false} \wedge (1 \in \{2\} \vee \text{false})) \\
\nu\, X_{mult,2} & \doteq & 2 \in \{1\} \vee (\text{false} \wedge ((X_{add,1} \wedge X_{mult,2} \wedge \text{true}) \vee \\
 & & \qquad (X_{add,2} \wedge \text{true})) \wedge (2 \in \{2\} \vee \text{false})) \\
\nu\, X_{S,1} & \doteq & 1 \in \{1\} \vee \text{false}
\end{array}$$

This can be simplified with standard calculus to the following equations.

$$\begin{array}{lcl}
\nu\, X_{add,1} & \doteq & \text{true} \\
\nu\, X_{add,2} & \doteq & X_{S,1} \wedge X_{add,2} \\
\nu\, X_{mult,1} & \doteq & \text{true} \\
\nu\, X_{mult,2} & \doteq & \text{false} \\
\nu\, X_{S,1} & \doteq & \text{true}
\end{array}$$

It is easy to see that the greatest solution for $X_{add,2}$ is true and therefore we know that both arguments of an *add* will always be rewritten. For *mult* this is not the case as the rewrite rule μ_1 discards the second argument.

Note that even though the second argument of *mult* is not always rewritten, it might be useful to rewrite it anyway. In general usage the occurrence of $mult(0, y)$ might be relatively scarce. This is, however, outside the scope of this thesis.

Currently we only consider whether an argument will be rewritten for all possible instantiations of the arguments. But if we know that the arguments have a specific structure we can detect even more arguments that will be rewritten. Consider, for example, the *mult* in the example above. If we know beforehand that the first argument of *mult* will be of the form $S(t)$ for some term t, then the second argument will always be rewritten.

To use this information we can extend the method to pattern equation systems (i.e. equation systems where variables range over patterns or sets of terms). Here a variable $X_{f,i}$ will be the pattern (or set of patterns) on the arguments of f such that argument i will always be rewritten. To illustrate, the variable $X_{mult,2}$ would be assigned the pattern $(S(\_), \_)$ (the other variables allow any arguments). It is, however, important to note that this only works where arguments like $S(t)$ are known to be in normal form (unless one has strategies that are not in-time).

Due to the significant increase in complexity (i.e. working with patterns instead of booleans) one may wonder whether or not such a method is practically desirable or whether there are more straightforward approaches.

Another approach could be to establish with what probability arguments will be rewritten. This would allow a rewriter to rewrite subterms in advance if the probability that they will be rewritten anyway is greater than a certain percentage. Taking the *mult* example again, we might be able to deduce that in certain cases 0

as first argument of *mult* is much less likely then something of the structure $S(t)$. This approach does additionally raise the question what a reasonable percentage is for determining whether an argument should be rewritten and how to determine percentages.

Chapter 5

Strategy Trees

5.1 Introduction

In order to allow for more efficient rewriting we have used the notion of sequential strategies as defined in Section 2.1. With it we can avoid rewriting arguments of functions that are not needed for rewriting a term to normal form. We recall having the function *if* with the following rewrite rules.

$$\begin{array}{lcl} \mathit{if}(\text{true}, x, y) & \to & x \\ \mathit{if}(\text{false}, x, y) & \to & y \\ \mathit{if}(b, x, x) & \to & x \end{array}$$

By using a strategy that first rewrites the first argument of an *if*, we can immediately apply one of the first two rules and "dispose" of the unused argument if this first argument rewrites to true or false.

We have seen that this is a very effective method, but there is still room for improvement. Consider, for example, the function *len* that computes the length of a list. This function would typically have the following rules.

$$\begin{array}{lcl} \mathit{len}([]) & \to & 0 \\ \mathit{len}(s \rhd l) & \to & 1 + \mathit{len}(l) \end{array}$$

No matter what kind of sequential strategy one would make, one must at some point rewrite the argument of *len* to a normal form.

However, looking at the rules for *len*, it is evident that it is irrelevant what the actual elements of the list are for the result of *len*. To illustrate, the term $\mathit{len}([\omega])$, where ω only rewrites to itself, can be rewritten to $1 + 0$ in two steps even though the argument of *len* has no normal form.

We therefore desire a notion of strategies that allows us to tackle these problems. The most essential part of the solution will be the ability to rewrite a term only as long as its head symbol is not *stable* (i.e. there might be a sequence of rewrite

rules applicable that would change the head symbol) and afterwards continuing with a strategy based on such a head symbol. In the case of *len* one would expect to get something of the following form.

1. Rewrite argument until it has a stable head symbol.
2. If this head symbol is [], then return 0.
3. If this head symbol is $\triangleright$, then return $1+len(l)$ where l is the second argument of this $\triangleright$.
4. Else rewrite argument to normal form (as there are no rules that apply).

Due to the branching nature of these kind of strategies, we use a tree structure to construct them. The nodes of these *strategy trees* correspond to activities such as rewriting a term to stable-head or normal form, branching according to the head symbol of a subterm and trying to apply some rewrite rules.

In [FKW00] and with the E-strategies of OBJ languages [GWJMJ00, OF97] attempts are made to improve the termination behaviour of rewriting by indicating argument indices of functions as either "eager" or "lazy". The idea is that lazy arguments are never rewritten unless there is a specific reason to (e.g. in order to match). Take, for example, the rules below:

$$\begin{array}{lcl} take(0, l) & \rightarrow & [] \\ take(n+1, s \triangleright l) & \rightarrow & s \triangleright take(n, l) \\ from(n) & \rightarrow & n \triangleright from(n+1) \end{array}$$

Here one would typically indicate the first argument of $\triangleright$ as eager and the second argument of $\triangleright$ as lazy. This way $from(0)$ will not rewrite any further than $0 \triangleright from(1)$ and a term like $take(3, from(0))$ can be easily calculated without trying to determine a normal form of $from(0)$. With E-strategies one can also indicate function symbols themselves as eager or lazy. For example, one could declare ω to be lazy in order to avoid rewriting it in terms like $len([\omega])$.

Although we have not investigated this, we believe such strategies can be formulated as strategy trees as well (with exception of the *on-demand* extension of E-strategies). We do think, however, that focus of this approach, which lies on function symbols in themselves, is not an optimal one. In our mind the context in which a function symbol (as head of a subterm) occurs is a much better indicator of what needs to be done. For example, with strategy trees one can also let a strategy depend on subterms deeper than the arguments of the root; with the above strategies you have to depend on the strategy given for the function such a subterm is an argument of.

We first give the precise syntax and semantics of these strategy trees in Section 5.2 and show that sequential strategies can easily be translated to a strategy tree. In Section 5.3 we define a property on strategy trees that guarantees that their usage really results in normal forms. Using this property we give a strategy generation function that constructs a strategy tree given a set of rewrite rules (Section 5.4). We conclude this chapter with Section 5.5, a note on combining strategy and match trees.

5.2 Syntax and Semantics

Strategy trees T have the following syntax. Here π is a non-empty position, φ a function mapping function symbols and $\bot$ to strategy trees, Π a set of non-empty positions and R a set of rewrite rules from the TRS.

$$T ::= \mathsf{F}(\pi, \varphi) \mid \mathsf{H}(\Pi, T) \mid \mathsf{NF}(\Pi, T) \mid \mathsf{T}(R, T) \mid \mathsf{E} \mid \mathsf{X}$$

We write $\mathbb{ST}$ for the set of strategy trees.

The constructs NF, T and E correspond to respectively the rewriting of arguments to normal form, trying rewrite rules and the empty strategy of sequential strategies. The only difference is that NF accepts any set of non-empty positions (instead of a set of indices only).

Similar to NF, H rewrites subterms to (at least) stable-head forms. With F one can choose a strategy depending on the head symbol at the given position. Finally, with X one can indicate that a term has an infinite rewrite sequence.

A strategy using strategy trees is a tuple $\langle \varsigma, \varsigma_\mathrm{h} \rangle$, where ς and ς_h are functions that map function symbols to strategy trees that are to be used for rewriting. For function symbol f, $\varsigma(f)$ and $\varsigma_\mathrm{h}(f)$ are the strategy trees for rewriting a term with head symbol f to normal form, respectively stable-head form.

We extend hs to $\mathrm{hs}^\bot$ such that $\mathrm{hs}^\bot(x) = \bot$. Similarly we write $\varsigma^\bot$ (and $\varsigma_\mathrm{h}^\bot$) for the extension of ς such that $\varsigma(\bot) = \mathsf{E}$.

We now give semantics to the strategy tree constructs we have just introduced. The functions rewr and rewr_h are meant to give the normal forms, respectively stable-head forms for a given term (using the strategies supplied by ς and ς_h).

Definition rewr, rewr_h**.** Let $\langle \varsigma, \varsigma_\mathrm{h} \rangle$ be a strategy. We define the functions $\mathrm{rewr}, \mathrm{rewr}_\mathrm{h} : \mathbb{T} \to \mathcal{P}(\mathbb{T})$ and $\mathrm{eval}, \mathrm{eval}_\mathrm{h} : \mathbb{ST} \times \mathbb{T} \to \mathcal{P}(\mathbb{T})$ as follows. Note that the definition of eval and eval_h is a minimal fixed-point definition (see Appendix A).

$$\begin{aligned} \mathrm{rewr}(t) &= \mathrm{eval}(\varsigma^\bot(\mathrm{hs}^\bot(t)), t) \\ \mathrm{rewr}_\mathrm{h}(t) &= \mathrm{eval}_\mathrm{h}(\varsigma_\mathrm{h}^\bot(\mathrm{hs}^\bot(t)), t) \end{aligned}$$

(continued on next page)

$$
\begin{array}{lcll}
\mathrm{eval}(\mathsf{F}(\pi,\varphi),t) & =_\mu & \mathrm{eval}(\varphi(\mathrm{hs}^\bot(t|_\pi)),t) & \text{if } \pi \in \mathrm{pos_f}(t) \\
\mathrm{eval}(\mathsf{F}(\pi,\varphi),t) & =_\mu & \mathrm{eval}(\varphi(\bot),t) & \text{if } \pi \notin \mathrm{pos_f}(t) \\
\mathrm{eval}(\mathsf{H}(\Pi,T),t) & =_\mu & \bigcup_{\varphi\in\mathrm{rewrf_h}(t,\Pi)} \mathrm{eval}(T,t[\varphi]_\Pi) & \\
\mathrm{eval}(\mathsf{NF}(\Pi,T),t) & =_\mu & \bigcup_{\varphi\in\mathrm{rewrf}(t,\Pi)} \mathrm{eval}(T,t[\varphi]_\Pi) & \\
\mathrm{eval}(\mathsf{T}(R,T),t) & =_\mu & \bigcup_{u\in\mathrm{app}(R,t)} \mathrm{rewr}(u) & \text{if } \mathrm{app}(R,t)\neq\emptyset \\
\mathrm{eval}(\mathsf{T}(R,T),t) & =_\mu & \mathrm{eval}(T,t) & \text{if } \mathrm{app}(R,t)=\emptyset \\
\mathrm{eval}(\mathsf{E},t) & =_\mu & \{t\} & \\
\mathrm{eval}(\mathsf{X},t) & =_\mu & \emptyset &
\end{array}
$$

$$
\begin{array}{lcll}
\mathrm{eval_h}(\mathsf{F}(\pi,\varphi),t) & =_\mu & \mathrm{eval_h}(\varphi(\mathrm{hs}(t|_\pi)),t) & \text{if } \pi \in \mathrm{pos_f}(t) \\
\mathrm{eval_h}(\mathsf{F}(\pi,\varphi),t) & =_\mu & \mathrm{eval_h}(\varphi(\bot),t) & \text{if } \pi \notin \mathrm{pos_f}(t) \\
\mathrm{eval_h}(\mathsf{H}(\Pi,T),t) & =_\mu & \bigcup_{\varphi\in\mathrm{rewrf_h}(t,\Pi)} \mathrm{eval_h}(T,t[\varphi]_\Pi) & \\
\mathrm{eval_h}(\mathsf{NF}(\Pi,T),t) & =_\mu & \bigcup_{\varphi\in\mathrm{rewrf}(t,\Pi)} \mathrm{eval_h}(T,t[\varphi]_\Pi) & \\
\mathrm{eval_h}(\mathsf{T}(R,T),t) & =_\mu & \bigcup_{u\in\mathrm{app}(R,t)} \mathrm{rewr_h}(u) & \text{if } \mathrm{app}(R,t)\neq\emptyset \\
\mathrm{eval_h}(\mathsf{T}(R,T),t) & =_\mu & \mathrm{eval_h}(T,t) & \text{if } \mathrm{app}(R,t)=\emptyset \\
\mathrm{eval_h}(\mathsf{E},t) & =_\mu & \{t\} & \\
\mathrm{eval_h}(\mathsf{X},t) & =_\mu & \emptyset &
\end{array}
$$

with auxiliary functions $\mathrm{app} : \mathcal{P}(\mathbb{R}) \times \mathbb{T} \to \mathcal{P}(\mathbb{T})$ and $\mathrm{rewrf}, \mathrm{rewrf_h} : \mathbb{T} \times \mathcal{P}(\mathbb{P}) \to \mathcal{P}(\mathbb{P} \to \mathbb{T})$ defined as follows:

$$
\mathrm{app}(R,t) = \{u \ : \ l \to r \ \mathbf{if}\ c \in R \ \wedge\ t = l\sigma \ \wedge\ u = r\sigma \ \wedge\ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}
$$

$$
\begin{array}{lcl}
\mathrm{rewrf}(t,\Pi) & = & \{\varphi \ : \ \forall_{\pi\in\Pi}(\pi \in \mathrm{pos}(t) \ \Rightarrow\ \varphi(\pi) \in \mathrm{rewr}(t|_\pi))\} \\
\mathrm{rewrf_h}(t,\Pi) & = & \{\varphi \ : \ \forall_{\pi\in\Pi}(\pi \in \mathrm{pos}(t) \ \Rightarrow\ \varphi(\pi) \in \mathrm{rewr_h}(t|_\pi))\}
\end{array}
$$

This definition is sound in the sense that rewr and $\mathrm{rewr_h}$ only return terms that can be obtained from the input by rewriting.

Theorem 5.2.1 *For all terms t and u such that $u \in \mathrm{rewr}(t)$, we have that $t \to^* u$. Similarly, for all terms t and u such that $u \in \mathrm{rewr_h}(t)$, we have that $t \to^* u$.*

The following theorem shows that we can easily translate sequential strategies to strategy trees while preserving their functionality.

Theorem 5.2.2 *Let φ, the translation function of sequential strategies to strategy trees, be defined as follows.*

$$
\begin{array}{lcl}
\varphi([]) & = & \mathsf{E} \\
\varphi(I \rhd s) & = & \mathsf{NF}(I,\varphi(s)) \\
\varphi(R \rhd s) & = & \mathsf{T}(R,\varphi(s))
\end{array}
$$

If ς is a sequential-strategy function, then we have that $\mathrm{rewr}(t)$ *for strategy $\langle \varsigma_\varphi, \varsigma_\mathrm{h}\rangle$ is equivalent to* $\mathrm{rewr_s}(t)$ *for strategy ς, where for all function symbols f*

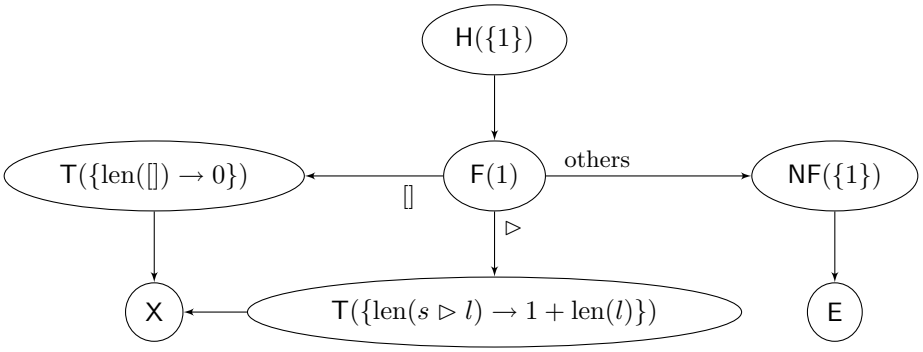


Figure 5.1: Strategy tree for len

$\varsigma_\varphi(f) = \varphi(\varsigma(f))$ *and* ς_h *is an arbitrary function.*

We illustrate some strategy trees in the following examples. Here we draw strategy trees such that each node is represented by an ellipse containing the node without subtrees (e.g. $\mathsf{NF}(\Pi)$ for node $\mathsf{NF}(\Pi, T)$) and there is an arrow from the ellipse for node T and the ellipse for U if U is a direct subtree of T. In the case of an $\mathsf{F}(\pi, \varphi)$ tree we consider all $\varphi(f^\perp)$ trees as subtrees and label the arrows with the corresponding $f^\perp$. Here we often draw one subtree with incoming arrow "others" for all equivalent subtrees that are not already separately drawn.

Example 5.2.3 A strategy tree for len is the following (as depicted in Figure 5.1).

$$\mathsf{H}(\{1\}, \mathsf{F}(1, \varphi))$$

Here φ is defined as follows.

$$\begin{array}{lcll} \varphi([]) & = & \mathsf{T}(\{\mathrm{len}([]) \to 0\}, \mathsf{X}) & \\ \varphi(\triangleright) & = & \mathsf{T}(\{\mathrm{len}(s \triangleright l) \to 1 + \mathrm{len}(l)\}, \mathsf{X}) & \\ \varphi(f^\perp) & = & \mathsf{NF}(\{1\}, \mathsf{E}) & \text{if } f^\perp \notin \{[], \triangleright\} \end{array}$$

This strategy corresponds to the one discussed in Section 5.1.

Example 5.2.4 In Figure 5.2 a strategy tree is depicted that corresponds to the innermost strategy for a function symbol f of arity n and with rewrite rules R_f.

Example 5.2.5 In Figure 5.3 a depiction is given of the translation (see Theorem 5.2.2) of the sequential strategy for if. In Figure 5.4 an alternative strategy tree is given that uses the extra possibilities of strategy trees. Here we use a H/F combination to avoid unnecessary attempts to apply rules $\mathrm{if}(\mathrm{true}, x, y) \to x$ and $\mathrm{if}(\mathrm{false}, x, y) \to y$.

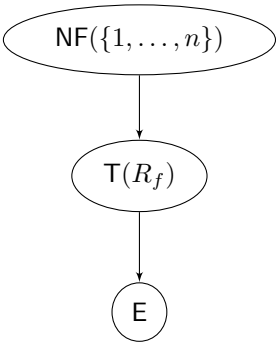


Figure 5.2: Strategy tree for innermost rewriting

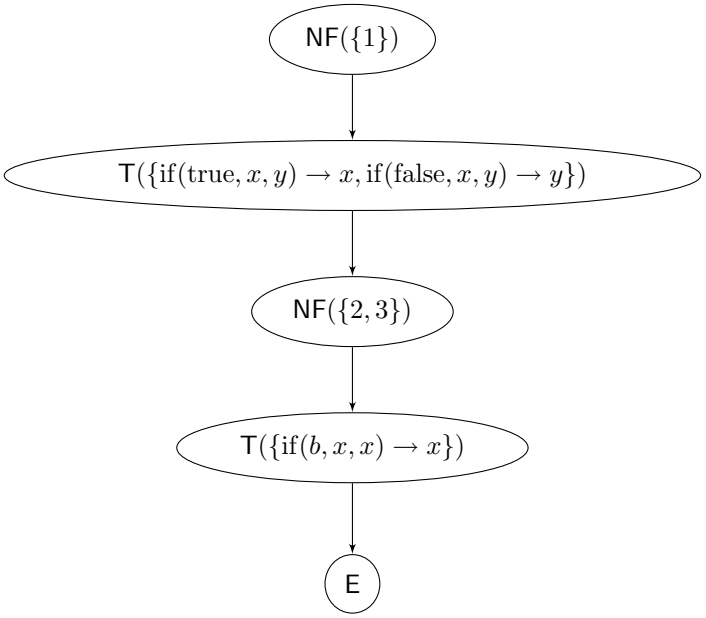


Figure 5.3: Strategy tree for if (sequential)

Strategy Tree Manipulation

In certain cases it is useful to be able to manipulate strategy trees while preserving their behaviour to some extent. For example, in the implementation of strategy trees one typically wants only a single normal form for a term, which can be achieved by trying rewrite rules one at a time instead of a set of rules at a time.

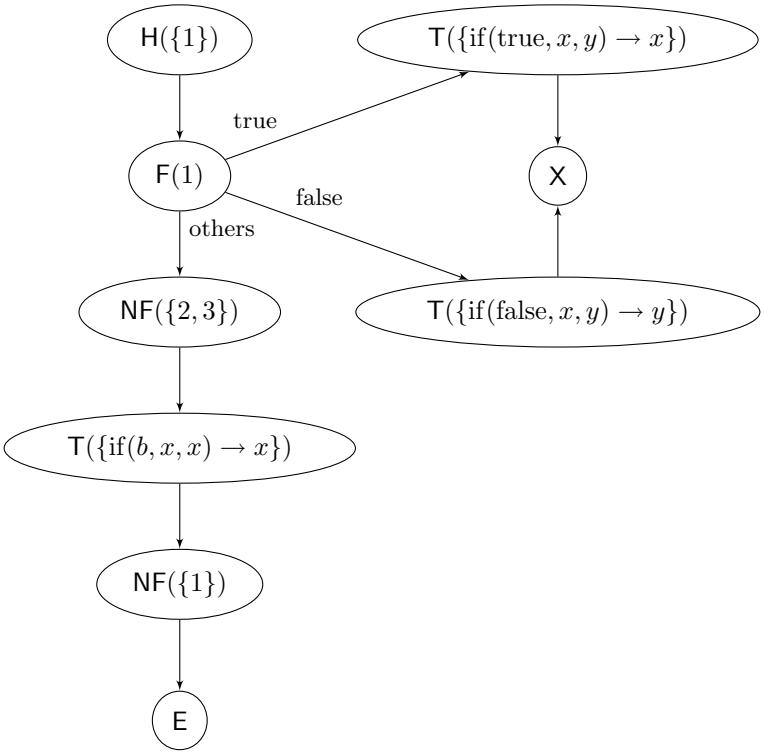


Figure 5.4: Strategy tree for if

Also, as we can see in Section 5.4, automatic generation of strategy trees can result in trees that are larger than strictly necessary and one might wish to simplify such trees.

First we specify what we mean with preservation of strategy tree behaviour. The semantics of strategy trees induces an equivalence on trees; strategy trees that result in the same set of normal forms for each term can be considered equivalent. For example, the trees $\mathsf{NF}(\{1,2\},T)$ and $\mathsf{NF}(\{1\},\mathsf{NF}(\{2\},T))$ have the same semantics. By assuming an order on sets of terms, we also easily get an order on strategy trees.

We define the following order on sets of terms and extend it to strategy trees. This order is chosen to reflect the amount of non-determinism in trees.

Definition $\leq$. We define the order $\leq \subseteq \mathcal{P}(\mathbb{T}) \times \mathcal{P}(\mathbb{T})$ as follows.

$$\begin{array}{lcll} \emptyset & \leq & \emptyset & \\ S & \leq & S \cup T & \text{if } S \neq \emptyset \end{array}$$

Definition $\leq, \leq_{\mathrm{h}}$**.** We define the orders $\leq, \leq_{\mathrm{h}} \subseteq \mathbb{ST} \times \mathbb{ST}$ as follows. For all strategy trees T and U, $T \leq U$ holds if, and only if, for all TRSs $\langle \Sigma, \rightarrow \rangle \eta$, terms t and strategies $\langle \varsigma, \varsigma_{\mathrm{h}} \rangle$, it holds that $\mathrm{eval}(T, t) \leq \mathrm{eval}(U, t)$. Similarly, for all strategy trees T and U, $T \leq_{\mathrm{h}} U$ holds if, and only if, for all TRSs $\langle \Sigma, \rightarrow \rangle \eta$, terms t and strategies $\langle \varsigma, \varsigma_{\mathrm{h}} \rangle$, it holds that $\mathrm{eval}_{\mathrm{h}}(T, t) \leq \mathrm{eval}_{\mathrm{h}}(U, t)$. We say two strategy trees T and U are equivalent (modulo eval) if $T \leq U \wedge U \leq T$ and equivalent (modulo $\mathrm{eval}_{\mathrm{h}}$) if $T \leq_{\mathrm{h}} U \wedge U \leq_{\mathrm{h}} T$. We write these equivalences as $=$ and $=_{\mathrm{h}}$ respectively.

Conjecture 5.2.6 *Let $\langle \varsigma, \varsigma_{\mathrm{h}} \rangle$ and $\langle \varsigma', \varsigma'_{\mathrm{h}} \rangle$ be strategies and t a term. If for all f we have that $\varsigma(f) \leq \varsigma'(f)$ and $\varsigma_{\mathrm{h}}(f) \leq \varsigma'_{\mathrm{h}}(f)$, $S = \mathrm{rewr}(t)$ using $\langle \varsigma, \varsigma_{\mathrm{h}} \rangle$ and $S' = \mathrm{rewr}(t)$ using $\langle \varsigma', \varsigma'_{\mathrm{h}} \rangle$, then $S \leq S'$.*

We have the following (in)equalities. These (in)equalities are mainly given as a means to eliminate non-determinism in the implementation of strategy trees. Note that these have not been proven to be sound.

Property 5.2.7

$$
\begin{array}{lcll}
\mathsf{F}(\pi, \varphi[f^{\perp} \mapsto \mathsf{F}(\pi, \psi)]) & = & \mathsf{F}(\pi, \varphi[f^{\perp} \mapsto \psi(f)]) & \\
\mathsf{F}(\pi, \varphi[f^{\perp} \mapsto \mathsf{F}(\pi, \psi)]) & =_h & \mathsf{F}(\pi, \varphi[f^{\perp} \mapsto \psi(f)]) & \\
\mathsf{H}(\Pi, \mathsf{H}(\Pi', T)) & = & \mathsf{H}(\Pi \cup \Pi', T) & \textit{if } \overline{\Pi} \cap \overline{\Pi'} = \emptyset \\
\mathsf{H}(\Pi, \mathsf{H}(\Pi', T)) & =_h & \mathsf{H}(\Pi \cup \Pi', T) & \textit{if } \overline{\Pi} \cap \overline{\Pi'} = \emptyset \\
\mathsf{NF}(\Pi, \mathsf{NF}(\Pi', T)) & = & \mathsf{NF}(\Pi \cup \Pi', T) & \textit{if } \overline{\Pi} \cap \overline{\Pi'} = \emptyset \\
\mathsf{NF}(\Pi, \mathsf{NF}(\Pi', T)) & =_h & \mathsf{NF}(\Pi \cup \Pi', T) & \textit{if } \overline{\Pi} \cap \overline{\Pi'} = \emptyset \\
\mathsf{T}(R, \mathsf{T}(R', T)) & \leq & \mathsf{T}(R \cup R', T) & \\
\mathsf{T}(R, \mathsf{T}(R', T)) & \leq_h & \mathsf{T}(R \cup R', T) &
\end{array}
$$

Theorem 5.2.8 *Let $\langle \varsigma, \varsigma_{\mathrm{h}} \rangle$ be a strategy. If all T nodes of all strategy trees $\varsigma(f)$ and $\varsigma_{\mathrm{h}}(f)$, for all function symbols f, have a set with at most one rewrite rule, then $|\mathrm{rewr}(t)| \leq 1$ and $|\mathrm{rewr}_{\mathrm{h}}(t)| \leq 1$ for all t.*

5.3 Normalisation

Of course, we also wish that rewr actually returns normal forms and that if it doesn't, that there is a reason for it (i.e. that there is an infinite rewrite sequence). That this isn't the case in general is obvious if one considers, for example, the strategy $\langle \varsigma, \varsigma_{\mathrm{h}} \rangle$ where $\varsigma(f) = \mathsf{E}$ for all f. Like with the sequential strategies, we need a property on strategies that guarantees that they do what we want.

Below we define what it means to be **thorough** for both strategy trees and strategies. Conceptually one can think of this property as saying that when one

"arrives" at an E or X node, one is certain that there is no use in trying to apply any rewrite rule (directly with T or indirectly with H or NF). In other words, when arriving at an E one wants that the term that is being rewritten is already in normal form. Similarly, when arriving at an X one wants that such a term has an infinite rewrite sequence.

To express this property we use three auxiliary sets. One set (S) is a set of terms that contains those terms that a tree can be applied to (in the given context). For example, if $T = \mathsf{T}(\{f(c) \to t\}, U)$ is a strategy tree that can be applied to the terms in S, then we have that U will only be applied to those elements in S that do not match $f(c)$.

A set of rewrite rules (R) is maintained to collect those rewrite rules we are certain of that they do not match terms in S. Finally, a set Π of positions is maintained to collect those positions of terms in S that are known to be in head normal form (i.e. if a position in Π is a valid position in a term $t \in S$, then $t|_\pi$ is in head normal form).

Note that we use the notion of head normal form instead of stable-head form. The reason for this is that we know that if all subterms of a term are in head normal form, then the term itself is in normal form (Theorem 2.1.1). For stable-head forms we do not have such a theorem. We believe that it is possible to also define a notion of thorough that uses stable-head forms instead of head normal forms but have not investigated this. We expect that in practice the H construct is mainly used to filter out constructor functions (e.g. [] and $\triangleright$), for which head normal form and stable-head form coincide, and that if no such stable head symbol can be found the whole (sub)term needs to be normalised.

Definition thrgh, thrgh_h**.** We define that a tree T is **thorough** with respect to a function symbol f, set of terms S, set of rewrite rules R and set of positions Π, notation $\text{thrgh}(T, S, R, \Pi)$, in the following way.

$$
\begin{aligned}
\text{thrgh}(\mathsf{F}(\pi,\psi), S, R, \Pi) \quad = \quad & \forall_{f'}(\text{thrgh}(\psi(f'), \\
& \quad \{t \in S \ : \ \pi \in \text{pos}_\text{f}(t) \ \wedge \ \text{hs}(t|_\pi) = f'\}, \\
& \quad R \cup \{l \to r \ \textbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \wedge \\
& \qquad \text{hs}(l\sigma|_\pi) = f')\}, \\
& \quad \Pi)) \wedge \\
& \text{thrgh}(\psi(\bot), \{t \in S \ : \ \pi \notin \text{pos}_\text{f}(t)\}, \\
& \quad R \cup \{l \to r \ \textbf{if} \ c \in R_f \ : \ \pi \in \text{pos}_\text{f}(l)\}, \Pi \cup \{\pi\}) \\
\text{thrgh}(\mathsf{H}(\Pi', T), S, R, \Pi) \quad = \quad & \text{thrgh}(T, \\
& \quad \{t[\psi]_{\Pi'} \ : \ t \in S \wedge \\
& \qquad \forall_{\pi \in \Pi'}(\pi \in \text{pos}(t) \ \Rightarrow \ \psi(\pi) \in \text{rewr}_\text{h}(t|_\pi))\}, \\
& \quad \{\rho \in R \ : \ \forall_{t \in S}(\partial(t,\rho) \setminus \Pi' \neq \emptyset) \vee \\
& \qquad \Pi' \cap \text{esspos}(\rho) \subseteq \Pi\}, \\
& \quad (\Pi \cup \Pi') \setminus \{\pi \cdot i \cdot \pi' \ : \ \pi \in \Pi'\})
\end{aligned}
$$

(continued on next page)

$$
\begin{array}{lcl}
\mathrm{thrgh}(\mathsf{NF}(\Pi', T), S, R, \Pi) & = & \mathrm{thrgh}(T, \\
 & & \quad \{t[\psi]_{\Pi'} \ : \ t \in S \wedge \\
 & & \quad\quad \forall_{\pi \in \Pi'}(\pi \in \mathrm{pos}(t) \ \Rightarrow \ \psi(\pi) \in \mathrm{rewr}(t|_\pi))\}, \\
 & & \quad \{\rho \in R \ : \ \forall_{t \in S}(\partial(t, \rho) \setminus \overline{\Pi'} \neq \emptyset) \vee \\
 & & \quad\quad \overline{\Pi'} \cap \mathrm{esspos}(\rho) \subseteq \Pi\}, \\
 & & \quad \Pi \cup \overline{\Pi'}) \\
\mathrm{thrgh}(\mathsf{T}(R', T), S, R, \Pi) & = & \mathrm{thrgh}(T, \\
 & & \quad S \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}, \\
 & & \quad R \cup R', \Pi) \\
\mathrm{thrgh}(\mathsf{E}, S, R, \Pi) & = & \forall_{t \in S}(\mathrm{pos}(t) \subseteq \Pi \cup \{\epsilon\}) \ \wedge \ (S \neq \emptyset \ \Rightarrow \ R_f \subseteq R) \\
\mathrm{thrgh}(\mathsf{X}, S, R, \Pi) & = & \forall_{t \in S}(t \to^\omega)
\end{array}
$$

We define that a tree T is **head thorough** with respect to a function symbol f, set of terms S, set of rewrite rules R and set of positions Π, notation $\mathrm{thrgh}_\mathrm{h}(T, S, R, \Pi)$, in the following way.

$$
\begin{array}{lcl}
\mathrm{thrgh}_\mathrm{h}(\mathsf{F}(\pi, \psi), S, R, \Pi) & = & \forall_{f'}(\mathrm{thrgh}_\mathrm{h}(\psi(f'), \\
 & & \quad \{t \in S \ : \ \pi \in \mathrm{pos}_\mathrm{f}(t) \ \wedge \ \mathrm{hs}(t|_\pi) = f'\}, \\
 & & \quad R \cup \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \mathrm{pos}_\mathrm{f}(l\sigma) \wedge \\
 & & \quad\quad \mathrm{hs}(l\sigma|_\pi) = f')\}, \\
 & & \quad \Pi)) \wedge \\
 & & \mathrm{thrgh}_\mathrm{h}(\psi(\bot), \{t \in S \ : \ \pi \notin \mathrm{pos}_\mathrm{f}(t)\}, \\
 & & \quad R \cup \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \pi \in \mathrm{pos}_\mathrm{f}(l)\}, \Pi \cup \{\pi\}) \\
\mathrm{thrgh}_\mathrm{h}(\mathsf{H}(\Pi', T), S, R, \Pi) & = & \mathrm{thrgh}_\mathrm{h}(T, \\
 & & \quad \{t[\psi]_{\Pi'} \ : \ t \in S \wedge \\
 & & \quad\quad \forall_{\pi \in \Pi'}(\pi \in \mathrm{pos}(t) \ \Rightarrow \ \psi(\pi) \in \mathrm{rewr}_\mathrm{h}(t|_\pi))\}, \\
 & & \quad \{\rho \in R \ : \ \forall_{t \in S}(\partial(t, \rho) \setminus \Pi' \neq \emptyset) \vee \\
 & & \quad\quad \Pi' \cap \mathrm{esspos}(\rho) \subseteq \Pi\}, \\
 & & \quad (\Pi \cup \Pi') \setminus \{\pi \cdot i \cdot \pi' \ : \ \pi \in \Pi'\}) \\
\mathrm{thrgh}_\mathrm{h}(\mathsf{NF}(\Pi', T), S, R, \Pi) & = & \mathrm{thrgh}_\mathrm{h}(T, \\
 & & \quad \{t[\psi]_{\Pi'} \ : \ t \in S \wedge \\
 & & \quad\quad \forall_{\pi \in \Pi'}(\pi \in \mathrm{pos}(t) \ \Rightarrow \ \psi(\pi) \in \mathrm{rewr}(t|_\pi))\}, \\
 & & \quad \{\rho \in R \ : \ \forall_{t \in S}(\partial(t, \rho) \setminus \overline{\Pi'} \neq \emptyset) \vee \\
 & & \quad\quad \overline{\Pi'} \cap \mathrm{esspos}(\rho) \subseteq \Pi\}, \quad \Pi \cup \overline{\Pi'}) \\
\mathrm{thrgh}_\mathrm{h}(\mathsf{T}(R', T), S, R, \Pi) & = & \mathrm{thrgh}_\mathrm{h}(T, \\
 & & \quad S \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}, \\
 & & \quad R \cup R', \Pi) \\
\mathrm{thrgh}_\mathrm{h}(\mathsf{E}, S, R, \Pi) & = & \forall_{t \in S, l \to r \ \mathbf{if} \ c \in R_f}(\partial(t, l) \cap \Pi \neq \emptyset \ \vee \ (\partial(t, l) = \emptyset \wedge \\
 & & \quad\quad \mathrm{esspos}(l \to r \ \mathbf{if} \ c) \subseteq \Pi \wedge \\
 & & \quad \neg\exists_\sigma(t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)))) \wedge \\
 & & (S \neq \emptyset \ \Rightarrow \ R_f \subseteq R) \\
\mathrm{thrgh}_\mathrm{h}(\mathsf{X}, S, R, \Pi) & = & \forall_{t \in S}(t \to^\omega)
\end{array}
$$

We say a strategy $\langle \varsigma, \varsigma_{\mathrm{h}} \rangle$ is **thorough** if all strategy trees $\varsigma(f)$ and $\varsigma_{\mathrm{h}}(f)$ are thorough, respectively head thorough with respect to function symbol f, set of terms $\{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\}$, set of rewrite rules $\emptyset$ and set of positions $\emptyset$. That is,

$$\begin{aligned} \mathrm{thrgh}(\langle \varsigma, \varsigma_{\mathrm{h}} \rangle) = \quad & \forall_f(\mathrm{thrgh}(\varsigma(f), \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\}, \emptyset, \emptyset) \wedge \\ & \mathrm{thrgh}_{\mathrm{h}}(\varsigma_{\mathrm{h}}(f), \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\}, \emptyset, \emptyset)) \ . \end{aligned}$$

Theorem 5.3.1 *Assume a strategy* $\langle \varsigma, \varsigma_{\mathrm{h}} \rangle$ *that is thorough. We then have that* $\mathrm{rewr}(t) \subseteq \mathrm{nf}(t)$ *and* $\mathrm{rewr}_{\mathrm{h}}(t) \subseteq \mathrm{hnf}(t)$ *for all terms* t*.*

Theorem 5.3.2 *Assume a strategy* $\langle \varsigma, \varsigma_{\mathrm{h}} \rangle$ *that is thorough. We then have that* $\mathrm{rewr}(t) = \emptyset \ \Rightarrow \ t \rightarrow^{\omega}$ *and* $\mathrm{rewr}_{\mathrm{h}}(t) = \emptyset \ \Rightarrow \ t \rightarrow^{\omega}$ *for all terms* t*.*

Note that if one has a thorough strategy and true itself is a normal form, then we trivially have that $\mathrm{true} \in \mathrm{rewr}(t)$ is equivalent to $\mathrm{true} \in \mathrm{rewr}_{\mathrm{h}}(t)$ for all terms t. This means that we can safely use $\mathrm{rewr}_{\mathrm{h}}$ for condition evaluation in such a context. This can avoid a lot of unnecessary rewriting if a condition does not rewrite to true (or, for example, false).

Example 5.3.3 The strategy trees of Example 5.2.3, Example 5.2.4 and Example 5.2.5 are all thorough. Below is the derivation that shows this for the strategy tree for len.

$$\begin{aligned} & \mathrm{thrgh}(\mathsf{H}(\{1\}, \mathsf{F}(1, \varphi)), \{\mathrm{len}(l) \ : \ l \in \mathbb{T}\}, \emptyset, \emptyset) \\ = \ & \\ & \mathrm{thrgh}(\mathsf{F}(1, \varphi), \{\mathrm{len}(l) \ : \ l \in \mathbb{T} \ \wedge \ l \in \mathrm{rewr}_{\mathrm{h}}(l)\}, \emptyset, \{1\}) \\ = \ & \\ & \mathrm{thrgh}(\mathsf{T}(\{\mathrm{len}([]) \rightarrow 0\}, \mathsf{X}), \\ & \quad \{\mathrm{len}([]) \ : \ [] \in \mathrm{rewr}_{\mathrm{h}}([])\}, \{\mathrm{len}(t \rhd l) \rightarrow 1 + \mathrm{len}(l)\}, \{1\}) \wedge \\ & \mathrm{thrgh}(\mathsf{T}(\{\mathrm{len}(t \rhd l) \rightarrow 1 + \mathrm{len}(l)\}, \mathsf{X}), \\ & \quad \{\mathrm{len}(e \rhd l) \ : \ e, l \in \mathbb{T} \ \wedge \ e \rhd l \in \mathrm{rewr}_{\mathrm{h}}(e \rhd l)\}, \{\mathrm{len}([]) \rightarrow 0\}, \{1\}) \wedge \\ & \forall_{f \notin \{[], \rhd\}}(\mathrm{thrgh}(\mathsf{NF}(\{1\}, \mathsf{E}), \\ & \qquad \{\mathrm{len}(l) \ : \ l \in \mathbb{T} \ \wedge \ l \in \mathrm{rewr}_{\mathrm{h}}(l) \ \wedge \ \epsilon \in \mathrm{pos}_{\mathrm{f}}(l) \ \wedge \ \mathrm{hs}(l) = f\}, \\ & \qquad R_{\mathrm{len}}, \{1\})) \wedge \\ & \mathrm{thrgh}(\mathsf{NF}(\{1\}, \mathsf{E}), \{\mathrm{len}(x) \ : \ x \in \mathbb{V}\}, R_{\mathrm{len}}, \{1\}) \\ = \ & \end{aligned}$$

$$
\begin{aligned}
&\mathrm{thrgh}(\mathsf{X}, \emptyset, R_{\mathrm{len}}, \{1\}) \wedge \\
&\mathrm{thrgh}(\mathsf{X}, \emptyset, R_{\mathrm{len}}, \{1\}) \wedge \\
&\forall_{f^{\perp} \notin \{[], \triangleright\}}(\mathrm{thrgh}(\mathsf{E}, \{\mathrm{len}(l) \ : \ l \in \mathbb{T} \ \wedge \ l \in \mathrm{rewr}(l)\}, R_{\mathrm{len}}, \{1 \cdot \pi \ : \ \mathrm{true}\})) \wedge \\
&\mathrm{thrgh}(\mathsf{E}, \{\mathrm{len}(x) \ : \ x \in \mathbb{V}\}, R_{\mathrm{len}}, \{1 \cdot \pi \ : \ \mathrm{true}\}) \\
=& \\
&\forall_{t \in \emptyset}(t \to^{\omega}) \wedge \\
&\forall_{t \in \emptyset}(t \to^{\omega}) \wedge \\
&\forall_{f^{\perp} \notin \{[], \triangleright\}}(\forall_{t \in \{\mathrm{len}(l) \, : \, l \in \mathbb{T} \wedge l \in \mathrm{rewr}(l)\}}(\mathrm{pos}(t) \subseteq \{1 \cdot \pi \ : \ \mathrm{true}\} \cup \{\epsilon\}) \wedge \\
&\qquad\qquad R_{\mathrm{len}} \subseteq R_{\mathrm{len}}) \wedge \\
&\forall_{t \in \{\mathrm{len}(x) \, : \, x \in \mathbb{V}\}}(\mathrm{pos}(t) \subseteq \{1 \cdot \pi \ : \ \mathrm{true}\} \cup \{\epsilon\}) \ \wedge \ R_{\mathrm{len}} \subseteq R_{\mathrm{len}} \\
=& \\
&\mathrm{true}
\end{aligned}
$$

We have that the notions full and in-time of sequential strategies are included in the notion of thoroughness.

Theorem 5.3.4 *Let ς be a sequential strategy that is full and in-time. We have that the strategy $\langle \varsigma_{\varphi}, \varsigma_{\mathrm{h}} \rangle$ as given by the translation in Theorem 5.2.2 is thorough.*

5.4 Strategy Generation

Until now we have only discussed the semantics of strategy trees. For practical applications a relevant question is how one obtains a strategy (given a set of rewrite rules). Many different approaches can be used. Similar to what we have done in Section 2.1 for sequential strategies, we wish to give a straightforward method for constructing strategy trees.

For sequential strategies the construction of strategies is guided by the neededness of arguments of the head symbol. Those which are needed by most rewrite rules are rewritten first. We extend this to strategy trees by considering the neededness of symbols at positions in the term to be rewritten. For example, for the rule $f(g(x), y, h(z, c)) \to t$ we can say that it needs the symbols g, h and c at positions 1, 3 and $3 \cdot 2$, respectively.

Another form of neededness is that of multiple occurrences of variables at the left-hand side or the right-hand side or the use of variables in conditions. Unlike with sequential strategy generation, we must differentiate these two types of neededness. The first requires only a rewrite to stable-head form and a check for the relevant symbols. The other form requires a rewrite to full normal form but no checks.

First we introduce some auxiliary functions. The functions $\mathrm{need}_{\mathrm{f}}$ and $\mathrm{need}_{\mathrm{v}}$ give the needed positions according to the above explanation.

$$\begin{aligned}
\mathrm{need_f}(l \to r \textbf{ if } c) &= \mathrm{pos_f}(l) \\
\mathrm{need_v}(l \to r \textbf{ if } c) &= \{\pi \in \mathrm{pos_v}(l) \ : \ \exists_{\pi' \neq \pi}(l|_{\pi'} = l|_\pi) \ \vee \ l|_\pi \in \mathrm{var}(c) \vee \\
&\qquad \exists_{\pi_1,\pi_2}(\pi_1 \neq \pi_2 \ \wedge \ r|_{\pi_1} = l|_\pi = r|_{\pi_2})\}
\end{aligned}$$

Note that $\mathrm{need_f}(\rho) \cup \overline{\mathrm{need_v}(\rho)}$ is a superset of $\mathrm{esspos}(\rho)$ for all rewrite rules ρ.

Next we define auxiliary functions that will determine which activity should be "done" first in a strategy tree based on a set R of rewrite rules that still need to be tried and a set Π of positions we have not rewritten to (head) normal form. The function stready : $\mathcal{P}(\mathbb{R}) \times \mathcal{P}(\mathbb{P}) \to \mathcal{P}(\mathbb{R})$ returns those rewrite rules of which all needed positions have been rewritten to (head) normal form.

$$\mathrm{stready}(R, \Pi) = \{\rho \ : \ \rho \in R \ \wedge \ \overline{\Pi} \cap (\mathrm{need_f}(\rho) \cup \overline{\mathrm{need_v}(\rho)}) = \emptyset\}$$

The functions $\mathrm{need_f^w}, \mathrm{need_v^w} : \mathcal{P}(\mathbb{R}) \times \mathcal{P}(\mathbb{P}) \to \mathcal{P}(\mathbb{P})$ return those positions that have a maximum weight according to weight function $w : \mathcal{P}(\mathbb{R}) \times \mathcal{P}(\mathbb{P}) \times (\mathbb{R} \to \mathcal{P}(\mathbb{P})) \to \mathcal{P}(\mathbb{P}) \times O$ with $\{\pi \ : \ \langle \pi, r\rangle \in w(R, \Pi, \varphi)\} \subseteq \Pi$ and O a complete lattice. This weight function is used to express what it means for a position to be needed by most rewrite rules. Here the first two arguments specify which rewrite rules and positions have to be taken into consideration and the third argument specifies a function that is to be used to determine which positions are needed for a single rule.

$$\begin{aligned}
\mathrm{need_f^w}(R, \Pi) &= \{\pi \ : \ \langle \pi, n\rangle \in w(R, \Pi, \mathrm{need_f}) \wedge \\
&\qquad n =\uparrow \{n' \ : \ \langle \pi', n'\rangle \in w(R, \Pi, \mathrm{need_f})\}\} \\
\mathrm{need_v^w}(R, \Pi) &= \{\pi \ : \ \langle \pi, n\rangle \in w(R, \Pi, \overline{\mathrm{need_v}}) \wedge \\
&\qquad n =\uparrow \{n' \ : \ \langle \pi', n'\rangle \in w(R, \Pi, \overline{\mathrm{need_v}})\}\}
\end{aligned}$$

Examples of weight functions are given below. Function w_1 corresponds to simply counting the number of rewrite rules that require a given position. Function w_2, on the other hand, also tries to take into account the amount of "progress" rewriting a given position leads to. For example, if one rule requires only position π and another requires different positions π' and π'', then w_1 will give all positions the same weight while w_2 will give π more weight then π' and π'' because rewriting π directly leads to being able to try a rewrite rule.

$$
\begin{array}{lcl}
w_1(R,\Pi,\varphi) & = & \{\langle \pi, n\rangle \ : \ \pi \in \Pi \ \wedge \ 0 < n = \sum_{\rho \in R, \pi \in \varphi(\rho)} 1\} \\
w_2(R,\Pi,\varphi) & = & \{\langle \pi, n\rangle \ : \ \pi \in \Pi \ \wedge \ 0 < n = \sum_{\rho \in R, \pi \in \varphi(\rho)} \frac{1}{|\varphi(\rho)|}\}
\end{array}
$$

A final auxiliary function filters a set of rewrite rules such that all remaining rules allow a given function symbol in a certain position. This function is used after the introduction of an F node to eliminate all rewrite rules that will never match in a given subtree.

$$
\begin{array}{lcl}
\mathrm{stfilter_f}(\pi, f, R) & = & \{l \to r \ \mathbf{if}\ c \in R \ : \ \exists_\sigma(\pi \in \mathrm{pos}(l\sigma) \ \wedge \ \mathrm{hs}(l\sigma|_\pi) = f)\} \\
\mathrm{stfilter_v}(\pi, R) & = & \{l \to r \ \mathbf{if}\ c \in R \ : \ \pi \notin \mathrm{pos_f}(l)\}
\end{array}
$$

Note that there is some improvement possible with respect to this filtering. For example a rule with pattern $f(x,x)$ will not be filtered out if it is determined (by means of F nodes) that the first argument starts with a symbol g and the second with a different symbol h. However, this requires one to keep track of the structure of the term that has been determined so far and it is not clear that doing so gives a significant effect in practice.

Our strategy generation function for normalising strategy trees, given a weight function w, is as follows (where $R \neq \emptyset$ and ι is some choice function).

$$
\begin{array}{lcll}
\mathrm{stgen}(R_f) & = & \mathrm{stgen}'(R_f, \{i \ : \ 1 \leq i \leq \mathrm{ar}(f)\}) & \\
 & & & \\
\mathrm{stgen}'(\emptyset, \Pi) & = & \mathsf{NF}(\Pi, \mathsf{E}) & \\
\mathrm{stgen}'(R, \Pi) & = & \mathsf{T}(R', \mathrm{stgen}'(R \setminus R', \Pi)) & \text{if } R' = \mathrm{stready}(R,\Pi) \neq \emptyset \\
\mathrm{stgen}'(R, \Pi) & = & \mathsf{H}(\{\pi\}, \mathsf{F}(\pi, \mathrm{stfunc}(\pi, R, \Pi))) & \text{if } \mathrm{stready}(R,\Pi) = \emptyset \ \wedge \\
 & & & \quad \mathrm{need}^w_{\mathrm{f}}(R,\Pi) \neq \emptyset \ \wedge \\
 & & & \quad \pi = \iota(\mathrm{need}^w_{\mathrm{f}}(R,\Pi)) \\
\mathrm{stgen}'(R, \Pi) & = & \mathsf{NF}(\{\pi\}, \mathrm{stgen}'(R, \Pi \setminus \{\pi\})) & \text{if } \mathrm{stready}(R,\Pi) = \emptyset \ \wedge \\
 & & & \quad \mathrm{need}^w_{\mathrm{f}}(R,\Pi) = \emptyset \ \wedge \\
 & & & \quad \pi = \iota(\mathrm{need}^w_{\mathrm{v}}(R,\Pi))
\end{array}
$$

where

$$
\begin{array}{lcl}
\mathrm{stfunc}(\pi, R, \Pi)(f) & = & \mathrm{stgen}'(\mathrm{stfilter_f}(\pi, f, R), \\
 & & \qquad (\Pi \setminus \{\pi\}) \cup \{\pi \cdot i \ : \ 1 \leq i \leq \mathrm{ar}(f)\}) \\
\mathrm{stfunc}(\pi, R, \Pi)(\bot) & = & \mathrm{stgen}'(\mathrm{stfilter_v}(\pi, R), \Pi \setminus \{\pi\})
\end{array}
$$

The strategy generation function for head normal form strategy trees (denoted by $\mathrm{stgen}_{\mathrm{h}}$ and with auxiliary function $\mathrm{stgen}'_{\mathrm{h}}$ and $\mathrm{stfunc}_{\mathrm{h}}$) is the same function except for the case $\mathrm{stgen}'_{\mathrm{h}}(\emptyset, \Pi)$, which is defined as E.

As long as applying stgen or $\mathrm{stgen}_{\mathrm{h}}$ to a set R_f does not result in a strategy tree with infinite depth, we have that the strategy tree is (head) thorough. This condition is guaranteed by requiring finiteness of the set of all positions that are valid for some left-hand side of a rule in R_f. Note that this formulation does allow infinite branching in strategy trees.

Theorem 5.4.1 *Let f be a function symbol such that $\bigcup_{l \to r \ \mathbf{if}\ c \in R_f} \mathrm{pos}(l)$ is finite. We have that $\mathrm{stgen}(R_f)$ is thorough w.r.t. f and, similarly, that $\mathrm{stgen}_{\mathrm{h}}(R_f)$ is head thorough w.r.t. f.*

Note that this strategy generation function can generate subtrees that will never be used as in the example below. This is due to the fact that it does not take into account that rewrite rules $l \to r$ **if** c with $\mathrm{esspos}(l \to r \ \mathbf{if}\ c) \cap \mathrm{pos}_{\mathrm{v}}(l) = \emptyset$ will always be applied when tried. Taking into account the structure of the term that the strategy has determined at a given point (as discussed above) would enable one to easily avoid such subtrees during construction. Do note, however, that these unreachable subtrees do not have a direct impact on performance (time wise).

Also note, that one might want to use the (in)equivalences from Section 5.2 to optimise the strategy trees if needed.

Example 5.4.2 We consider the function if again. We recall rewrite rules $\alpha = \mathrm{if}(\mathrm{true}, x, y) \to x$, $\beta = \mathrm{if}(\mathrm{false}, x, y) \to y$ and $\gamma = \mathrm{if}(b, x, x) \to x$. We use weight function w_1 (although the weight function does not have any effect on the actual results).

$$
\begin{aligned}
& \mathrm{stgen}(\{\alpha, \beta, \gamma\}) \\
= \quad & \\
& \mathrm{stgen}'(\{\alpha, \beta, \gamma\}, \{1,2,3\}) \\
= \quad & \{\ w^1(\{\alpha, \beta, \gamma\}, \{1,2,3\}) = \{\langle 1,2\rangle\}\ \} \\
& \mathsf{H}(\{1\}, \mathsf{F}(1, \mathrm{stfunc}(1, \{\alpha, \beta, \gamma\}, \{1,2,3\})))
\end{aligned}
$$

We continue with $\mathrm{stfunc}(1, \{\alpha, \beta, \gamma\}, \{1,2,3\})$ for the relevant cases. First for true:

$$
\begin{aligned}
& \mathrm{stfunc}(1, \{\alpha, \beta, \gamma\}, \{1,2,3\})(\mathrm{true}) \\
= \quad & \\
& \mathrm{stgen}'(\mathrm{stfilter}_{\mathrm{f}}(1, \mathrm{true}, \{\alpha, \beta, \gamma\}), \{2,3\}) \\
= \quad &
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{stgen}'(\{\alpha,\gamma\},\{2,3\}) \\
= \\
& \mathsf{T}(\{\alpha\},\mathrm{stgen}'(\{\gamma\},\{2,3\}))
\end{aligned}
$$

We have the following for $\mathrm{stgen}'(\{\gamma\},\{2,3\})$. Note that this is a subtree that will actually never be reached.

$$
\begin{aligned}
& \mathrm{stgen}'(\{\gamma\},\{2,3\}) \\
= \\
& \mathsf{NF}(\{2\},\mathrm{stgen}'(\{\gamma\},\{3\})) \\
= \\
& \mathsf{NF}(\{2\},\mathsf{NF}(\{3\},\mathrm{stgen}'(\{\gamma\},\emptyset))) \\
= \\
& \mathsf{NF}(\{2\},\mathsf{NF}(\{3\},\mathsf{T}(\{\gamma\},\mathrm{stgen}'(\emptyset,\emptyset)))) \\
= \\
& \mathsf{NF}(\{2\},\mathsf{NF}(\{3\},\mathsf{T}(\{\gamma\},\mathsf{NF}(\emptyset,\mathsf{E}))))
\end{aligned}
$$

Next for false we get a similar derivation:

$$
\begin{aligned}
& \mathrm{stfunc}(1,\{\alpha,\beta,\gamma\},\{1,2,3\})(\mathrm{false}) \\
= \\
& \mathrm{stgen}'(\mathrm{stfilter}_{\mathrm{f}}(1,\mathrm{false},\{\alpha,\beta,\gamma\}),\{2,3\}) \\
= \\
& \mathrm{stgen}'(\{\beta,\gamma\},\{2,3\}) \\
= \\
& \mathsf{T}(\{\beta\},\mathrm{stgen}'(\{\gamma\},\{2,3\}))
\end{aligned}
$$

And finally for arbitrary symbols different from true and false:

$$
\begin{aligned}
& \mathrm{stfunc}(1,\{\alpha,\beta,\gamma\},\{1,2,3\})(f) \\
= \\
& \mathrm{stgen}'(\mathrm{stfilter}_{\mathrm{f}}(1,f,\{\alpha,\beta,\gamma\}),\{1\cdot 1,\ldots,1\cdot \mathrm{ar}(f),2,3\}) \\
= \\
& \mathrm{stgen}'(\{\gamma\},\{1\cdot 1,\ldots,1\cdot \mathrm{ar}(f),2,3\}) \\
= \\
& \mathsf{NF}(\{2\},\mathrm{stgen}'(\{\gamma\},\{1\cdot 1,\ldots,1\cdot \mathrm{ar}(f),3\})) \\
= \\
& \mathsf{NF}(\{2\},\mathsf{NF}(\{3\},\mathrm{stgen}'(\{\gamma\},\{1\cdot 1,\ldots,1\cdot \mathrm{ar}(f)\}))) \\
=
\end{aligned}
$$

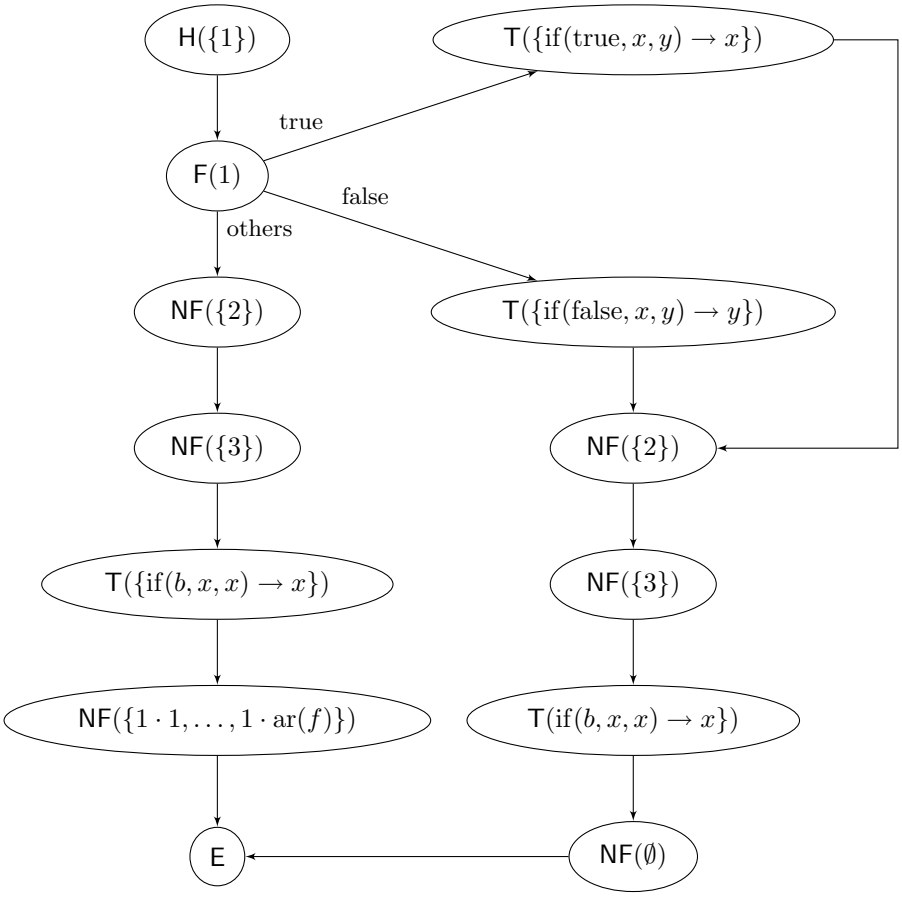


Figure 5.5: Generated strategy tree for if

$$\begin{aligned} & \mathsf{NF}(\{2\}, \mathsf{NF}(\{3\}, \mathsf{T}(\{\gamma\}, \mathrm{stgen}'(\emptyset, \{1 \cdot 1, \ldots, 1 \cdot \mathrm{ar}(f)\})))) \\ = \\ & \mathsf{NF}(\{2\}, \mathsf{NF}(\{3\}, \mathsf{T}(\{\gamma\}, \mathsf{NF}(\{1 \cdot 1, \ldots, 1 \cdot \mathrm{ar}(f)\}, \mathsf{E})))) \end{aligned}$$

The complete strategy tree is depicted in Figure 5.5.

5.5 Strategies and Matching

As strategy trees already make decisions based on the structure of terms, it would be useful to pass this information on to use during matching. For example, in

the example of len (Example 5.2.3) the strategy rewrites the argument to stable-head form and then tries the appropriate rewrite rule. As both rewrite rules only depend on the head symbol of the argument, one no longer has to check for it during matching as it is only called with matching terms.

In general, where it is possible that the strategy tree has not examined all positions esspos(ρ) of a term t before trying to apply rewrite rule ρ on t, we have a determined pattern p and substitution σ such that $t = p\sigma$. We would then like to use this p in generating the match tree for ρ and pass on σ when actually matching. That is, we are interested in the following extension of the function γ_1.

$$t = p\sigma \;\Rightarrow\; \mu'(\gamma_1(l \to r \;\mathbf{if}\; c, p), [t], \sigma) = \{r\tau \;:\; t = l\tau \;\wedge\; \text{true} \in \text{rewr}(c\tau)\}$$

Note that we can safely assume that $\text{var}(p) \cap \text{var}(l) = \emptyset$, as we construct p ourselves. With this, we propose the following definition of the extended γ_1 (where we use $\bot$ as a sort of "don't care").

$$\begin{array}{lcll}
\gamma_1(l \to r, t) & = & \gamma_1'([l], r, [t], \emptyset) & \\
 & & & \\
\gamma_1'([], r, [], V) & = & \mathsf{E}(r) & \\
\gamma_1'(x \rhd s, r, \bot \rhd s', V) & = & \mathsf{S}^1(x, \gamma_1'(s, r, s', V \cup \{x\})) & \text{if } x \notin V \\
\gamma_1'(x \rhd s, r, y \rhd s', V) & = & \mathsf{N}(\gamma_1'(s[y/x], r, s', V \cup \{y\})) & \text{if } x \notin V \\
\gamma_1'(x \rhd s, r, f(t_1, \ldots, t_n) \rhd s', V) & = & \mathsf{S}^1(x, \gamma_1'(s, r, s', V \cup \{x\})) & \text{if } x \notin V \\
\gamma_1'(x \rhd s, r, t \rhd s', V) & = & \mathsf{M}^1(x, \gamma_1'(s, r, s', V), \mathsf{X}) & \text{if } x \in V \\
\gamma_1'(f(p_1, \ldots, p_n) \rhd s, r, \bot \rhd s', V) & = & \mathsf{F}(f, \gamma_1'(p_1 \rhd \ldots \rhd p_n \rhd s, & \\
 & & \quad r, \bot \rhd \ldots \rhd \bot \rhd s', V), \mathsf{X}) & \\
\gamma_1'(f(p_1, \ldots, p_n) \rhd s, r, x \rhd s', V) & = & \mathsf{F}(f, \gamma_1'(p_1 \rhd \ldots \rhd p_n \rhd s, & \\
 & & \quad r, \bot \rhd \ldots \rhd \bot \rhd s', V), \mathsf{X}) & \\
\gamma_1'(f(p_1, \ldots, p_n) \rhd s, r, & & & \\
\quad f(t_1, \ldots, t_n) \rhd s', V) & = & \gamma_1'(p_1 \rhd \ldots \rhd p_n \rhd s, & \\
 & & \quad r, t_1 \rhd \ldots \rhd t_n \rhd s', V) & \\
\gamma_1'(f(p_1, \ldots, p_n) \rhd s, r, & & & \\
\quad g(t_1, \ldots, t_m) \rhd s', V) & = & \mathsf{X} &
\end{array}$$

Note that if $p\tau$ does not match l for any τ, then we can also simply take X as the match tree for this rule or even remove the rule from the strategy tree.

Conjecture 5.5.1

$$t = p\sigma \;\Rightarrow\; \mu'(\gamma_1(l \to r \;\mathbf{if}\; c, p), [t], \sigma) = \{r\tau \;:\; t = l\tau \;\wedge\; \text{true} \in \text{rewr}(c\tau)\}$$

There are also other options to improve the connection between strategy trees and matching. First of all, one could consider adding constructs such as S, M and

C to strategy trees to combine strategies with matching. The advantage of doing so would be that one can evaluate conditions or check the equivalence of subterms even sooner and continue with a strategy using the result of such checks.

Another option is to not use match trees but a method that is tailored to fit the used strategy trees. For example, if one uses the generated strategies of Section 5.4, then one knows that when a rewrite rule is tried on a term t, this term has the same structure as the pattern (looking only at function symbols). Thus, to complete matching, one only needs to check that certain subterms are equivalent and that the condition of the rewrite rule is satisfied.

Chapter 6

Evaluation

6.1 Introduction

We evaluate the techniques described in this thesis per chapter. First we consider the match trees from Chapter 3. In Section 6.2 we compare matching rules in a rule-by-rule way and matching using match trees by looking at the minimal, average and maximal number of steps (as indication of execution time) that are needed to find a match (or determine that there is no match).

In Section 6.3 we evaluate the temporary-term construction of Chapter 4. Here we look at the number of rewrite rules that are tested for a match, the number of rules that is actually applied, the number of calls to the rewriter and the number of term constructions that are needed to rewrite some specific examples. The same approach is taken for the evaluation of the strategy trees of Chapter 5 (generated according to Section 5.4) in Section 6.4. We also consider some previously obtained results (from [vW07] and some unpublished work) in Section 6.5.

6.2 Match Trees

We evaluate the match trees as generated by γ by comparing them with rule-by-rule matching. For the rule-by-rule matching we use the match trees for each of the separate rules (i.e. those generated with γ^1). The metrics we consider are the number of nodes the match trees have and the minimum, average and maximal path lengths through these trees (i.e. the number of "steps" in μ). The number of nodes gives an indication of the (static) memory requirements while the path metrics give an indication of execution times.

The *equality* test consists of rewrite rules to define an equality function on a sort with N constructor elements. For example, for $N = 2$ we have elements s_1

and s_2 and the following rewrite rules:

$$\begin{array}{lcl} eq(s_1, s_1) & \rightarrow & \text{true} \\ eq(s_1, s_2) & \rightarrow & \text{false} \\ eq(s_2, s_1) & \rightarrow & \text{false} \\ eq(s_2, s_2) & \rightarrow & \text{true} \end{array}$$

For test *fac* and *case* we have taken the following rules. The *case* test is a generalisation of the *if* using a sort with elements s_1, s_2, s_3 and s_4.

$$\begin{array}{lcl} fac(0) & \rightarrow & S(0) \\ fac(S(n) & \rightarrow & mult(S(n), fac(n)) \end{array}$$

$$\begin{array}{lcl} case(s_1, x_1, x_2, x_3, x_4) & \rightarrow & x_1 \\ case(s_2, x_1, x_2, x_3, x_4) & \rightarrow & x_2 \\ case(s_3, x_1, x_2, x_3, x_4) & \rightarrow & x_3 \\ case(s_4, x_1, x_2, x_3, x_4) & \rightarrow & x_4 \end{array}$$

As *case reversed* test we have taken the rules from the *case* test and reverse the arguments (i.e. elements s_i are in the fifth argument).

The *prioritised eq* and *(prioritised) fac* are taken from [BBKW89] (see Appendix F.1 and Appendix F.2). For *plus* and *ho plus* we have taken the rules from the introduction (with the arguments swapped) and from Example 3.3.2.

The results of our evaluation, as displayed in Table 6.1, show that using (combined) match trees is in most cases an improvement over rule-by-rule matching. The only metric where match trees sometimes perform worse is the minimal path metric. This is due to the fact that with rule-by-rule matching you might be lucky and be able to apply the first rule you try. However, the average path metric seems to suggest that this is well compensated for.

We can clearly see the effect of using (combined) match trees instead of rule-by-rule matching in the *equality* tests. With rule-by-rule matching there is quadratic growth in the number of nodes, average path and maximal path. This is obviously related to the quadratic growth in the number of rules. Also with the combined match trees there is a quadratic growth in the number of nodes. However, this only has effect on the amount of (static) memory. For time efficiency we can see that the maximal path grows only linearly and the average path even seems to be slightly sublinear.

Also, we see that application of reduce/clean can lead to significant reductions in all measures, even though there is no change for many cases.

6.3 Temporary-Term Construction

To evaluate the various techniques described in Chapter 4, we look at how these techniques influence the number of calls to the rewriter and the number of times

before reduce/clean

		rule-by-rule				combined			
	rules	nodes	min.	avg.	max.	nodes	min.	avg.	max.
equality ($N = 2$)	4	20	3	9.67	12	13	3	4.00	5
equality ($N = 3$)	9	45	3	22.16	27	25	3	5.15	7
equality ($N = 4$)	16	80	3	39.66	48	41	3	6.24	9
equality ($N = 5$)	25	125	3	62.16	75	61	3	7.29	11
equality ($N = 6$)	36	180	3	89.66	108	85	3	8.33	13
prioritised eq	2	11	5	7.00	9	10	7	7.00	7
case	4	44	8	12.45	16	41	5	10.20	13
case reversed	4	44	10	33.18	40	29	22	23.80	25
fac	2	8	2	4.00	6	7	3	3.33	4
prioritised fac	2	6	3	4.00	5	5	3	3.50	4
plus	2	12	4	5.33	8	11	3	4.67	6
ho plus	2	15	2	6.29	10	13	3	4.75	6

after reduce/clean

		rule-by-rule				combined			
	rules	nodes	min.	avg.	max.	nodes	min.	avg.	max.
prioritised eq	2	9	5	6.00	7	7	5	5.00	5
case	4	32	7	10.27	13	29	5	7.80	10
case reversed	4	32	7	23.23	28	17	10	11.80	13

(Test for which application of reduce and clean had no effect have not been included.)

Table 6.1: Evaluation results

a function application is constructed during innermost and just-in-time rewriting. We also briefly look at the effect on the number of rewrite rules that were applied or checked for a match.

The variations we consider are rewriting with (A) and without annotations where the former is varied in whether constructor functions are taken into account (C), the use of the term construction function from Section 4.3 (T) and the use of essential-argument detection (E). The latter is only used (and applicable) in variants where also the term construction function is used. We refer to these different variants by the relevant combination of letters (A, C, T and E). To be more precise, we actually always use the term construction function, but for (T) we use true as second argument instead of false without (T) and when not using (C) we effectively replace $R_f = \emptyset$ with false in the definition of term construction function φ^N_σ.

As benchmarks we use the same examples as used in [vW07] except for the higher-order binary search (which was only added there to show the use of higher-order functions). These examples are the Fibonacci function (fib), and the benchmarks from [Oli00] (evalexp, evalsym, evaltree). The latter have been slightly adapted to compensate for the fact that we (in general) do not assume an order on rules. The version of evalexp used here is also slightly different from the one used in [vW07] because in the latter a suboptimal alternative was chosen to com-

pensate for the absence of ordered rules (which resulted in infinite reductions for innermost rewriting).

Besides the benchmarks of [vW07], for which both innermost and just-in-time rewriting use the same rewrite steps (in terms of rules applied), we include 4 others such that we have a somewhat more representative collection of benchmarks.

A frequently used operation is a simple traversal through a structure such as a list or a tree. This includes inserting or removing an element, finding a specific element and more complex functions (such as, for example, sorting) typically are repeated applications of these operations. We have chosen to take two such operations. One (set add) is the insertion of an element to a sorted list. Sorted lists are typically used as an easy representation for sets. The other benchmark consist of the evaluation of a function that determines whether all elements in a list of numbers are even.

The final 2 benchmarks are calculations of exponentials. One use Peano numbers to calculate 2^3 and the other benchmark uses numbers in binary representation to calculate 2^{21}. The latter benchmark corresponds to the implementation used in the mCRL2 toolset. All specifications of the used benchmarks are given in Appendix F.

First we take a quick look at the effect on matching and application of rewrite rules. For innermost rewriting there is no effect with respect to this measure; the specific benchmarks all have in common that normal forms consist of constructor functions only, so no extra rewrite rules are checked when traversing a normal form multiple times. For just-in-time rewriting we do see some variation as different annotations lead to the use of different strategies. In Table 6.2 we have summarised the results. Note that there is no change for the Fibonacci function and Peano exponentiation because there are no functions in these benchmarks that benefit from just-in-time strategies in the sense that there are no subterms that could be discarded at some point. For "all even" there is also no change, but this time because no additional information is needed to take the optimal choices. For the other benchmarks we do see some differences when varying the used techniques, but these are purely due to the specific examples. We believe it is possible to create examples for any difference one might want to see. For example, you can use a rewrite rule $f \to \mathit{if}(\neg\mathrm{false}, 1, 0)$ that causes just-in-time rewriting with only annotations to check less rules for a match than it would if one also uses constructor function information; in the latter case the rewriter is fooled into first trying rule $\mathit{if}(b, x, x) \to x$ even though it cannot be applied.

Next we consider the effect on the number of calls to the rewriter and the number of times function applications are constructed. In Table 6.3 the results are displayed for innermost rewriting and Table 6.4 contains the results for just-in-time rewriting.

The most obvious observation is that for each technique we have that it is generally advantageous to add it to the rewriter. Another observation that can be

	fib(15)	evalexp(5)	evaltree(5)	evalsym(5)
	13481	18016	21230	20443
A	13481	10200	21230	11475
A C	13481	10511	21884	11866
A T	13481	10200	21230	11475
A C T	13481	10511	21884	11866
A T E	13481	10200	21230	11475
A C T E	13481	10200	21230	11475

	set add	all even	exp peano	exp binary
	188	35	56	48
A	188	35	56	48
A C	189	35	56	82
A T	188	35	56	48
A C T	189	35	56	82
A T E	188	35	56	74
A C T E	189	35	56	82

Table 6.2: Number of rules checked for match during just-in-time rewriting

	fib(15)		evalexp(5)		evaltree(5)		evalsym(5)	
	calls	constr.	calls	constr.	calls	constr.	calls	constr.
	364026	358692	49128	48919	134137	131472	55956	56291
A	13498	19062	9737	15234	20272	31455	11834	18671
A C	10915	17466	3263	9771	7140	20301	3750	11834
A T	13498	13496	9737	12737	20272	25889	11834	15843
A C T	6945	8540	3237	7259	7109	14736	3750	9008
A T E	13498	6568	9737	6515	20272	13168	11834	8091
A C T E	6945	6568	3227	6515	7109	13168	3748	8091

	set add		all even		exp peano		exp binary	
	calls	constr.	calls	constr.	calls	constr.	calls	constr.
	3525	3071	305	234	208	159	959	495
A	245	270	91	82	59	71	100	116
A C	198	242	81	82	54	70	44	92
A T	245	188	91	53	59	47	100	92
A C T	198	161	81	53	40	33	44	68
A T E	245	116	91	39	59	24	100	62
A C T E	189	116	81	39	40	24	44	64

Table 6.3: Evaluation of temporary-term construction for innermost rewriting

easily made is that the difference between "plain" rewriting or rewriting with all of the considered techniques can make an enormous difference in performance. In the case of rewriting fib(15) with an innermost strategy, we see a reduction of the number of calls with a factor of over 50. Do note that it is easy to crank up this number to arbitrary values by simple choosing the right example (e.g. for fib(16) you get a reduction factor of almost 80). What we can conclude is that it is possible

	fib(15)		evalexp(5)		evaltree(5)		evalsym(5)	
	calls	constr.	calls	constr.	calls	constr.	calls	constr.
	171040	170662	45139	45893	113695	114085	51418	52985
A	13498	19062	9426	15232	19618	31426	11443	18669
A C	10915	17466	3263	9771	7171	20332	3750	11834
A T	13498	13496	9426	12735	19618	25860	11443	15841
A C T	6945	8540	3237	7259	7109	14736	3750	9008
A T E	13498	6568	9426	6828	19618	13799	11443	8484
A C T E	6945	6568	3228	6519	7109	13170	3749	8095

	set add		all even		exp peano		exp binary	
	calls	constr.	calls	constr.	calls	constr.	calls	constr.
	1296	1167	163	128	147	122	292	198
A	164	205	61	53	59	71	94	116
A C	152	203	57	53	54	70	49	97
A T	164	152	61	39	59	47	94	92
A C T	152	150	57	39	40	33	49	73
A T E	164	143	61	36	59	27	94	66
A C T E	152	141	57	36	40	27	44	66

Table 6.4: Evaluation of temporary-term construction for just-in-time rewriting

to get very significant improvements in performance, but that it highly depends on the input. We would not be surprised if one can also construct examples, for every combination of techniques, that have a negative effect on the performance.

Looking a bit closer at the results we can see that using the term construction function only reduces the number of constructions. This is to be expected because it replaces term constructions with direct calls to specialised rewriter functions when they would have been rewritten later on anyway. For using essential-argument detection we also see that mainly the number of term constructions is reduced; exactly what it is supposed to do. It hardly ever affects the number of calls as it only changes the moment at which certain calls are made. The usage of constructor function information influences both measures (albeit somewhat more for the number of calls). This is because it replaces calls with constructions when possible and also marks subterms consisting completely of constructor functions as normal forms. The latter saves calls, and consequently constructions, later on.

6.4 Strategy Trees

For the evaluation of strategy trees – or more specifically: the strategy trees as generated by stgen – we compare them with innermost and just-in-time rewriting by looking at the number of tried and applied rules as well as the number of calls to the rewriter. For strategy trees the latter is divided in calls for normalisation and calls for head normalisation. We write this as $n + h$, with n the number of normalisations and h the number of head normalisations.

Let us look at the example of len. As benchmark we have taken the calculation of the length of lists $[\text{fib}(n-1), \text{fib}(n-2), \ldots, \text{fib}(0)]$ for $4 \leq n \leq 7$. The results are given in Table 6.5 (where just-in-time rewriting is left out as it performs the same as innermost rewriting in this case). It is clear that the results for strategy trees are linear in the length of the list while without strategy trees the results depend highly on the elements in the list. Note that the data for innermost rewriting can be tuned to arbitrarily high values by choosing the right elements for the list.

	innermost			stgen		
	#tries	#applied	#calls	#tries	#applied	#calls
fiblist(4)	47	26	108	10	10	20+15
fiblist(5)	85	46	191	12	12	27+18
fiblist(6)	150	80	330	14	14	35+21
fiblist(7)	263	139	571	16	16	44+24

Table 6.5: Evaluation of strategy trees for len

For a somewhat more general comparison between the different strategies, we look at the benchmarks as used in Section 6.3. In Table 6.6 the results are displayed. In each case we can see that using strategy trees the number of calls to the rewriter is reduced by a significant amount. What is also clear is that strategy trees only try to apply a rule when it is certain that this will be successful. Of course, one can argue that strategy trees use matching to do so and thus the comparison is not entirely fair. For example, to rewrite a term that is already in normal form, strategy trees will never try to apply a rewrite rule but do use a lot of matching that does not turn up in the given result. However, looking at the number of calls to the rewriter, this does not seem to be a significant problem.

6.5 Previous Results

In [vW07] two implementations of rewriters for the mCRL2 toolset [Too] are evaluated w.r.t. execution time. One is an innermost compiling rewriter and the other uses just-in-time strategies. Both use annotations and match trees. They are compared with other implementations of rewriters and functional languages (Maude [CDE+02], Glasgow Haskell Compiler (GHC) [LS93], Clean [Pla95] and ASF+SDF [vdBvDH+01]) as well as evaluated in the context of state-space generators (w.r.t. CADP [GLM02] and μCRL [BFG+01]). All specifications were written for μCRL and converted as direct as possible to the other languages (except for the binary search, which was written for mCRL2). We recapitulate the results from [vW07], where OoM means that a benchmark ran out of memory before completion and NA means that a benchmark was not applicable (due to the lack of support for applicative terms). Here we are mainly interested in the comparison between the two mCRL2 rewriters. These rewriters share as much of the implementation as possible and thus only differ in the used techniques, which allows for a more ac-

		fib(15)	evalexp(5)	evaltree(5)	evalsym(5)
innermost	#tries	13481	10196	21230	11471
	#applied	6929	3221	7103	3742
	#calls	364026	49128	134137	55956
just-in-time	#tries	13481	18016	21230	20443
	#applied	6929	3221	7103	3742
	#calls	171040	45139	113695	51418
stgen	#tries	6929	3268	7569	3789
	#applied	6929	3268	7569	3789
	#calls	3802+14830	3146+7510	11051+17374	3945+8529

		set add	all even	exp peano	exp binary
innermost	#tries	289	75	56	82
	#applied	118	44	33	31
	#calls	3525	305	208	959
just-in-time	#tries	188	35	56	48
	#applied	73	20	33	31
	#calls	1296	163	147	292
stgen	#tries	73	20	33	31
	#applied	73	20	33	31
	#calls	565+180	7+34	71+33	108+66

Table 6.6: Strategy-tree evaluation

curate comparison. Note that the OoM for evalexp(17) are due to the suboptimal translation of the original specification to μCRL as noted in Section 6.3. For our purposes, however, this has little importance.

The results of the comparison with other rewriters/function languages are shown in Table 6.7. The benchmarks used are the same as we have used in the previous sections together with a higher-order binary search (Appendix F.11). The most interesting information to observe at this point is that the just-in-time rewriter is significantly slower than the innermost implementation for certain benchmarks.

	Maude	GHC	Clean	ASF	CADP	μCRL	mCRL2	
							Inner.	*JITty*
fib(32)	23.4s	4.0s	2.6s	2.7s	2.4s	2.3s	4.0s	11.2s
evalexp(17)	3.3s	0.4s	0.3s	OoM	0.5s	OoM	OoM	5.4s
evalsym(17)	231.3s	18.7s	15.8s	36.3s	OoM	19.0s	49.3s	254.2s
evaltree(17)	16.7s	OoM	2.1s	1.6s	0.6s	1.0s	1.9s	25.6s
b.search	NA	4.5s	2.5s	NA	NA	NA	OoM	10.8s

Table 6.7: Rewriting benchmarks from [vW07]

In Table 6.8 are the results for the comparison with respect to state-space generation for several benchmarks that can be found in both the μCRL and the mCRL2 toolset. An important aspect of these benchmarks is that both μCRL and mCRL2 repeatedly rewrite open terms to normal form, each time substituting variables

with other (small) open terms. This means that often terms are already in normal form to a certain degree. We observe that in this case the just-in-time rewriter no longer lags behind the innermost rewriter.

	# states	CADP	μCRL	mCRL2	
				Innermost	*JITty*
chatboxt	65536	1.3s	5.0s	4.0s	3.5s
1394-fin	400	65.3s	0.1s	0.5s	0.4s
1394-fin	371804	OoM	103.8s	212.1s	92.3s
ccp33	7000	25.5s	27.6s	61.8s	8.7s
ccp33	20000	OoM	79.0s	171.9s	26.2s
commprot	700	53.9s	11.0s	12.4s	13.0s
commprot	5000	OoM	77.8s	92.1s	93.0s

Table 6.8: State-space generation benchmarks from [vW07]

After observing the results from Table 6.7 the temporary-term construction techniques of Section 4.3 were implemented in the mCRL2 toolset (aside from the annotations, which were already used). In unpublished work, these new implementations were compared against an implementation without the new techniques (though not the same implementation as used in [vW07]). The results are given in Table 6.9 and are relative to the old implementation. Here the superscripts C and E indicate the use of constructor-function information and essential-argument detection, respectively. Note that the innermost rewriter was implemented using the trick described in Section 2.2 (making the need to use these new techniques not necessary).

	Inner.	JITty	JITty^C	JITty^E	JITty^{CE}
fib(32)	0.35	1.00	0.66	0.48	0.36
evalexp(17)	OoM	1.00	0.38	0.35	0.35
evalsym(17)	0.34	1.00	0.37	0.34	0.33
evaltree(17)	0.26	1.00	0.29	0.36	0.28
b.search	NA	1.00	0.53	0.44	0.28

Table 6.9: Benchmarks

These results clearly show that using the techniques from Section 4.3 brings the mCRL2 just-in-time implementation up to speed with the innermost implementation for those benchmarks on which the just-in-time rewriter performed worse before. This corresponds precisely with the results of Table 6.3 and Table 6.4, where the Inner. corresponds to row A T E of Table 6.3, JITty to row A of Table 6.4, JITty^x to rows with A T in combination with x.

Combining the results from Table 6.7 and Table 6.9, we can conclude that the mCRL2 just-in-time rewriter with all the techniques of Chapter 4 is quite competitive with respect to the other rewriters/functional languages. Given the

results from Section 6.4, it would be very interesting to see how an implementation of strategy trees can further improve these results. Of course, this does require some additional work to extend the temporary-term construction techniques to strategy trees.

Chapter 7

Conclusions

In Chapter 3 we have given a formal definition of match trees for non-linear patterns as well as extensions for conditional rules, applicative terms and priorities. We have given separate functions for construction of match trees – to be used in advance of rewriting – and for efficient matching using these trees. We have seen in Section 6.2 that using match trees has no significant negative effect in any of the cases we considered while it does have a very significant positive effect in certain cases.

Although we established that it is not clear what optimality means for match trees, we did give some reduction functions that clearly have a positive effect on the efficiency of matching as shown in Section 6.2. It would be interesting to investigate measures on match trees that given an indication of the influence on the performance of rewriting as a whole.

The strategy trees introduced in Chapter 5 are an extension of the just-in-time strategies of [vdP01]. We have defined the notion of thorough on strategy trees that guarantees normalisation that extends the notions of full and in-time of just-in-time strategies. Also, we have given a function that returns thorough strategy trees given a set of rewrite rules. In Section 6.4, we have seen that using strategy trees results in a significant performance improvement over just-in-time strategies. In addition, strategy trees are capable of normalising terms that result in infinite behaviour with just-in-time strategies.

One of the things we are still interested in is whether or not E-strategies can be written as strategy trees. In investigating this and to make strategy trees more generally applicable, we would suggest combining the H and NF nodes into one with an extra parameter to describe what kind of rewriting is desired (e.g. to full normal form or some weaker form). Strategies would then become functions of function symbols and the desired kind of rewriting to strategy trees. This allows for a more natural usage of strategy trees where more and/or different kinds of

normal forms are required.

We defined methods for term annotation (to keep track of normal forms) and essential-argument detection in Chapter 4. As demonstrated in Section 6.3, both these methods have a significant effect on the efficiency of rewriting. As future work, determining how to best apply annotations in the context of head-normal forms would allow us to use annotations in combination with strategy trees. In addition, extending essential-argument detection to strategies trees would allow us to take full advantage of the techniques described in this thesis.

Appendix A

Fixed-Point Definitions

A.1 Introduction

We introduce fixed-point definitions which allow one to easily define functions with infinite recursion (typically as model for non-terminating behaviour).

A.2 Semantics

First we define fixed points for arbitrary functions. For sake of simplicity we only consider functions of sort $D \to E$. It is straightforward to transform a function of sort $D_1 \times \ldots \times D_n \to E$ to $D \to E$ by introducing a sort D that consists of vectors with elements from $D_1, \ldots, D_n$.

For the order $\leq_{D \to E}$ on functions of sort $D \to E$ we use a pointwise comparison. That is, $F \leq_{D \to E} G$ is equivalent to $\forall_{v \in D}(F(v) \leq_E G(v))$. We usually leave out the subscripts when it is clear what relation is meant. We say a function $F : D \to E$ is monotonic if, and only if, $\forall_{v,w \in D}(v \leq_D w \;\Rightarrow\; F(v) \leq_E F(w))$. We write $\bigcup$ and $\bigcap$ for the join, respectively meet of function sort.

We define the least fixed point of a monotonic function $F : (D \to E) \to (D \to E)$, notation $\mu X(d : D).F(X)(d)$ (or just $\mu X.F(X)$), as follows:

$$\mu X(d : D).F(X)(d) = \bigcap \{\varphi : D \to E \;:\; F(\varphi) \leq \varphi\}$$

We define the function $\mathrm{IF} : \mathcal{P}(D) \times (D \to E) \times (D \to E) \to (D \to E)$ as follows (with $S \subseteq D$, $F, G : D \to E$ and $v \in D$).

$$\begin{array}{lcll} \mathrm{IF}(S, F, G)(v) & = & F(v) & \text{if } v \in S \\ \mathrm{IF}(S, F, G)(v) & = & G(v) & \text{if } v \notin S \end{array}$$

Theorem A.2.1 IF *is monotonic in its second and third argument.*

Proof Let $F \leq F'$. We must show that $\mathrm{IF}(S, F, G) \leq \mathrm{IF}(S, F', G)$ or equivalently that for all $v \in D$, $\mathrm{IF}(S, F, G)(v) \leq \mathrm{IF}(S, F', G)(v)$. Assume that $v \in S$. Then we have that $\mathrm{IF}(S, F, G)(v) = F(v) \leq F'(v) = \mathrm{IF}(S, F', G)(v)$. Now assume that $v \notin S$. Then we have that $\mathrm{IF}(S, F, G)(v) = G(v) \leq G(v) = \mathrm{IF}(S, F', G)(v)$. So IF is clearly monotonic in its second argument.

The proof for the third argument is symmetrical to the one above. □

In the following definition we use a notion of parameterised (syntactic) patterns. If $p(d)$ is a pattern of sort D over variable $d : E$, then we write $\mathcal{S}(p, d)$ for the set of all elements $v \in D$ such that there is a value for d that makes $p(d)$ equal to v. We write p^{-1} for a function that, given a $v \in \mathcal{S}(p, d)$, returns a value for d such that $p(d) = v$.

Definition $=_\mu$. A **(minimal) fixed-point definition** for a function $\varphi : D \to E$, with E a lattice with infimum $\bot_E$, is a finite sequence of equations of the form $\varphi(p(d)) =_\mu F(\varphi, d)$, where d is a variable of sort A, $p(d)$ is a pattern of sort D containing d and $F : (D \to E) \times A \to E$ monotonic in its first argument. The semantics of such a definition

$$\begin{array}{lcl} \varphi(p_0(d_0)) & =_\mu & F_0(\varphi, d_0) \\ & \vdots & \\ \varphi(p_n(d_n)) & =_\mu & F_n(\varphi, d_n) \end{array}$$

is that $\varphi = \mu X(d : D).\mathrm{IF}(\mathcal{S}(p_0, d_0), \lambda d' : D.F_0(X, p_0^{-1}(d')), \ldots \mathrm{IF}(\mathcal{S}(p_n, d_n), \lambda d' : D.F_n(X, p_n^{-1}(d')), \lambda d' : D.\bot_E) \ldots)(d)$.

A.3 Approximations

Given a fixed-point definition for function φ as in Definition A.2, we can define approximations φ^α of φ for ordinals α in the following way. Note that we use λ to denote limit ordinals.

$$\begin{array}{lcl} \varphi^0(d) & = & \bot_E \\ \varphi^{\alpha+1}(p_0(d_0)) & = & F_0(\varphi^\alpha, d_0) \\ & \vdots & \\ \varphi^{\alpha+1}(p_n(d_n)) & = & F_n(\varphi^\alpha, d_n) \\ \varphi^\lambda(d) & = & \bigcup_{\alpha<\lambda} \varphi^\alpha(d) \end{array}$$

From [LNS82] we get that there is an β such that $\varphi^\beta = \varphi^{\beta+1} = \varphi$.

Appendix B

Preliminaries Proofs

In this appendix we give the proofs of the theorems from Chapter 2.

B.1 Definitions and Lemmata

We define the notion of overlap for sets of positions and show that we can replace a set Π in a generalised substitution $t[\varphi]_\Pi$ with a set of positions of t without any overlap. By doing this we effectively remove all irrelevant positions from Π.

Definition *overlap.* We say a set of positions Π contains **overlap** if, and only if, there are positions π and π', both different from ϵ, such that $\pi \in \Pi$ and $\pi \cdot \pi' \in \Pi$.

Lemma B.1.1 *Let t be a term, Π a set of positions and φ a function mapping positions to terms. There is a non-overlapping set of positions $\Pi' \subseteq \mathrm{pos}(t) \cap \Pi$ such that $t[\varphi]_\Pi = t[\varphi]_{\Pi'}$.*

Proof Let Π'' be $\Pi \cap \mathrm{pos}(t)$. By definition we have $t[\varphi]_\Pi = t[\varphi]_{\Pi''}$. Also, we have that Π'' is finite due to the fact that terms are defined inductively.

Now let Π' be defined as $\{\pi \in \Pi'' \ : \ \neg\exists_{\pi',\pi''}(\pi' \cdot \pi'' \in \Pi'')\}$. Clearly Π' has no overlap and $\Pi' \subseteq \Pi'' = \Pi \cap \mathrm{pos}(t)$. We now show that $t[\varphi]_{\Pi'} = t|_{\Pi'''}\varphi$ for all finite Π''' with $\Pi' \subseteq \Pi''' \subseteq \overline{\Pi'}$ by induction on the size of $\Pi''' \setminus \Pi'$. Note that $\Pi' \subseteq \Pi'' \subseteq \overline{\Pi'}$ and that this will therefore complete the proof of Lemma B.1.1.

If $|\Pi''' \setminus \Pi'| = 0$ we have that $\Pi''' = \Pi'$ and thus it trivially holds that $t[\varphi]_{\Pi'} = t[\varphi]_{\Pi'''}$. If $|\Pi''' \setminus \Pi'| = n + 1$, for some natural number n, we have that there is are $\pi \in \Pi'$ and π' such that $\pi \cdot \pi' \in \Pi'''$. By definition this means that $t[\varphi]_{\Pi'''} = t[\varphi]_{\Pi'''\setminus\{\pi\cdot\pi'\}}$. By induction we have that $t[\varphi]_{\Pi'} = t[\varphi]_{\Pi'''\setminus\{\pi\cdot\pi'\}}$ which trivially gives us $t[\varphi]_{\Pi'} = t[\varphi]_{\Pi'''}$. □

We define more formally what it means for a sequential strategy to be full or in-time. We use the following auxiliary function ψ_{i} and ψ_{r}.

$$
\begin{array}{lcl}
\psi_{\mathrm{i}}([]) & = & \emptyset \\
\psi_{\mathrm{i}}(I \rhd s) & = & I \cup \psi_{\mathrm{i}}(s) \\
\psi_{\mathrm{i}}(R \rhd s) & = & \psi_{\mathrm{i}}(s)
\end{array}
$$

$$
\begin{array}{lcl}
\psi_{\mathrm{r}}([]) & = & \emptyset \\
\psi_{\mathrm{r}}(I \rhd s) & = & \psi_{\mathrm{r}}(s) \\
\psi_{\mathrm{r}}(R \rhd s) & = & R \cup \psi_{\mathrm{r}}(s)
\end{array}
$$

A sequential strategy s for f is full when both $\psi_{\mathrm{i}}(s) = \{1, \ldots, \mathrm{ar}(f)\}$ and $\psi_{\mathrm{r}}(s) = R_f$. A sequential strategy s for f is in-time when for each s', s'' and R with $s = s' +\!\!+ (R \rhd s'')$ it holds that $(\{1, \ldots, \mathrm{ar}(f)\} \cap \bigcup_{\rho \in R} \mathrm{esspos}(\rho)) \subseteq \psi_{\mathrm{i}}(s')$.

B.2 Theorem 2.1.1

We must show that $t|_\pi \in \mathrm{hnf}(t|_\pi)$ for all $\pi \in \mathrm{pos}(t)$ if, and only if, $t \in \mathrm{nf}(t)$. Assume that $t|_\pi \in \mathrm{hnf}(t|_\pi)$ for all $\pi \in \mathrm{pos}(t)$ and $t \notin \mathrm{nf}(t)$. The latter means, by definition, that there is a u such that $t \to u$, which means that there is a position $\pi \in \mathrm{pos}(t)$, rewrite rules $l \to r$ **if** c and substitution σ such that $t|_\pi = l\sigma$, $\eta(c\sigma)$ and $u = t[r\sigma]_\pi$. We also have that $t|_\pi \in \mathrm{hnf}(t|_\pi)$, which means that there are no v, rewrite rule $l' \to r'$ **if** c' and substitution σ' such that $t|_\pi \to^* v$, $v = l'\sigma'$ and $\eta(c'\sigma')$. This is a contradiction as u does satisfy this condition.

In the same manner one can show that it is not possible that there is a position $\pi \in \mathrm{pos}(t)$ with $t|_\pi \notin \mathrm{hnf}(t|_\pi)$ and $t \in \mathrm{nf}(t)$. This concludes the proof of Theorem 2.1.1.

B.3 Theorem 2.1.2

We must show that, for all $\rho = l \to r$ **if** c, t and Π with $\Pi \cap \mathrm{esspos}(l \to r \textbf{ if } c) = \emptyset$, we have that forall φ there exists a σ with $t = l\sigma$ and $\eta(c\sigma)$ if, and only if, there exists a τ with $t[\varphi]_\Pi = l\tau$ and $\eta(c\tau)$.

Take a σ such that $t = l\sigma$ and $\eta(c\sigma)$. We define a τ such that $t[\varphi]_\Pi = l\sigma[\varphi]_\Pi = l\tau$ and $\sigma(x) = \tau(x)$ for all $x \in \mathrm{var}(c)$ (and thus $c\sigma = c\tau$ and $\eta(c\tau)$). With Lemma B.1.1 we get a minimal Π' with $l\sigma[\varphi]_{\Pi'} = l\sigma[\varphi]_\Pi$. Note that Π' is finite as it is a subset of $\mathrm{pos}(t)$ (and terms are defined inductively).

We take $\tau(x) = \sigma(x)$ for all $x \notin \mathrm{pos_v}(l) \setminus \mathrm{esspos}(\rho)$ and we take $\tau(x) = \sigma(x)[\lambda y.\varphi(\pi \cdot y)]_{\{\pi' \,:\, \pi \cdot \pi' \in \Pi'\}}$ for all $x \in \mathrm{pos_v}(l) \setminus \mathrm{esspos}(\rho)$ and with $l|_\pi = x$. As $\mathrm{var}(c) \in \mathrm{esspos}(\rho)$, we trivially have that $c\sigma = c\tau$ and thus $\eta(c\tau)$.

We define measure m on the positions of l as follows. For all $\pi \in \mathrm{pos_v}(l)$ we define $m(\pi) = 0$ and for all $\pi \in \mathrm{pos_f}(l)$ we define $m(\pi) = 1 + \uparrow (\{0\} \cup \{m(\pi \cdot i) \;:$

$\pi \cdot i \in \mathrm{pos}(l)\}$). With induction we show that $l|_\pi \sigma[\lambda y.\varphi(\pi \cdot y)]_{\{\pi' \,:\, \pi\cdot\pi' \in \Pi'\}} = l|_\pi \tau$ for all $\pi \in \mathrm{pos}(l)$ (using m). If $m(\pi)$ is 0, we have that $l|_\pi = x$ for some variable x. If $\pi \in \mathrm{esspos}(\rho)$ we have that $\sigma(x) = \tau(x)$ and that $\{\pi' \,:\, \pi \cdot \pi' \in \Pi'\} = \emptyset$. Therefore we trivially have that $\sigma(x)[\lambda y.\varphi(\pi \cdot y)]_\emptyset = \sigma(x) = \tau(x)$. If $\pi \notin \mathrm{esspos}(\rho)$, then we have that $\sigma(x)[\lambda y.\varphi(\pi \cdot y)]_{\{\pi' \,:\, \pi\cdot\pi' \in \Pi'\}} = \tau(x)$ per definition of τ.

Now, if $m(\pi) = n + 1$ for some natural number n, we have that $\pi \in \mathrm{pos}(f)$ and thus that $l|_\pi = f(t_1, \ldots, t_m)$ for some symbol f and terms $t_1, \ldots, t_m$ with $t_i = l|_{\pi\cdot i}$ for $1 \leq i \leq m$. As Π' is finite, we have distinct π_i for $1 \leq i \leq |\Pi'|$ such that $\{\pi' \,:\, \pi \cdot \pi' \in \Pi'\} = \{\pi_1, \ldots, \pi_{|\Pi|}\}$ and thus that $f(t_1, \ldots, t_m)\sigma[\lambda y.\varphi(\pi \cdot y)]_{\{\pi' \,:\, \pi\cdot\pi' \in \Pi'\}} = f(t_1\sigma, \ldots, t_m\sigma)[\varphi(\pi \cdot \pi_1)]_{\pi_1} \ldots [\varphi(\pi \cdot \pi_{|Pi'|})]_{\pi_{|\Pi'|}}$. As $\Pi' \cap \mathrm{pos_f}(l) = \emptyset$, we can then distribute each of these substitutions inward an regroup them to obtain $f(t_1\sigma[\lambda y.\varphi(\pi \cdot 1 \cdot y)]_{\{\pi' \,:\, \pi\cdot 1\cdot\pi' \in \Pi'\}}, \ldots, t_m\sigma[\lambda y.\varphi(\pi \cdot m \cdot y)]_{\{\pi' \,:\, \pi\cdot m\cdot\pi' \in \Pi'\}})$. By induction we have that $t_i\sigma[\lambda y.\varphi(\pi \cdot i \cdot y)]_{\{\pi' \,:\, \pi\cdot i\cdot\pi' \in \Pi'\}} = t_i\tau$ for all i with $1 \leq i \leq m$. This trivially means that $l|_\pi \sigma[\lambda y.\varphi(\pi \cdot y)]_{\{\pi' \,:\, \pi\cdot\pi' \in \Pi'\}} = l|_\pi \tau$.

For the case that we have a τ with $t[\varphi]_\Pi = l\tau$ and $\eta(c\tau)$ we note that we can write t as $(t[\varphi]_\Pi)[\lambda x.t|_x]_\Pi$. This way the proof obligation becomes an instantiation of the one above.

B.4 Corollary 2.1.3

We must show that for all terms t and u such that $u \in \mathrm{rewr_s}(t)$ we have that $t \to^* u$. By Theorem 5.2.2 we have that there is a strategy $\langle \varsigma, \varsigma_\mathrm{h} \rangle$ such that $\mathrm{rewr}(t) = \mathrm{rewr_s}(t)$ and thus $u \in \mathrm{rewr}(t)$. The latter means that $t \to^* u$ by Theorem 5.2.1.

B.5 Corollary 2.1.4

We must show that if, for sequential strategy function ς, it holds that $\varsigma(f)$ is full and in-time for all function symbols f, then we have that $\mathrm{rewr_s}(t) \subseteq \mathrm{nf}(t)$ for all t. By Theorem 5.2.2 we have that there is a strategy $\langle \varsigma, \varsigma_\mathrm{h} \rangle$ such that $\mathrm{rewr}(t) = \mathrm{rewr_s}(t)$. As ς is full and in-time, we know that $\langle \varsigma, \varsigma_\mathrm{h} \rangle$ is thorough by Theorem 5.3.4. This, with Theorem 5.3.1, means that $\mathrm{rewr_s}(t) \subseteq \mathrm{nf}(t)$.

B.6 Corollary 2.1.5

We must show that if, for sequential strategy function ς, it holds that $\varsigma(f)$ is full and in-time for all function symbols f, then we have that $\mathrm{rewr_s}(t) = \emptyset$ implies that $t \to^\omega$ for all t. By Theorem 5.2.2 we have that there is a strategy $\langle \varsigma, \varsigma_\mathrm{h} \rangle$ such that $\mathrm{rewr}(t) = \mathrm{rewr_s}(t)$ and thus $\mathrm{rewr}(t) = \emptyset$. As ς is full and in-time, we know that $\langle \varsigma, \varsigma_\mathrm{h} \rangle$ is thorough by Theorem 5.3.4. This, with Theorem 5.3.2, means that $t \to^\omega$.

B.7 Theorem 2.1.6

We must show that the sequential strategies generated with strat are both full and in-time (see Section B.1 for a more formal definition). That is, we must show, for each function symbol f and finite set of rewrite rules R_f, that we have that $\psi_{\mathrm{i}}(\mathrm{strat}(R_f, \{1, \ldots, \mathrm{ar}(f)\})) = \{1, \ldots, \mathrm{ar}(f)\}$, $\psi_{\mathrm{r}}(\mathrm{strat}(R_f, \{1, \ldots, \mathrm{ar}(f)\})) = R_f$ and for all s, s' and R with $s +\!\!+ (R \rhd s') = \mathrm{strat}(R_f, \{1, \ldots, \mathrm{ar}(f)\})$ that $(\{1, \ldots, \mathrm{ar}(f)\} \cap \bigcap_{\rho \in R} \mathrm{esspos}(\rho)) \subseteq \psi_{\mathrm{i}}(s)$. Note that we trivially have that $\psi_{\mathrm{i}}(I \rhd_c s) = I \cup \psi_{\mathrm{i}}(s)$, $\psi_{\mathrm{i}}(R \rhd_c s) = \psi_{\mathrm{i}}(s)$, $\psi_{\mathrm{r}}(I \rhd_c s) = \psi_{\mathrm{r}}(s)$ and $\psi_{\mathrm{r}}(R \rhd_c s) = R \cup \psi_{\mathrm{r}}(s)$.

First we show that $\psi_{\mathrm{i}}(\mathrm{strat}(R, I)) = I$ by induction on R. If $R = \emptyset$, we have that $\mathrm{strat}(\emptyset, I) = I \rhd_c []$. We then get $\psi_{\mathrm{i}}(I \rhd_c []) = I \cup \psi_{\mathrm{i}}([]) = I \cup \emptyset = I$.

If $R \neq \emptyset$, we have that $\mathrm{strat}(R, I) = T \rhd_c J \rhd_c \mathrm{strat}(R \setminus T, I \setminus J)$ with $T = \{\rho \in R : \mathrm{dep}(\rho) \cap I = \emptyset$ and $J = \{i : i \in I \wedge \mathrm{occ}(i, R \setminus T) =\uparrow_{j \in I} \mathrm{occ}(j, R \setminus T)\}$. With induction and the fact that $J \subseteq I$, we then have that $\psi_{\mathrm{i}}(\mathrm{strat}(R, I)) = \psi_{\mathrm{i}}(T \rhd_c J \rhd_c \mathrm{strat}(R \setminus T, I \setminus J)) = J \cup \psi_{\mathrm{i}}(\mathrm{strat}(R \setminus T, I \setminus J)) = J \cup (I \setminus J) = I$.

In the same manner as above we can trivially prove that $\psi_{\mathrm{r}}(\mathrm{strat}(R, I)) = R$. Thus strat returns sequential strategies that are full.

To show that the strategies returned by strat are also in-time we observe that, due to the definition of strat, each strategy of the form $s +\!\!+ (R \rhd s')$ can be written as $s +\!\!+ \mathrm{strat}(R', I')$ for some $R' \subset R_f$ and $I' \subseteq \{1, \ldots, \mathrm{ar}(f)\}$ such that $R = \{\rho \in R' : \mathrm{dep}(\rho) \cap I' = \emptyset\}$. Also, as we have shown above, we have that $\psi_{\mathrm{i}}(s +\!\!+ \mathrm{strat}(R', I')) = \{1, \ldots, \mathrm{ar}(f)\}$ and $\psi_{\mathrm{i}}(\varsigma(R', I')) = I'$ and thus that $\{1, \ldots, \mathrm{ar}(f)\} = \psi_{\mathrm{i}}(s) \cup I'$. We trivially have that $\bigcup_{\rho \in R} \mathrm{dep}(\rho) \subseteq \{1, \ldots, \mathrm{ar}(f)\}$ and, as $\mathrm{dep}(\rho) \cap I' = \emptyset$ for all $\rho \in R$, that $\bigcup_{\rho \in R} \mathrm{dep}(\rho) \subseteq \psi_{\mathrm{i}}(s)$. Finally, by unfolding $\mathrm{dep}(\rho)$ and applying some calculus, we easily get $(\{1, \ldots, \mathrm{ar}(f)\} \cap \bigcup_{\rho \in R} \mathrm{esspos}(\rho)) \subseteq \psi_{\mathrm{i}}(s)$.

Appendix C

Match-Tree Proofs

In this appendix we give the proofs of the theorems and properties from Chapter 3.

C.1 Theorem 3.2.3

We generalise Theorem 3.2.3 to the following (where we write $\text{len}(s)$ for the size of stack s and $s.i$ for the ith element on the stack):

$$\begin{aligned}&\text{len}(s) = \text{len}(s') \ \wedge \ \text{var}(r) \subseteq (\text{var}(s) \cup V) \ \Rightarrow \\ &\qquad \mu'(\gamma_1'(s, r, V), s', \sigma) = \{r\tau \ : \ \forall_{x \in V}(\tau(x) = \sigma(x)) \ \wedge \ \forall_i(s'.i = (s.i)\tau)\}\end{aligned}$$

From this Theorem 3.2.3 follows easily:

$$\begin{aligned}&\mu(\gamma_1(l \to r), t) \\ =& \quad \{ \text{ Definitions } \mu \text{ and } \gamma_1 \ \} \\ &\mu'(\gamma_1'([l], r, \emptyset), [t], \sigma) \\ =& \quad \{ \text{ Above statement, } \text{len}([l]) = \text{len}([t]),\ \text{var}(r) \subseteq \text{var}(l) \subseteq (\text{var}([l]) \cup V) \ \} \\ &\{r\tau \ : \ \forall_{x \in \emptyset}(\tau(x) = \sigma(x)) \ \wedge \ \forall_i([t].i = ([l].i)\tau)\} \\ =& \quad \{ \text{ Simplification } \} \\ &\{r\tau \ : \ t = l\tau\}\end{aligned}$$

Before we prove the above statement we note that $V \cup \text{var}(s) = W$ is an invariant of $\gamma_1'(s, r, V)$. This is needed in order to be able to apply de induction hypothesis later on. It can be clearly seen from the definition of γ_1' that $V \cup \text{var}(s)$ is equal on both sides of each equation.

With induction on the sum of the sizes of terms on the stack: case $s = []$.

$$\begin{aligned}&\mu'(\gamma_1'([], r, V), [], \sigma) \\ =& \quad \{ \text{ Definition } \gamma_1' \ \}\end{aligned}$$

$$
\begin{array}{ll}
 & \mu'(\mathsf{R}(r), [], \sigma) \\
= & \quad \{ \text{ Definition } \mu' \ \} \\
 & \{r\sigma\} \\
= & \quad \{ \ \mathrm{var}(r) \subseteq V \ \} \\
 & \{r\tau \ : \ \forall_{x \in V}(\tau(x) = \sigma(x))\} \\
= & \quad \{ \text{ Calculus } \} \\
 & \{r\tau \ : \ \forall_{x \in V}(\tau(x) = \sigma(x)) \ \wedge \ \forall_i([].i = ([].i)\tau)\}
\end{array}
$$

The next case is $s = x \rhd s''$. Let $s' = t \rhd s'''$. We use case distinction on $x \in V$. With case distinction on $\sigma(x) = t$ for the case $x \in V$ we get the following.

$$
\begin{array}{ll}
 & \mu'(\gamma'_1(x \rhd s'', r, V), t \rhd s''', \sigma) \\
= & \quad \{ \text{ Definition } \gamma'_1, \ x \in V \ \} \\
 & \mu'(\mathsf{M}(x, \gamma'_1(s'', r, V), \mathsf{X}), t \rhd s''', \sigma) \\
= & \quad \{ \text{ Definition } \mu', \ \sigma(x) = t \ \} \\
 & \mu'(\gamma'_1(s'', r, V), s''', \sigma) \\
= & \quad \{ \text{ Induction Hypothesis } \} \\
 & \{r\tau \ : \ \forall_{y \in V}(\tau(y) = \sigma(y)) \ \wedge \ \forall_i(s'''.i = (s''.i)\tau)\} \\
= & \quad \{ \text{ Calculus, } x \in V \ \} \\
 & \{r\tau \ : \ \forall_{y \in V}(\tau(y) = \sigma(y)) \ \wedge \ \tau(x) = \sigma(x) \ \wedge \ \forall_i(s'''.i = (s''.i)\tau)\} \\
= & \quad \{ \ \sigma(x) = t \ \} \\
 & \{r\tau \ : \ \forall_{y \in V}(\tau(y) = \sigma(y)) \ \wedge \ \tau(x) = t \ \wedge \ \forall_i(s'''.i = (s''.i)\tau)\} \\
= & \quad \{ \text{ Calculus } \} \\
 & \{r\tau \ : \ \forall_{y \in V}(\tau(y) = \sigma(y)) \ \wedge \ \forall_i((t \rhd s''').i = ((x \rhd s'').i)\tau)\}
\end{array}
$$

For the case $x \in V \ \wedge \ \sigma(x) \neq t$ we get the following.

$$
\begin{array}{ll}
 & \mu'(\gamma'_1(x \rhd s'', r, V), t \rhd s''', \sigma) \\
= & \quad \{ \text{ Definition } \gamma'_1, \ x \in V \ \} \\
 & \mu'(\mathsf{M}(x, \gamma'_1(s'', r, V), \mathsf{X}), t \rhd s''', \sigma) \\
= & \quad \{ \text{ Definition } \mu', \ \sigma(x) \neq t \ \} \\
 & \mu'(\mathsf{X}, s''', \sigma) \\
= & \quad \{ \text{ Definition } \mu' \ \} \\
 & \emptyset \\
= & \quad \{ \text{ Calculus (note the false) } \} \\
 & \{r\tau \ : \ \forall_{y \in V}(\tau(y) = \sigma(y)) \ \wedge \ \text{false} \ \wedge \ \forall_i(s'''.i = (s''.i)\tau)\} \\
= & \quad \{ \ \sigma(x) \neq t \ \}
\end{array}
$$

$$
\begin{array}{ll}
 & \{r\tau \ : \ \forall_{y\in V}(\tau(y) = \sigma(y)) \ \wedge \ \sigma(x) = t \ \wedge \ \forall_i(s'''.i = (s''.i)\tau)\} \\
= & \quad \{ \text{ Calculus, } x \in V \ \} \\
 & \{r\tau \ : \ \forall_{y\in V}(\tau(y) = \sigma(y)) \ \wedge \ \tau(x) = t \ \wedge \ \forall_i(s'''.i = (s''.i)\tau)\} \\
= & \quad \{ \text{ Calculus } \} \\
 & \{r\tau \ : \ \forall_{y\in V}(\tau(y) = \sigma(y)) \ \wedge \ \forall_i((t \rhd s''').i = ((x \rhd s'').i)\tau)\}
\end{array}
$$

The case that $x \notin V$ is as follows.

$$
\begin{array}{ll}
 & \mu'(\gamma'_1(x \rhd s'', r, V), t \rhd s''', \sigma) \\
= & \quad \{ \text{ Definition } \gamma'_1, \ x \notin V \ \} \\
 & \mu'(\mathsf{S}(x, \gamma'_1(s'', r, V \cup \{x\})), t \rhd s''', \sigma) \\
= & \quad \{ \text{ Definition } \mu' \ \} \\
 & \mu'(\gamma'_1(s'', r, V \cup \{x\}), s''', \sigma[x \mapsto t]) \\
= & \quad \{ \text{ Induction Hypothesis } \} \\
 & \{r\tau \ : \ \forall_{y\in V\cup\{x\}}(\tau(y) = \sigma[x \mapsto t](y)) \ \wedge \ \forall_i(s'''.i = (s''.i)\tau)\} \\
= & \quad \{ \text{ Calculus } \} \\
 & \{r\tau \ : \ \forall_{y\in V}(\tau(y) = \sigma[x \mapsto t](y)) \ \wedge \\
 & \qquad \tau(x) = \sigma[x \mapsto t](x) \ \wedge \ \forall_i(s'''.i = (s''.i)\tau)\} \\
= & \quad \{ \text{ Calculus, } x \notin V \ \} \\
 & \{r\tau \ : \ \forall_{y\in V}(\tau(y) = \sigma(y)) \ \wedge \ \tau(x) = t \ \wedge \ \forall_i(s'''.i = (s''.i)\tau)\} \\
= & \quad \{ \text{ Calculus } \} \\
 & \{r\tau \ : \ \forall_{y\in V}(\tau(y) = \sigma(y)) \ \wedge \ \forall_i((t \rhd s''').i = ((x \rhd s'').i)\tau)\}
\end{array}
$$

This concludes the proof.

C.2 Corollary 3.2.4

We must show that γ satisfies its specification. That is, we must show that $\mu(\gamma(R), t) = \{r\sigma \ : \ l \to r \in R \ \wedge \ t = l\sigma\}$ for all finite sets of rules R and terms t. We prove this with induction on the size of R and use the fact that $\|$ is well-defined for all cases $\gamma(R') \parallel \gamma_1(\rho)$. If R is empty we get the following.

$$
\begin{array}{ll}
 & \mu(\gamma(\emptyset), t) \\
= & \\
 & \mu(\mathsf{X}, t) \\
= & \\
 & \mu'(\mathsf{X}, [t], \tau) \\
= &
\end{array}
$$

$$
\begin{aligned}
& \emptyset \\
= & \\
& \{r\sigma \ : \ l \to r \in \emptyset \ \wedge \ t = l\sigma\}
\end{aligned}
$$

When $R = \{\rho\} \cup R'$ for rewrite rule $\rho = \iota(R)$, we have the following.

$$
\begin{aligned}
& \mu(\gamma(\{\rho\} \cup R'), t) \\
= & \\
& \mu(\gamma(R' \setminus \{\rho\}) \parallel \gamma_1(\rho), t) \\
= & \\
& \mu(\gamma(R' \setminus \{\rho\}), t) \cup \mu(\gamma_1(\rho), t) \\
= & \\
& \{r\sigma \ : \ l \to r \in R' \setminus \{\rho\} \ \wedge \ t = l\sigma\} \cup \{r\sigma \ : \ l \to r \in \{\rho\} \ \wedge \ t = l\sigma\} \\
= & \\
& \{r\sigma \ : \ l \to r \in \{\rho\} \cup R' \ \wedge \ t = l\sigma\}
\end{aligned}
$$

This means γ satisfies its specification.

C.3 Property 3.2.5

For *(S¹ S)* and with case distinction on the stack we get the following two derivations. First the case that $s = []$:

$$
\begin{aligned}
& \mu'(\mathsf{S}^1(x, T), [], \sigma) \\
= & \\
& \emptyset \\
= & \\
& \mu'(\mathsf{S}(x, \mathsf{N}(T)), [], \sigma)
\end{aligned}
$$

Next $s = t \rhd s'$.

$$
\begin{aligned}
& \mu'(\mathsf{S}^1(x, T), t \rhd s', \sigma) \\
= & \\
& \mu'(T, s', \sigma[x \mapsto t]) \\
= & \\
& \mu'(\mathsf{N}(T), t \rhd s', \sigma[x \mapsto t]) \\
= & \\
& \mu'(\mathsf{S}(x, \mathsf{N}(T)), t \rhd s', \sigma)
\end{aligned}
$$

This concludes the proof of *($S^1 S$)*.

We use the same case distinction for *($M^1 M$)*.

$$\begin{aligned} & \mu'(\mathsf{M}^1(x,T,U),[],\sigma) \\ = & \\ & \mu'(U,[],\sigma) \\ = & \\ & \mu'(\mathsf{M}(x,\mathsf{N}(T),U),[],\sigma) \end{aligned}$$

For the case $s = t \rhd s'$ we also use case distinction on $\sigma(x) = t$. First the case that the latter does not hold.

$$\begin{aligned} & \mu'(\mathsf{M}^1(x,T,U),t \rhd s',\sigma) \\ = & \\ & \mu'(U,t \rhd s',\sigma) \\ = & \\ & \mu'(\mathsf{M}(x,\mathsf{N}(T),U),t \rhd s',\sigma) \end{aligned}$$

Next the case that $\sigma(x) = t$ does hold.

$$\begin{aligned} & \mu'(\mathsf{M}^1(x,T,U),t \rhd s',\sigma) \\ = & \\ & \mu'(T,s',\sigma) \\ = & \\ & \mu'(\mathsf{N}(T),t \rhd s',\sigma) \\ = & \\ & \mu'(\mathsf{M}(x,\mathsf{N}(T),U),t \rhd s',\sigma) \end{aligned}$$

This concludes the proof of *($M^1 M$)*.

Also for *(rR)* we use case distinction on the stack.

$$\begin{aligned} & \mu'(\mathsf{R}(r),[],\sigma) \\ = & \\ & \{r\sigma\} \\ = & \\ & \mu'(\mathsf{R}(\{r\}),[],\sigma) \end{aligned}$$

And finally:

$$\begin{aligned} & \mu'(\mathsf{R}(r),t \rhd s',\sigma) \\ = & \end{aligned}$$

$$
\begin{array}{ll}
 & \emptyset \\
= & \\
 & \mu'(\mathsf{R}(\{r\}), t \rhd s', \sigma)
\end{array}
$$

This concludes the proof of *(rR)*.

C.4 Theorem 3.2.7

For this proof we limit ourselves to the most interesting/complicated cases. We use induction on the structure of the arguments of $\|$.

Let $\eta(t\sigma)$. We then have the following.

$$
\begin{array}{ll}
 & \mu'(\mathsf{C}(t, T, T') \parallel U, s, \sigma) \\
= & \quad \{ \text{ Definition } \| \ \} \\
 & \mu'(\mathsf{C}(t, T \parallel U, T' \parallel U), s, \sigma) \\
= & \quad \{ \text{ Definition } \mu, \ \eta(t\sigma) \ \} \\
 & \mu'(T \parallel U, s, \sigma) \\
= & \quad \{ \text{ Induction Hypothesis } \} \\
 & \mu'(T, s, \sigma) \cup \mu'(U, s, \sigma) \\
= & \quad \{ \text{ Definition } \mu, \ \eta(t\sigma) \ \} \\
 & \mu'(\mathsf{C}(t, T, T'), s, \sigma) \cup \mu'(U, s, \sigma)
\end{array}
$$

In the case that $\neg\, \eta(t\sigma)$ we have a similar derivation.

Note that a similar technique can be used for M and S where we can limit U to the cases as observed in Section 3.4. The same holds for E where one has to use the fact that most nodes can be eliminated if they occur in the wrong subtree of E:

$$
\begin{array}{ll}
 & \mu'(\mathsf{E}(T, T') \parallel \mathsf{N}(U), [], \sigma) \\
= & \quad \{ \text{ Definition } \| \ \} \\
 & \mu'(\mathsf{E}(T \parallel \mathsf{N}(U), T'), [], \sigma) \\
= & \quad \{ \text{ Definition } \mu \ \} \\
 & \mu'(T', [], \sigma) \\
= & \quad \{ \text{ Definition } \mu \ \} \\
 & \mu'(T', [], \sigma) \cup \mu'(\mathsf{N}(U), [], \sigma) \\
= & \quad \{ \text{ Definition } \mu \ \} \\
 & \mu'(\mathsf{E}(T, T'), [], \sigma) \cup \mu'(\mathsf{N}(U), [], \sigma)
\end{array}
$$

We conclude with combination of F and N. Let the stack have at least one element.

$$
\begin{array}{ll}
 & \mu'(\mathsf{F}(f,T,T') \parallel \mathsf{N}(U), t \rhd s, \sigma) \\
= & \quad \{ \text{ Definition } \parallel \ \} \\
 & \mu'(\mathsf{F}(f, T \parallel \mathsf{N}^{\mathrm{ar}(f)}(U), T' \parallel \mathsf{N}(U)), t \rhd s, \sigma) \\
= & \quad \{ \text{ Case distinction on } t \ \} \\
 & \mu'(\mathsf{F}(f, T \parallel \mathsf{N}^{\mathrm{ar}(f)}(U), T' \parallel \mathsf{N}(U)), x \rhd s, \sigma) \quad \textbf{if } t = x \\
 & \mu'(\mathsf{F}(f, T \parallel \mathsf{N}^{\mathrm{ar}(f)}(U), T' \parallel \mathsf{N}(U)), \quad \textbf{if} \\
 & \quad f(t_1, \ldots, t_n) \rhd s, \sigma) \quad \textbf{if } t = f(t_1, \ldots, t_n) \\
 & \mu'(\mathsf{F}(f, T \parallel \mathsf{N}^{\mathrm{ar}(f)}(U), T' \parallel \mathsf{N}(U)), \quad \textbf{if} \\
 & \quad g(t_1, \ldots, t_n) \rhd s, \sigma) \quad \textbf{if } t = g(t_1, \ldots, t_n) \ \wedge \ f \neq g \\
= & \quad \{ \text{ Definition } \mu \ \} \\
 & \mu'(T' \parallel \mathsf{N}(U), x \rhd s, \sigma) \quad \textbf{if } t = x \\
 & \mu'(T \parallel \mathsf{N}^{\mathrm{ar}(f)}(U), t_1 \rhd \ldots \rhd t_n \rhd s, \sigma) \quad \textbf{if } t = f(t_1, \ldots, t_n) \\
 & \mu'(T' \parallel \mathsf{N}(U), g(t_1, \ldots, t_n) \rhd s, \sigma) \quad \textbf{if } t = g(t_1, \ldots, t_n) \ \wedge \ f \neq g \\
= & \quad \{ \text{ Induction Hypothesis } \} \\
 & \mu'(T', x \rhd s, \sigma) \cup \mu'(\mathsf{N}(U), x \rhd s, \sigma) \quad \textbf{if } t = x \\
 & \mu'(T, t_1 \rhd \ldots \rhd t_n \rhd s, \sigma) \cup \mu'(\mathsf{N}^{\mathrm{ar}(f)}(U), \quad \textbf{if} \\
 & \quad t_1 \rhd \ldots \rhd t_n \rhd s, \sigma) \quad \textbf{if } t = f(t_1, \ldots, t_n) \\
 & \mu'(T', g(t_1, \ldots, t_n) \rhd s, \sigma) \cup \mu'(\mathsf{N}(U), \quad \textbf{if} \\
 & \quad g(t_1, \ldots, t_n) \rhd s, \sigma) \quad \textbf{if } t = g(t_1, \ldots, t_n) \ \wedge \ f \neq g \\
= & \quad \{ \text{ Definition } \mu \ \} \\
 & \mu'(F(f,T,T'), x \rhd s, \sigma) \cup \mu'(\mathsf{N}(U), x \rhd s, \sigma) \quad \textbf{if } t = x \\
 & \mu'(F(f,T,T'), f(t_1, \ldots, t_n) \rhd s, \sigma) \cup \quad \textbf{if} \\
 & \quad \mu'(U, s, \sigma) \quad \textbf{if } t = f(t_1, \ldots, t_n) \\
 & \mu'(F(f,T,T'), g(t_1, \ldots, t_n) \rhd s, \sigma) \cup \quad \textbf{if} \\
 & \quad \mu'(\mathsf{N}(U), g(t_1, \ldots, t_n) \rhd s, \sigma) \quad \textbf{if } t = g(t_1, \ldots, t_n) \ \wedge \ f \neq g \\
= & \quad \{ \text{ Definition } \mu \ \} \\
 & \mu'(F(f,T,T'), x \rhd s, \sigma) \cup \mu'(\mathsf{N}(U), x \rhd s, \sigma) \quad \textbf{if } t = x \\
 & \mu'(F(f,T,T'), f(t_1, \ldots, t_n) \rhd s, \sigma) \cup \quad \textbf{if} \\
 & \quad \mu'(\mathsf{N}(U), f(t_1, \ldots, t_n) \rhd s, \sigma) \quad \textbf{if } t = f(t_1, \ldots, t_n) \\
 & \mu'(F(f,T,T'), g(t_1, \ldots, t_n) \rhd s, \sigma) \cup \quad \textbf{if} \\
 & \quad \mu'(\mathsf{N}(U), g(t_1, \ldots, t_n) \rhd s, \sigma) \quad \textbf{if } t = g(t_1, \ldots, t_n) \ \wedge \ f \neq g \\
= & \quad \{ \text{ Case elimination } \} \\
 & \mu'(F(f,T,T'), t \rhd s, \sigma) \cup \mu'(\mathsf{N}(U), t \rhd s, \sigma)
\end{array}
$$

C.5 Theorem 3.3.1

The proof of Theorem 3.3.1 follows the same lines as that of Theorem 3.2.3 and is therefore not given here.

C.6 Property 3.3.3

We must show that $\mathsf{R}'(R) =_\mu \mathsf{R}(\{r \ : \ l \to r \in R\})$ or, equivalently, $\mu'(\mathsf{R}'(R), s, \sigma) = \mu'(\mathsf{R}(\{r \ : \ l \to r \in R\}), s, \sigma)$ for all stacks s and substitutions σ. Assume that $s = []$. Then we get the following.

$$\begin{aligned} & \mu'(\mathsf{R}'(R), [], \sigma) \\ = & \\ & \{r\sigma \ : \ l \to r \in R\} \\ = & \\ & \{r'\sigma \ : \ r' \in \{r \ : \ l \to r \in R\}\} \\ = & \\ & \mu'(\mathsf{R}(\{r \ : \ l \to r \in R\}), [], \sigma) \end{aligned}$$

If $s = t \rhd s'$ for some term t and stack s', then we have the following.

$$\begin{aligned} & \mu'(\mathsf{R}'(R), t \rhd s', \sigma) \\ = & \\ & \emptyset \\ = & \\ & \mu'(\mathsf{R}(\{r \ : \ l \to r \in R\}), t \rhd s', \sigma) \end{aligned}$$

C.7 Theorem 3.3.4

We must show that for all finite sets R of rewrite rules, priority functions φ and terms t we have that $\mu_{\mathrm{p}}(\overline{\gamma}(R), \varphi, t) = \mu(\mathrm{prior}(\overline{\gamma}(R), \varphi), t)$ or, equivalently, $\mu(\mathrm{prior}(\overline{\gamma}(R), \varphi), t) = \{r\sigma \ : \ t = l\sigma \ \wedge \ l \to r \in \varphi(\{l' \to r' \in R \ : \ t = l'\sigma\})\}$. The proof of this follows the same lines as the proof of Theorem 3.2.3 and is therefore not given here.

C.8 Theorem 3.4.1

We must show that $\mathrm{reduce}(T) =_\mu T$. In this proof we use the following predicates.

Definition P_{F}.

$$P_{\mathsf{F}}(F, s) = \forall_{f,t,s'}(f \in F \ \wedge \ s = t \rhd s' \ \Rightarrow \ t \in \mathbb{V} \ \vee \ \mathrm{hs}(t) \neq f)$$

Definition P_{M}.

$$P_{\mathsf{F}}(M_t, M_f, s, \sigma) = \forall_{t,s'}(s = t \rhd s' \ \Rightarrow \ \forall_{x \in M_t}(\sigma(x) = t) \ \wedge \ \forall_{x \in M_f}(\sigma(x) \neq t))$$

To prove Theorem 3.4.1 we simultaneously have to prove the following lemmas.

Lemma C.8.1.

$$P_{\mathsf{F}}(F, s) \;\Rightarrow\; \mu'(\text{reduce}_{\mathsf{F}}(T, F), s, \sigma) = \mu'(T, s, \sigma)$$

Lemma C.8.2.

$$\begin{array}{lll} \text{reduces}_{\mathsf{S}}(T, \emptyset) & =_{\mu} & T \\ \text{reduces}_{\mathsf{S}}(T, \{x\} \cup S) & =_{\mu} & \mathsf{S}(x, T[x/S]) \end{array}$$

Lemma C.8.3.

$$P_{\mathsf{M}}(M_t, M_f, s, \sigma) \;\Rightarrow\; \mu'(\text{reduce}_{\mathsf{M}}(T, M_t, M_f), s, \sigma) = \mu'(T, s, \sigma)$$

First, however, we prove the following lemmas.

Lemma C.8.4. $\forall_{F,U}(T = \text{reduce}_{\mathsf{F}}(U, F) \;\Rightarrow\; P_{\mathsf{F}}(F, s, \sigma)$ is an invariant of $\mu'(T, s, \sigma)$

Proof For the introductions of $\text{reduce}_{\mathsf{F}}$ in reduce, $\text{reduces}_{\mathsf{S}}$ and $\text{reduce}_{\mathsf{M}}$ the invariant holds trivially as $P_{\mathsf{F}}(\emptyset, s)$ holds for every s and σ.

We look at the case that we have $\mu'(\text{reduce}_{\mathsf{F}}(\mathsf{F}(f, T, U), F), s, \sigma)$ with $P_{\mathsf{F}}(F, s)$. If $f \in F$, then we get $\mu'(\text{reduce}_{\mathsf{F}}(U, F), s, \sigma)$ and must show that $P_{\mathsf{F}}(F, s)$ holds. The latter is trivially the case. If $f \notin F$, then we get $\mu'(\mathsf{F}(f, \text{reduce}(T), \text{reduce}_{\mathsf{F}}(U, F \cup \{f\})), s, \sigma)$. The only interesting case here is when $s = g(t_1, \ldots, t_n) \rhd s'$ with $g \neq f$. In this case we get $\mu'(\text{reduce}_{\mathsf{F}}(U, F \cup \{f\}), g(t_1, \ldots, t_n) \rhd s', \sigma)$ for which we must show that $P_{\mathsf{F}}(F \cup \{f\}, g(t_1, \ldots, t_n) \rhd s')$. The latter is trivially equivalent to $P_{\mathsf{F}}(F, s) \;\wedge\; (g(t_1, \ldots, t_n) \in \mathbb{V} \;\vee\; \text{hs}(g(t_1, \ldots, t_n)) \neq f)$. We have that $P_{\mathsf{F}}(F, s)$, $\text{hs}(g(t_1, \ldots, t_n)) = g$ and $g \neq f$, which means we are done. □

Lemma C.8.5. $\forall_{U,M_t,M_f}(T = \text{reduce}_{\mathsf{M}}(U, M_t, M_f) \;\Rightarrow\; P_{\mathsf{M}}(M_t, M_f, s, \sigma)$ is an invariant of $\mu'(T, s, \sigma)$

Proof For the introduction of $\text{reduce}_{\mathsf{M}}$ in reduce the invariant holds trivially as $P_{\mathsf{M}}(\emptyset, \emptyset, s, \sigma)$ holds for every s and σ.

We look at the case that we have $\mu'(\text{reduce}_{\mathsf{M}}(\mathsf{M}(x, T, U), M_t, M_f), s, \sigma)$ with $P_{\mathsf{M}}(M_t, M_f, s, \sigma)$. If $x \in M_t$, then we get $\mu'(\text{reduce}_{\mathsf{M}}(T, M_t, M_f), s, \sigma)$ and must show that $P_{\mathsf{M}}(M_t, M_f, s, \sigma)$ holds. The latter is trivially the case. If $x \in M_f$, then a similar reasoning can be used. In the case that $x \notin M_t \cup M_f$ we get $\mu'(\mathsf{M}(x, \text{reduce}_{\mathsf{M}}(T, M_t \cup \{x\}, M_f), \text{reduce}_{\mathsf{M}}(U, M_t, M_f \cup \{x\})), s, \sigma)$. The only interesting cases here are when $s = t \rhd s'$. If $\sigma(x) = t$, we get $\mu'(\text{reduce}_{\mathsf{M}}(T, M_t \cup \{x\}, M_f), t \rhd s', \sigma)$ for which we must show that $P_{\mathsf{M}}(M_t \cup \{x\}, M_f, t \rhd s', \sigma)$. The latter is trivially equivalent to $P_{\mathsf{M}}(M_t, M_f, s, \sigma) \;\wedge\; \sigma(x) = t$. We have that both $P_{\mathsf{M}}(M_t, M_f, s, \sigma)$ and $\sigma(x) = t$ hold, which means we are done. The case that $\sigma(x) \neq t$ follows a similar reasoning. □

We now prove Theorem 3.4.1 together with Lemma C.8.1, Lemma C.8.2 and Lemma C.8.3. For this proof we use induction on the structure of the (tree) argument.

Many cases are straightforward (by application of the induction hypothesis or because they are of the form $\text{reduce}_a(T,\ldots) = \text{reduce}_b(T,\ldots)$; the latter can be ignored due to the absence of cyclic dependencies) and therefore not given here.

First we look at $\text{reduce}_{\mathsf{F}}(\mathsf{F}(f,T,U),F)$. Let $f \in F$:

$$
\begin{array}{ll}
 & \mu'(\text{reduce}_{\mathsf{F}}(\mathsf{F}(f,T,U),F),s,\sigma) \\
= & \quad \{\ f \in F\ \} \\
 & \mu'(\text{reduce}_{\mathsf{F}}(U,F),s,\sigma) \\
= & \quad \{\ \text{Induction Hypothesis}\ \} \\
 & \mu'(U,s,\sigma) \\
= & \quad \{\ \text{Case distinction on } s, \text{ Definition } \mu, \\
 & \qquad U \text{ is a } \mathsf{F}, \mathsf{X} \text{ or } \mathsf{N} \text{ (conform to the observation in Section 3.4)}\ \} \\
 & \emptyset \qquad\qquad\qquad \textbf{if}\ \ s = [] \\
 & \mu'(U, t \rhd s', \sigma) \quad \textbf{if}\ \ s = t \rhd s' \\
= & \quad \{\ \text{Definition } \mu,\ P_{\mathsf{F}}(F,s) \text{ implies } t \in Vars\ \vee\ \text{hs}(t) \neq f\ \} \\
 & \mu'(F(f,T,U),[],\sigma \qquad \textbf{if}\ \ s = [] \\
 & \mu'(F(f,T,U), t \rhd s', \sigma) \quad \textbf{if}\ \ s = t \rhd s' \\
= & \quad \{\ \text{Case elimination}\ \} \\
 & \mu'(F(f,T,U),s,\sigma)
\end{array}
$$

If $f \notin F$ we have the following.

$$
\begin{array}{ll}
 & \mu'(\text{reduce}_{\mathsf{F}}(\mathsf{F}(f,T,U),F),s,\sigma) \\
= & \quad \{\ f \in F\ \} \\
 & \mu'(\mathsf{F}(f,\text{reduce}(T),\text{reduce}_{\mathsf{F}}(U,F \cup f),s,\sigma) \\
= & \quad \{\ \text{Induction Hypothesis, Lemma C.8.4}\ \} \\
 & \mu'(\mathsf{F}(f,T,U),s,\sigma)
\end{array}
$$

Next we consider the case $\text{reduce}_{\mathsf{S}}(\mathsf{N}(T),V)$. The cases for $\text{reduce}_{\mathsf{S}}(\mathsf{X},V)$ and $\text{reduce}_{\mathsf{S}}(\mathsf{F}(f,T,U),V)$ follow the same lines and are therefore not given here. Let $V = \emptyset$.

$$
\begin{array}{ll}
 & \mu'(\text{reduce}_{\mathsf{S}}(\mathsf{N}(T),\emptyset),s,\sigma) \\
= & \quad \{\ \text{Definition reduce}_{\mathsf{S}}\ \} \\
 & \mu'(\text{reduce}(\mathsf{N}(T)),s,\sigma)
\end{array}
$$

For the case $V = \{x\} \cup S$ we get the following.

$$
\begin{array}{ll}
& \mu'(\text{reduce}_{\mathsf{S}}(\mathsf{N}(T), \{x\} \cup S), s, \sigma) \\
= & \quad \{ \text{ Definition reduce}_{\mathsf{S}} \ \} \\
& \mu'(\mathsf{S}(x, \text{reduce}(\mathsf{N}(T)[x/V])), s, \sigma) \\
= & \quad \{ \text{ Definition } [/], \text{ reduce } \} \\
& \mu'(\mathsf{S}(x, \mathsf{N}(\text{reduce}(T[x/V]))), s, \sigma) \\
= & \quad \{ \text{ Induction Hypothesis } \} \\
& \mu'(\mathsf{S}(x, \mathsf{N}(T[x/V])), s, \sigma) \\
= & \quad \{ \text{ Definition } [/] \ \} \\
& \mu'(\mathsf{S}(x, \mathsf{N}(T)[x/V]), s, \sigma)
\end{array}
$$

The following case is $\text{reduce}_{\mathsf{S}}(\mathsf{S}(x, T), V)$.

$$
\begin{array}{ll}
& \mu'(\text{reduce}_{\mathsf{S}}(\mathsf{S}(x, T), V), s, \sigma) \\
= & \quad \{ \text{ Definition reduce}_{\mathsf{S}} \ \} \\
& \mu'(\text{reduce}_{\mathsf{S}}(T, V \cup \{x\}), s, \sigma) \\
= & \quad \{ \text{ Induction Hypothesis } \} \\
& \mu'(\mathsf{S}(x, T[x/V]), s, \sigma)
\end{array}
$$

Next we consider the case $\text{reduce}_{\mathsf{M}}(\mathsf{M}(x, T, U), M_t, Mf)$. Let $x \in M_t$.

$$
\begin{array}{ll}
& \mu'(\text{reduce}_{\mathsf{M}}(\mathsf{M}(x, T, U), M_t, M_f), s, \sigma) \\
= & \quad \{ \text{ Definition reduce}_{\mathsf{M}} \ \} \\
& \mu'(\text{reduce}_{\mathsf{M}}(T, M_t, M_f), s, \sigma) \\
= & \quad \{ \text{ Induction Hypothesis, Lemma C.8.5 } \} \\
& \mu'(T, s, \sigma) \\
= & \quad \{ \text{ Definition } \mu, \text{ invariant and } x \in M_t \text{ imply } s = t \rhd s'' \ \Rightarrow \ \sigma(x) = t \ \} \\
& \mu'(\mathsf{M}(x, T, U), s, \sigma)
\end{array}
$$

The case that $x \in M_f$ is similar. Finally the case that $x \notin M_t \cup M_f$.

$$
\begin{array}{ll}
& \mu'(\text{reduce}_{\mathsf{M}}(\mathsf{M}(x, T, U), M_t, M_f), s, \sigma) \\
= & \quad \{ \text{ Definition reduce}_{\mathsf{M}} \ \} \\
& \mu'(\mathsf{M}(x, \text{reduce}_{\mathsf{M}}(T, M_t \cup \{x\}, M_f), \text{reduce}_{\mathsf{M}}(U, M_t, M_f \cup \{x\})), s, \sigma) \\
= & \quad \{ \text{ Induction Hypothesis, Lemma C.8.5 } \} \\
& \mu'(\mathsf{M}(x, T, U), s, \sigma)
\end{array}
$$

This concludes the proof of Theorem 3.4.1.

C.9 Theorem 3.4.2

We must show that $\text{clean}(T) =_\mu T$. To prove this theorem we use the following definition of free variables in match trees.

Definition fv**.**

$$\begin{array}{lcl}
\text{fv}(\mathsf{X}) & = & \emptyset \\
\text{fv}(\mathsf{F}(f,T,U)) & = & \text{fv}(T) \cup \text{fv}(U) \\
\text{fv}(\mathsf{S}(x,T)) & = & \text{fv}(T) \setminus \{x\} \\
\text{fv}(\mathsf{M}(x,T,U)) & = & \text{fv}(T) \cup \text{fv}(U) \cup \{x\} \\
\text{fv}(\mathsf{C}(t,T,U)) & = & \text{fv}(T) \cup \text{fv}(U) \cup \text{var}(t) \\
\text{fv}(\mathsf{N}(T)) & = & \text{fv}(T) \\
\text{fv}(\mathsf{R}(R) & = & \text{var}(R) \\
\text{fv}(\mathsf{E}(T,U)) & = & \text{fv}(T) \cup \text{fv}(U)
\end{array}$$

We then have the following lemma.

Lemma C.9.1.

$$V = \text{fv}(U) \text{ where } \langle U, V\rangle = \text{clean}(T)$$

Proof This follows in a straightforward manner from the definition of clean (note the similarity with the definition of fv). □

We also have the following property.

Property C.9.1 *For match trees constructed with* γ *and* reduce *we have the following properties.*

$$\begin{array}{llll}
\textit{(Selm)} & \mathsf{S}(x,T) & =_\mu T & \textit{if } x \notin \text{fv}(T) \\
\textit{(Melm)} & \mathsf{M}(x,T,T) & =_\mu T & \\
\textit{(Celm)} & \mathsf{C}(t,T,T) & =_\mu T & \\
\textit{(ETX)} & \mathsf{E}(T,\mathsf{X}) & =_\mu T & \\
\textit{(EXT)} & \mathsf{E}(\mathsf{X},T) & =_\mu T &
\end{array}$$

Proof We prove *(Selm)* with case distinction on the stack. First the case $s = []$:

$$\begin{array}{ll}
 & \mu'(\mathsf{S}(x,T),[],\sigma) \\
= & \quad \{ \text{ Definition } \mu \text{ } \} \\
 & \emptyset \\
= & \quad \{ \text{ } T \text{ is an } \mathsf{F}, \mathsf{X} \text{ or } \mathsf{N} \text{ node (conform to the observation in Section 3.4) } \} \\
 & \mu'(T,[],\sigma)
\end{array}$$

For the case that $s = t \rhd s'$ we have that $\mu'(\mathsf{S}(x,T), t \rhd s', \sigma) = \mu'(T, s', \sigma[x \mapsto t])$. We prove that $\mu'(T, s'', \sigma) = \mu'(T, s'', \sigma[x \mapsto t])$ which trivially concludes the proof of *(Selm)*. With induction on the structure of T we get the following cases.

- $T = \mathsf{X}$; this case holds trivially.
- $T = \mathsf{F}(f, U, U')$; trivial with induction hypothesis.
- $T = \mathsf{S}(y, U)$; this trivially holds when $s'' = []$. If $s'' = u \rhd s'''$ and $y = x$, we can derive the following:

$$\begin{aligned}
& \mu'(\mathsf{S}(x, U), u \rhd s''', \sigma[x \mapsto t]) \\
= & \\
& \mu'(U, u \rhd s''', \sigma[x \mapsto t][x \mapsto u]) \\
= & \\
& \mu'(U, u \rhd s''', \sigma[x \mapsto u]) \\
= & \\
& \mu'(\mathsf{S}(x, U), u \rhd s''', \sigma)
\end{aligned}$$

  The case that $y \neq x$ is similar to the case below.

- $T = \mathsf{M}(y, U, U')$; this trivially holds when $s'' = []$. If $s'' = u \rhd s'''$, we can derive the following:

$$\begin{aligned}
& \mu'(\mathsf{M}(y, U, U'), u \rhd s''', \sigma[x \mapsto t]) \\
= & \quad \{\ V \text{ is } U \text{ if } \sigma[x \mapsto t](y) = u \text{ and } U' \text{ otherwise } \} \\
& \mu'(V, u \rhd s''', \sigma[x \mapsto t]) \\
= & \quad \{\ \text{Induction Hypothesis } \} \\
& \mu'(V, u \rhd s''', \sigma) \\
= & \quad \{\ x \notin \mathrm{fv}(\mathsf{M}(y, U, U')) \text{ implies } y \neq x \text{ and thus } \sigma[x \mapsto t](y) = \sigma(y) \ \} \\
& \mu'(\mathsf{M}(y, U, U'), u \rhd s''', \sigma)
\end{aligned}$$

- $T = \mathsf{C}(u', U, U')$; this case is similar to the case above, where $\tau(y) = u$ is replaced with $u'\tau \to^* \text{true}$ (for all τ).
- $T = \mathsf{N}(U)$; trivial with induction hypothesis.
- $T = \mathsf{R}(R)$; this trivially holds when $s'' \neq []$. If $s'' = []$, we can derive the following:

$$\begin{aligned}
& \mu'(\mathsf{R}(R), [], \sigma[x \mapsto t]) \\
= & \quad \{\ \text{Definition } \mu\ \} \\
& \{r(\sigma[x \mapsto t]) \ :\ r \in R\} \\
= & \quad \{\ x \notin \mathrm{fv}(\mathsf{R}(R)) \text{ implies } x \notin \mathrm{var}(r) \text{ for all } r \in R \text{ and} \\
& \quad\ \ \text{thus } r(\sigma[x \mapsto t]) = r\sigma \text{ for all such } r\ \}
\end{aligned}$$

$$
\begin{aligned}
& \{r\sigma \ : \ r \in R\} \\
= \quad & \{ \text{ Definition } \mu \ \} \\
& \mu'(\mathsf{R}(R), [], \sigma)
\end{aligned}
$$

- $T = \mathsf{E}(U, U')$; trivial with induction hypothesis.

We prove *(Melm)* with case distinction on the stack. First the case $s = []$:

$$
\begin{aligned}
& \mu'(\mathsf{M}(x, T, T), [], \sigma) \\
= \quad & \{ \text{ Definition } \mu \ \} \\
& \emptyset \\
= \quad & \{ \ T \text{ is an } \mathsf{S}, \mathsf{F}, \mathsf{X} \text{ or } \mathsf{N} \text{ node (conform to the observation in Section 3.4)} \ \} \\
& \mu'(T, [], \sigma)
\end{aligned}
$$

Next the case $s = t \rhd s'$.

$$
\begin{aligned}
& \mu'(\mathsf{M}(x, T, T), t \rhd s', \sigma) \\
= \quad & \{ \text{ Definition } \mu \ \} \\
& \mu'(T, t \rhd s', \sigma)
\end{aligned}
$$

This concludes the proof of *(Melm)*. The proofs of *(Celm)* and *(ETX)* are similar.
For *(EXT)* we also use case distinction on the stack. First the case $s = []$:

$$
\begin{aligned}
& \mu'(\mathsf{E}(\mathsf{X}, T), [], \sigma) \\
= \quad & \{ \text{ Definition } \mu \ \} \\
& \mu'(T, [], \sigma)
\end{aligned}
$$

Next the case $s = t \rhd s'$.

$$
\begin{aligned}
& \mu'(\mathsf{E}(\mathsf{X}, T), t \rhd s', \sigma) \\
= \quad & \{ \text{ Definition } \mu \ \} \\
& \mu'(\mathsf{X}, t \rhd s', \sigma) \\
= \quad & \{ \text{ Definition } \mu \ \} \\
& \emptyset \\
= \quad & \{ \ T \text{ is an } \mathsf{R} \text{ node (conform to the observation in Section 3.4)} \ \} \\
& \mu'(T, t \rhd s', \sigma)
\end{aligned}
$$

□

With Lemma C.9.1 and Property C.9.1 it is quite straightforward to show that Theorem 3.4.2 holds. We therefore do not do so here.

Appendix D

Temporary-Term-Construction Proofs

In this appendix we give the proofs of the theorems from Chapter 4.

D.1 Lemmata

First we show that if the boolean argument of our term-construction function is true, then the boolean in the result is also true. That is, if we ask for a normal form, ithe function should return something it "thinks" is a normal form.

Lemma D.1.1 *For all terms t, substitutions σ and variable sets N we have that $\varphi_\sigma^N(t, \mathrm{true}) = \langle u, \mathrm{true}\rangle$ for some term u.*

Proof This follows trivially from the defintion by induction on the structure of t; each equality with true as second argument of φ_σ^N on the left-hand side has true as the second element of the result on the right-hand side. □

Next we show that if the boolean part of the result of our term-construction function is true (i.e. indicates that the term part should be in normal form), then the term part of the result is indeed a normal form.

Lemma D.1.2 *For all terms t and u, substitutions σ, variable sets N and booleans b we have that if $\varphi_\sigma^N(t, b) = \langle u, \mathrm{true}\rangle$ then $u \in \mathrm{nf}(u)$.*

Proof By induction on the size of t:

- $t = x$; we then either have $x \in N$ or $x \notin N$. The first case means that $u = \sigma(x)$ and $\sigma(x) \in \mathrm{nf}(\sigma(x))$. In the second case we have that $u = \mathrm{rewrite}(\sigma(x))$. By definition we have that $\mathrm{rewrite}(\sigma(x)) \in \mathrm{nf}(\sigma(x))$.

- $t = f(t_1, \ldots, t_n)$; we then either have $R_f = \emptyset$ or $R_f \neq \emptyset$. The first case means that $u = f(t'_1, \ldots, t'_n)$. If $b = \mathrm{false}$, we have that $\forall_{1 \leq i \leq n}(b_i)$ by definition. If $b = \mathrm{true}$, we get $\forall_{1 \leq i \leq n}(b_i)$ by Lemma D.1.1. This means, by induction, that $t'_i \in \mathrm{nf}(t'_i)$ for all i. Note that we also have that the strategy for f is $[\{1, \ldots, n\}]$ due to $R_f = \emptyset$. Therefore we have that $\mathrm{nf}(f(t'_1, \ldots, t'_n)) = f(\mathrm{nf}(t'_1), \ldots, \mathrm{nf}(t'_n)) = f(t'_1, \ldots, t'_n)$.

  In the second case ($R_f \neq \emptyset$) we have that $b = \mathrm{true}$. This means we have that $u = \mathrm{rewrite}_f^{\{i \,:\, 1 \leq i \leq n \,\wedge\, b_i\}}(t'_1, \ldots, t'_n)$ with $\langle t'_i, b_i \rangle = \varphi_\sigma^N(t_i, \mathrm{false})$ for all i such that $1 \leq i \leq n$. By induction we then have that if b_i, then also $t'_i \in \mathrm{nf}(t'_i)$, for all i. But then by definition $\mathrm{rewrite}_f^{\{i \,:\, 1 \leq i \leq n \,\wedge\, b_i\}}(t'_1, \ldots, t'_n) \in \mathrm{nf}(f(t'_1, \ldots, t'_n))$.

□

D.2 Theorem 4.3.1

We must show that, for all terms t and u, substitutions σ, sets of variables N, and booleans b and c, if $\varphi_\sigma^N(t, b) = \langle u, c \rangle$, then $t\sigma \to^* u$. By induction on the size of t:

- $t = x$; we then either have $\neg b \vee x \in N$ or $x \notin N$. The first case means that $u = \sigma(x)$ and thus trivially $\sigma(x) \to^* \sigma(x)$. In the second case we have that $u = \mathrm{rewrite}(\sigma(x))$. By definition we have that $\sigma(x) \to^* \mathrm{rewrite}(\sigma(x))$.

- $t = f(t_1, \ldots, t_n)$; we then either have $\neg b \vee R_f = \emptyset$ or $R_f \neq \emptyset$. The first case means that $u = f'(t'_1, \ldots, t'_n)$ with $f' = f$ or $f' = f^{\{i \,:\, 1 \leq i \leq n \,\wedge\, b_i\}}$ and $\langle t'_i, b_i \rangle = \varphi_\sigma^N(t_i, b)$ for all i such that $1 \leq i \leq n$. By induction we then have that $t_i\sigma \to^* t'_i$ for all i. Then we trivially have that $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma) \to^* f'(t'_1, \ldots, t'_n)$. In the second case we have that $u = \mathrm{rewrite}_f^{b_1 \ldots b_n}(t'_1, \ldots, t'_n)$ with $\langle t'_i, b_i \rangle = \varphi_\sigma^N(t_i, \mathrm{false})$ for all i such that $1 \leq i \leq n$. By induction we then have that $t_i\sigma \to^* t'_i$ for all i. Thus also $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma) \to^* f(t'_1, \ldots, t'_n)$. By Lemma D.1.2 we additionally have that $t'_i \in \mathrm{nf}(t'_i)$ for all i with b_i. But then we have that $f(t'_1, \ldots, t'_n) \to^* \mathrm{rewrite}_f^{b_1 \ldots b_n}(t'_1, \ldots, t'_n)$ by definition.

D.3 Theorem 4.3.2

We must show that, for all terms t and u, substitutions σ, sets of variables N, and booleans b and c, if $\varphi_\sigma^N(t, b) = \langle u, c \rangle$ and $b \vee c$, then $u \in \mathrm{nf}(t\sigma)$.

By Lemma D.1.1 we have that if $b = \text{true}$, then also $c = \text{true}$. That is, we have that $b \vee c$ implies c. This means, by Lemma D.1.2, that $u \in \text{nf}(u)$. Furthermore, by Theorem 4.3.1 we have that $t\sigma \to^* u$. Then we have $u \in \text{nf}(t\sigma)$ by definition.

D.4 Theorem 4.3.3

We must show that, assuming t is a term, σ a substitution and N a set of variables, we have that $\varphi^N_\sigma(t, \text{false}) = \langle u, \text{true}\rangle$ for some u if, and only if, t is either a variable x such that $x \in N$, or $t = f(t_1, \ldots, t_n)$ with $R_f = \emptyset$ and $\varphi^N_\sigma(t_i, \text{false}) = \langle u_i, \text{true}\rangle$ for some terms u_i (with $1 \leq i \leq n$). Furthermore we must show that we also have that if $\varphi^N_\sigma(t, \text{false}) = \langle u, \text{true}\rangle$ for some term u, then $u = t\sigma$.

Assume that $\varphi^N_\sigma(t, \text{false}) = \langle u, \text{true}\rangle$. By induction on the structure of t:

- $t = x$ for some variable x; this means that $\varphi^N_\sigma(t, \text{false}) = \langle \sigma(x), x \in N\rangle$ and thus $u = \sigma(x)$ and $x \in N$. Also, $u = \sigma(x) = x\sigma = t\sigma$.
- $t = f(t_1, \ldots, t_n)$ for some terms function symbol f and terms t_i with $1 \leq i \leq n$; this, means that $\varphi^N_\sigma(t, \text{false}) = \langle f(t'_1, \ldots, t'_n), \text{true}\rangle$, $R_f = \emptyset$ and $\varphi^N_\sigma(t_i, \text{false}) = \langle t'_i, \text{true}\rangle$ for all i such that $1 \leq i \leq n$. Then also, by induction, we have that $t'_i = t_i\sigma$ for all i with $1 \leq i \leq n$. And then $f(t'_1, \ldots, t'_n) = f(t_1\sigma, \ldots, t_n\sigma) = f(t_1, \ldots, t_n)\sigma = t\sigma$.

Now, assume a variable x such that $x \in N$. Then we have $\varphi^N_\sigma(t, \text{false}) = \langle \sigma(x), x \in N\rangle = \langle \sigma(x), \text{true}\rangle$. Finally, assume a function symbol f without any rewrite rules (i.e. $R_f = \emptyset$) and terms t_i with $\varphi^N_\sigma(t_i, \text{false}) = \langle t'_i, \text{true}\rangle$ for some terms t'_i with $1 \leq i \leq n$ (where n is the arity of f). Then we trivially have $\varphi^N_\sigma(f(t_1, \ldots, t_n), \text{false}) = \langle f(t'_1, \ldots, t'_n), \text{true}\rangle$.

Appendix E

Strategy Tree Proofs

In this appendix we give the proofs of the theorems and properties from Chapter 5.

E.1 Definitions and Lemmata

We define approximations of rewr according to Section A.3 as follows. Here we write λ for limit ordinals.

$$
\begin{array}{lcll}
\mathrm{rewr}^{\alpha}(t) & = & \mathrm{eval}^{\alpha}(\varsigma^{\bot}(\mathrm{hs}^{\bot}(t)), t) & \\
\mathrm{rewr}^{\alpha}_{\mathrm{h}}(t) & = & \mathrm{eval}^{\alpha}_{\mathrm{h}}(\varsigma^{\bot}_{\mathrm{h}}(\mathrm{hs}^{\bot}(t)), t) & \\
 & & & \\
\mathrm{eval}^{0}(T, t) & = & \emptyset & \\
\mathrm{eval}^{0}_{\mathrm{h}}(T, t) & = & \emptyset & \\
 & & & \\
\mathrm{eval}^{\alpha+1}(\mathsf{F}(\pi, \varphi), t) & = & \mathrm{eval}^{\alpha}(\varphi(\mathrm{hs}(t|_{\pi})), t) & \text{if } \pi \in \mathrm{pos}_{\mathrm{f}}(t) \\
\mathrm{eval}^{\alpha+1}(\mathsf{F}(\pi, \varphi), t) & = & \mathrm{eval}^{\alpha}(\varphi(\bot), t) & \text{if } \pi \notin \mathrm{pos}_{\mathrm{f}}(t) \\
\mathrm{eval}^{\alpha+1}(\mathsf{H}(\Pi, T), t) & = & \bigcup_{\varphi \in \mathrm{rewrf}^{\alpha}_{\mathrm{h}}(t,\Pi)} \mathrm{eval}^{\alpha}(T, t[\varphi]_{\Pi}) & \\
\mathrm{eval}^{\alpha+1}(\mathsf{NF}(\Pi, T), t) & = & \bigcup_{\varphi \in \mathrm{rewrf}^{\alpha}(t,\Pi)} \mathrm{eval}^{\alpha}(T, t[\varphi]_{\Pi}) & \\
\mathrm{eval}^{\alpha+1}(\mathsf{T}(R, T), t) & = & \bigcup_{u \in \mathrm{app}^{\alpha}(R,t)} \mathrm{rewr}^{\alpha}(u) & \text{if } \mathrm{app}^{\alpha}(R, t) \neq \emptyset \\
\mathrm{eval}^{\alpha+1}(\mathsf{T}(R, T), t) & = & \mathrm{eval}^{\alpha}(T, t) & \text{if } \mathrm{app}^{\alpha}(R, t) = \emptyset \\
\mathrm{eval}^{\alpha+1}(\mathsf{E}, t) & = & \{t\} & \\
\mathrm{eval}^{\alpha+1}(\mathsf{X}, t) & = & \emptyset & \\
 & & & \\
\mathrm{eval}^{\alpha+1}_{\mathrm{h}}(\mathsf{F}(\pi, \varphi), t) & = & \mathrm{eval}^{\alpha}_{\mathrm{h}}(\varphi(\mathrm{hs}(t|_{\pi})), t) & \text{if } \pi \in \mathrm{pos}_{\mathrm{f}}(t) \\
\mathrm{eval}^{\alpha+1}_{\mathrm{h}}(\mathsf{F}(\pi, \varphi), t) & = & \mathrm{eval}^{\alpha}_{\mathrm{h}}(\varphi(\bot), t) & \text{if } \pi \notin \mathrm{pos}_{\mathrm{f}}(t) \\
\mathrm{eval}^{\alpha+1}_{\mathrm{h}}(\mathsf{H}(\Pi, T), t) & = & \bigcup_{\varphi \in \mathrm{rewrf}^{\alpha}_{\mathrm{h}}(t,\Pi)} \mathrm{eval}^{\alpha}_{\mathrm{h}}(T, t[\varphi]_{\Pi}) & \\
 & & \multicolumn{2}{l}{\text{(continued on next page)}}
\end{array}
$$

$$
\begin{array}{lcll}
\mathrm{eval}_{\mathrm{h}}^{\alpha+1}(\mathsf{NF}(\Pi,T),t) & = & \bigcup_{\varphi\in\mathrm{rewrf}^{\alpha}(t,\Pi)}\mathrm{eval}_{\mathrm{h}}^{\alpha}(T,t[\varphi]_{\Pi}) & \\
\mathrm{eval}_{\mathrm{h}}^{\alpha+1}(\mathsf{T}(R,T),t) & = & \bigcup_{u\in\mathrm{app}^{\alpha}(R,t)}\mathrm{rewr}_{\mathrm{h}}^{\alpha}(u) & \text{if } \mathrm{app}^{\alpha}(R,t)\neq\emptyset \\
\mathrm{eval}_{\mathrm{h}}^{\alpha+1}(\mathsf{T}(R,T),t) & = & \mathrm{eval}_{\mathrm{h}}^{\alpha}(T,t) & \text{if } \mathrm{app}^{\alpha}(R,t)=\emptyset \\
\mathrm{eval}_{\mathrm{h}}^{\alpha+1}(\mathsf{E},t) & = & \{t\} & \\
\mathrm{eval}_{\mathrm{h}}^{\alpha+1}(\mathsf{X},t) & = & \emptyset & \\
 & & & \\
\mathrm{eval}^{\lambda}(T,t) & = & \bigcup_{\alpha<\lambda}\mathrm{eval}^{\alpha}(T,t) & \\
\mathrm{eval}_{\mathrm{h}}^{\lambda}(T,t) & = & \bigcup_{\alpha<\lambda}\mathrm{eval}_{\mathrm{h}}^{\alpha}(T,t) &
\end{array}
$$

with

$$
\mathrm{app}^{\alpha}(R,t) \quad = \quad \{u \;:\; l\to r \;\mathbf{if}\; c\in R \;\wedge\; t=l\sigma \;\wedge\; u=r\sigma \;\wedge\; \mathrm{true}\in\mathrm{rewr}^{\alpha}(c\sigma)\}
$$

$$
\begin{array}{lcl}
\mathrm{rewrf}^{\alpha}(t,\Pi) & = & \{\varphi \;:\; \forall_{\pi\in\Pi}(\pi\in\mathrm{pos}(t) \;\Rightarrow\; \varphi(\pi)\in\mathrm{rewr}^{\alpha}(t|_{\pi}))\} \\
\mathrm{rewrf}_{\mathrm{h}}^{\alpha}(t,\Pi) & = & \{\varphi \;:\; \forall_{\pi\in\Pi}(\pi\in\mathrm{pos}(t) \;\Rightarrow\; \varphi(\pi)\in\mathrm{rewr}_{\mathrm{h}}^{\alpha}(t|_{\pi}))\}
\end{array}
$$

Similarly, we have the following approximations for $\mathrm{rewr}_{\mathrm{s}}$.

$$
\begin{array}{lcll}
\mathrm{rewr}_{\mathrm{s}}^{\alpha}(t) & = & \mathrm{eval}_{\mathrm{s}}^{\alpha}(\varsigma^{\perp}(\mathrm{hs}^{\perp}(t)),t) & \\
 & & & \\
\mathrm{eval}_{\mathrm{s}}^{0}(\varsigma,t) & = & \emptyset & \\
 & & & \\
\mathrm{eval}_{\mathrm{s}}^{\alpha+1}([],t) & = & \{t\} & \\
\mathrm{eval}_{\mathrm{s}}^{\alpha+1}(I\rhd\varsigma,t) & = & \bigcup_{\psi\in\mathrm{rewrf}_{\mathrm{s}}^{\alpha}(t,I)}\mathrm{eval}_{\mathrm{s}}^{\alpha}(\varsigma,t[\psi]_{I}) & \\
\mathrm{eval}_{\mathrm{s}}^{\alpha+1}(R\rhd\varsigma,t) & = & \bigcup_{u\in\mathrm{app}^{\alpha}(R,t)}\mathrm{rewr}_{\mathrm{s}}^{\alpha}(u) & \text{if } \mathrm{app}^{\alpha}(R,t)\neq\emptyset \\
\mathrm{eval}_{\mathrm{s}}^{\alpha+1}(R\rhd\varsigma,t) & = & \mathrm{eval}_{\mathrm{s}}^{\alpha}(\varsigma,t) & \text{if } \mathrm{app}^{\alpha}(R,t)=\emptyset \\
 & & & \\
\mathrm{eval}_{\mathrm{s}}^{\lambda}(\varsigma,t) & = & \bigcup_{\alpha<\lambda}\mathrm{eval}_{\mathrm{s}}^{\alpha}(\varsigma,t) &
\end{array}
$$

with

$$
\begin{array}{lcl}
\mathrm{app}_{\mathrm{s}}^{\alpha}(R,t) & = & \{r\sigma \;:\; l\to r \;\mathbf{if}\; c\in R \;\wedge\; t=l\sigma \;\wedge\; \mathrm{true}\in\mathrm{rewr}_{\mathrm{s}}^{\alpha}(c\sigma)\} \\
 & & \\
\mathrm{rewrf}_{\mathrm{s}}^{\alpha}(t,I) & = & \{\varphi \;:\; \forall_{i\in I}(i\in\mathrm{pos}(t) \;\Rightarrow\; \varphi(i)\in\mathrm{rewr}_{\mathrm{s}}^{\alpha}(t|_{i}))\}
\end{array}
$$

Next we prove a few lemmas that we need for the theorems below. First we show that if a subterm of a term t can be rewritten, then t itself can also be rewritten (by rewriting the subterm in the context of t).

Lemma E.1.1 *If $t|_{\pi}\to_R^* u$ for some $\pi\in\mathrm{pos}(t)$, then also $t\to_R^* t[u]_{\pi}$. Also, if $t|_{\pi}\to_R^*\varphi(\pi)$ for all $\pi\in\mathrm{pos}(t)\cap\Pi$, then $t\to_R^* t[\varphi]_{\Pi}$.*

Proof The first statement follows trivially from the definition of $\to_R$ with induction on the number of steps in $\to_R^*$. We focus on the second statement.

With Lemma B.1.1 we have that there is a non-overlapping $\Pi' \subseteq \text{pos}(t) \cap \Pi$ such that $t[\varphi]_\Pi = t[\varphi]_{\Pi'}$. Then by induction on the size of Π' (note that Π' is finite as there are only a finite number of positions in t per definition) we show that $t \to^* t[\varphi]_{\Pi'}$:

- $\Pi' = \emptyset$; this means that $t[\varphi]_{\Pi'} = t$. We trivially have that $t \to_R^* t$.
- $\Pi' = \Pi'' \cup \{\pi\}$, with $\pi \notin \Pi''$; this means that $t[\varphi]_{\Pi'' \cup \{\pi\}} = (t[\varphi(\pi)]_\pi)[\varphi]_{\Pi''}$. By definition we have that $t|_\pi \to_R^* \varphi(\pi)$ and, from the above, also $t \to_R^* t[\varphi(\pi)]_\pi$. By induction we have that $t[\varphi(\pi)]_\pi \to_R^* (t[\varphi(\pi)]_\pi)[\varphi]_{\Pi''}$. (Note that Π'' does not contain any positions $\pi \cdot \pi'$ that might be invalidated by the substitution at π.)

□

Now we show that if you take a subterm of a term t to which a substitution is applied, you can also apply a related substitution to a subterm of t.

Lemma E.1.2 *Let $\pi \in \text{pos}(t)$. We have $(t[\varphi]_\Pi)|_\pi = (t|_\pi)[\lambda x.\varphi(\pi \cdot x)]_{\{\pi' \,:\, \pi.\pi' \in \Pi\}}$ if there are no π' and π'' with $\pi = \pi' \cdot \pi''$ and $\pi' \in \Pi$.*

Proof With Lemma B.1.1 we have that there is a $\Pi' \subseteq \Pi \cap \text{pos}(t)$ such that $t[\varphi]_\Pi = t[\varphi]_{\Pi'}$. With induction on the size of Π':

- $\Pi' = \emptyset$; this trivially means $(t[\varphi]_\emptyset)|_\pi = t|_\pi = (t|_\pi)[\lambda x.\varphi(\pi \cdot x)]_\emptyset$.
- $\Pi' = \Pi'' \cup \{\pi'\}$ with $\pi' \notin \Pi''$; this means we have $(t[\varphi]_{\Pi'' \cup \{\pi'\}})|_\pi = ((t[\varphi(\pi')]_{\pi'})[\varphi]_{\Pi''})|_\pi$, which, by the induction hypothesis, is equal to the term $((t[\varphi(\pi')]_{\pi'})|_\pi)[\lambda x.\varphi(\pi \cdot x)]_{\{\pi'' \,:\, \pi \cdot \pi'' \in \Pi''\}}$. We know per assumption that π' is not a prefix of π and we either have that π is a prefix of π' or not. If π is a prefix of $\pi' = \pi \cdot \pi''$ (for some π''), we must show that $(t'[u]_{\pi'})|_\pi = (t'|_\pi)[u]_{\pi''}$ (for all t') to get $(t|_\pi)[\varphi(\pi \cdot \pi'')]_{\pi''}[\lambda x.\varphi(\pi \cdot x)]_{\{\pi'' \,:\, \pi \cdot \pi'' \in \Pi''\}}$ and thus $(t|_\pi)[\lambda x.\varphi(\pi \cdot x)]_{\{\pi'' \,:\, \pi \cdot \pi'' \in \Pi'\}}$. With induction on the structure of t':
  - $t' = x$ for some variable x; this means that both π and π' must be ϵ which trivially gives $\pi' = \pi''$ and $(t'[u]_{\pi'})|_\pi = t'[u]_{\pi'} = (t'|_\pi)[u]_{\pi''}$.
  - $t' = f(t_1, \ldots, t_n)$ for some symbol f and terms $t_1, \ldots, t_n$; there are two cases:
    - $\pi = \epsilon$; which is similar to the case that $t' = x$.
    - $\pi = i.\pi'''$ for some index i and position π'''; this means that we have that $(t'[u]_{\pi'})|_\pi = f(t_1, \ldots, t_i[u]_{\pi''' \cdot \pi''}, \ldots, t_n)|_\pi = (t_i[u]_{\pi''' \cdot \pi''})|_{\pi'''}$. With induction we get $(t_i[u]_{\pi''' \cdot \pi''})|_{\pi'''} = (t_i|_{\pi'''})[u]_{\pi''} = (t'|_\pi)[u]_{\pi''}$.

If π is not a prefix of π', then we show that $(t'[u]_{\pi'})|_\pi = t'|_\pi$ with trivially gives us $(t|_\pi)[\lambda x.\varphi(\pi \cdot x)]_{\{\pi'' \,:\, \pi \cdot \pi'' \in \Pi'\}}$. With induction on the structure of t':

- $t = x$ for some variable x; this means that both π and π' must be ϵ which contradicts the assumption.
- $t = f(t_1, \ldots, t_n)$ for some symbol f and terms $t_1, \ldots, t_n$; there are tree cases:
  - $\pi = \epsilon$ or $\pi' = \epsilon$; which both contradict the assumptions.
  - $\pi = i.\pi''$ and $\pi' = i.\pi'''$ for some index i and positions π'' and π''' such that π'' is not a prefix of π'''; this means that we have that $(t'[u]_{\pi'})|_\pi = f(t_1, \ldots, t_i[\varphi(\pi')]_{\pi'''}, \ldots, t_n)|_\pi = (t_i[\varphi(\pi')]_{\pi'''})|_{\pi''}$. By induction we have that $(t_i[\varphi(\pi')]_{\pi'''})|_{\pi''} = t_i|_{\pi'''}$ and trivially $t_i|_{\pi'''} = t'|_\pi$.
  - $\pi = i.\pi''$ and $\pi' = j.\pi'''$ for some indices i and j and positions π'' and π''' such that $i \neq j$; this means that we have that $(t[\varphi(\pi')]_{\pi'})|_\pi = f(t_1, \ldots, t_j[\varphi(\pi')]_{\pi'''}, \ldots, t_n)|_\pi = t_j|_{\pi''}$. Trivially we have that $t_i|_{\pi'''} = t'|_\pi$.

□

Next, we show that when substituting a head normal form of t by another head normal form of t, you can also just substitute just those parts of these head normal forms that differ.

Lemma E.1.3 *Let t be a term, φ a function mapping positions to terms and Π and Π' sets of positions such that Π is non-overlapping and a subset of $\mathrm{pos}(t)$, $\forall_{\pi \in \Pi \cap \mathrm{pos}(t)}(\varphi(\pi) \in \mathrm{hnf}(t|_\pi))$ and $\forall_{\pi \in \Pi' \cap \mathrm{pos}(t)}(t|_\pi \in \mathrm{hnf}(t|_\pi))$. Then there are φ' and non-overlapping $\Pi'' \subseteq \mathrm{pos}(t)$ such that we have that $\Pi'' \cap \Pi' = \emptyset$ and $t[\varphi]_\Pi = t[\varphi]_{\Pi''}$.*

Proof We prove this with induction on Π. As measure we use the smallest position in $\Pi \cap \Pi'$ where we say a position π is smaller then π' if π contains less indices than π' or they have the same amount but first index in which they differ is smaller for π than for π'. That is, the order on set of positions we use is such that if $\Pi \cap \Pi' = \emptyset$, Π is a smallest element and all other sets Π are ordered according to the smallest position in $\Pi \cap \Pi'$ where those with a larger smallest position are smaller.

If $\Pi \cap \Pi' = \emptyset$ then $\Pi'' = \Pi$ and $\varphi' = \varphi$ satisfies our goal. Otherwise, let π be the smallest position in $\Pi \cap \Pi'$. With induction on t:

- $t = x$; this means that $\pi = \epsilon$ and $\varphi(\pi) = x$ and thus $t[\varphi(\pi)]_\pi = t$. We therefore trivially have that $t[\varphi]_\Pi = t[\varphi]_{\Pi \setminus \{\pi\}}$ and with induction the desired Π'' and φ'.

- $t = f(t_1, \ldots, t_n)$; we have the following two cases:
  - $\pi = \epsilon$; as $t|_\pi = t$ is a head normal form, we have that $\varphi(\pi)$ must be of the form $f(u_1, \ldots, u_n)$. This means that $t[\varphi(\pi)]_\pi = t[u_1]_{\pi\cdot 1} \ldots [u_n]_{\pi\cdot n}$ and thus $t[\varphi]_\Pi = t[\varphi[\pi \cdot 1 \mapsto u_1] \ldots [\pi \cdot n \mapsto u_n]]_{\Pi \cup \{1,\ldots,n\}}$. As $(\Pi \setminus \{\pi\}) \cup \{1, \ldots, n\}$ is smaller than Π, we get Π'' and φ' that satisfy our goal by induction.
  - $\pi = i \cdot \pi'$; this means that $1 \leq i \leq n$. We have that $t[\varphi(\pi)]_\pi = f(t_1, \ldots, t_{i-1}, t_i[\varphi(\pi)]_{\pi'}, t_{i+1}, \ldots, t_n)$, $t_i[\varphi(\pi)]_{\pi'} = t_i[\lambda x.\varphi(i \cdot x)]_{\{\pi'\}}$ and for the latter we have, by induction, an equivalent $t_i[\varphi'']_{\Pi'''}$ for some Π''' and φ'' with $\Pi''' \subseteq \text{pos}(t_i)$ and $\Pi''' \cap \{\pi'' \ : \ i \cdot \pi'' \in \Pi'\} = \emptyset$. We then have a φ''', with $\varphi'''(i.\pi'') = \varphi''(\pi'')$ for all π'' and $\varphi'''(\pi'') = \varphi(\pi'')$ for all positions π'' that do not start with index i, such that $f(t_1, \ldots, t_{i-1}, t_i[\varphi'']_{\Pi'''}, t_{i+1}, \ldots, t_n) = t[\varphi''']_{\{i\cdot\pi'' \,:\, \pi'' \in \Pi'''\}}$. As we have that $t[\varphi''']_{\{i\cdot\pi'' \,:\, \pi'' \in \Pi'''\}}[\varphi]_{\Pi\setminus\{\pi\}}$ is equal to $t[\varphi''']_{\{i\cdot\pi'' \,:\, \pi'' \in \Pi'''\}}[\varphi''']_{\Pi\setminus\{\pi\}}$ (there is no other position in Π that starts with index i), we trivially have that this is equivalent to $t[\varphi''']_{(\Pi\setminus\{\pi\})\cup\{i\cdot\pi'' \,:\, \pi'' \in \Pi'''\}}$. By induction we then get a Π'' and φ' that satisfies our goal.

□

Finally, the following lemma states that there can be no difference between a pattern and a term if the term matches the pattern.

Lemma E.1.1. Let t and u be terms and σ a substitution. If $t = u\sigma$, then $\partial(t, u) = \emptyset$.

Proof To show this we prove, with induction on the structure of u, that if $t = u\sigma$, then $\partial(t, u, \pi) = \emptyset$ for all positions π. From this it trivially follows that also $\partial(t, u) = \emptyset$. The case that $u = x$, for some variable x, is trivially satisfied. Let $u = f(u_1, \ldots, u_n)$ for some function symbol f and terms $u_1, \ldots, u_n$. We have that $t = f(u_1, \ldots, u_n)\sigma = f(u_1\sigma, \ldots, u_n\sigma)$. This means that $\partial(t, u, \pi) = \bigcup_{1\leq i\leq n} \partial(u_i\sigma, u_i, \pi \cdot i)$. As $u_i\sigma = u_i\sigma$ for all i with $1 \leq i \leq n$, we have that $\partial(u_i\sigma, u_i, \pi \cdot i) = \emptyset$ by induction. Therefore, $\partial(t, u, \pi) = \emptyset$ holds as well. □

E.2 Theorem 5.2.1

We must show that, for all term t and u with $u \in \text{rewr}(t)$, we have that $t \to^* u$. We prove this by showing that if $u \in \text{eval}^\alpha(T, t) \cup \text{eval}^\alpha_{\text{h}}(T, t)$, we have that $t \to^* u$ with transfinite induction on α. For the case $\alpha = 0$ we have that $\text{eval}^\alpha(T, t) = \emptyset$ and $\text{eval}^\alpha_{\text{h}}(T, t) = \emptyset$ and thus that the statement is trivially satisfied. For the case $\alpha = \beta + 1$ we have the following cases (note that we trivially have $x \to^* u$ for $u \in \text{rewr}^\alpha(x)$):

- $\text{eval}^\alpha(\mathsf{F}(\pi, \psi), t)$, which is either $\text{eval}^\beta(\psi(\text{hs}(t|_\pi)), t)$ or $\text{eval}^\beta(\psi(\bot), t)$. With induction we trivially have that the statement is satisfied.

- $\mathrm{eval}^\alpha(\mathsf{H}(\Pi, U), t)$, or $\bigcup_{\psi \in \{\psi' \,:\, \forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \,\Rightarrow\, \psi'(\pi) \in \mathrm{rewr}^\beta_{\mathrm{h}}(t|_\pi))\}} \mathrm{eval}^\beta(T, t[\psi]_\Pi)$.
  Let ψ be such that $\forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \;\Rightarrow\; \psi(\pi) \in \mathrm{rewr}^\beta_{\mathrm{h}}(t|_\pi))$. By induction we have that for all $\pi \in \Pi \cap \mathrm{pos}(t)$ we have that $t|_\pi \to^* \psi(\pi)$. Then by Lemma E.1.1 we trivially have that $t \to^* t[\psi]_\Pi$. Finally, with induction we have that $t[\psi]_\Pi \to^* u'$ for all $u' \in \mathrm{eval}^\beta(T, t[\psi]_\Pi)$ and thus trivially $t \to^* u$.

- The proof for $\mathrm{eval}^\alpha(\mathsf{NF}(\Pi, U))$ is the same as for $\mathrm{eval}^\alpha(\mathsf{H}(\Pi, U))$ with $\mathrm{eval}^\beta_{\mathrm{h}}$ replaced by eval^β.

- $\mathrm{eval}^\alpha(\mathsf{T}(R, U), t)$. Assume that there are rules in R that can be applied to t. Then $\mathrm{eval}^\beta(\mathsf{T}(R, T), t) = \bigcup_{l \to r \;\mathbf{if}\; c \in R \,\wedge\, t = l\sigma \,\wedge\, \mathrm{true} \in \mathrm{rewr}(c\sigma)} \mathrm{eval}^\beta(\varsigma(\mathrm{hs}(r\sigma)), r\sigma)$. For each rule $l \to r$ **if** $c \in R$ and substitutions σ with $\mathrm{true} \in \mathrm{rewr}(c\sigma)$ we trivially have that $t \to r\sigma$ and by induction we have that $r\sigma \to^* u$.

  Now assume that there is no rule in R that can be applied to t. This case is trivial.

- $\mathrm{eval}^\alpha(\mathsf{E}, t)$, which is $\{t\}$. We trivially have that $t \to^* t$

- $\mathrm{eval}^\alpha(\mathsf{X}, t)$, which is $\emptyset$ by definition.

The cases for $\mathrm{eval}^\alpha_{\mathrm{h}}$ the similar. The case where α is a limit ordinal is trivially satisfied.

E.3 Theorem 5.2.2

We need to show that $\mathrm{rewr}(t) = \mathrm{rewr}_{\mathrm{s}}(t)$ or, equivalently, $\mathrm{eval}(\varsigma^\perp_\varphi(\mathrm{hs}^\perp(t)), t) = \mathrm{eval}_{\mathrm{s}}(\varsigma^\perp(\mathrm{hs}^\perp(t)), t)$. If $\mathrm{hs}^\perp(t) = \perp$, we get $\mathrm{eval}(\mathsf{E}, t) = \{t\} = \mathrm{eval}_{\mathrm{s}}([], t)$. Otherwise we get $\mathrm{eval}(\varphi(\varsigma(\mathrm{hs}(t))), t) = \mathrm{eval}_{\mathrm{s}}(\varsigma(\mathrm{hs}(t)), t)$. We prove this by showing that $\mathrm{eval}(\varphi(\varsigma), t) = \mathrm{eval}_{\mathrm{s}}(\varsigma, t)$ by transfinite induction on the approximations α of the definitions of eval and $\mathrm{eval}_{\mathrm{s}}$.

For the case $\alpha = 0$ we have that $\mathrm{eval}^\alpha(\varphi(\varsigma), t) = \emptyset = \mathrm{eval}^\alpha_{\mathrm{s}}(\varsigma, t)$. For the case that $\alpha = \beta + 1$ we have the following cases.

- $\varsigma = []$; this means that we get $\mathrm{eval}^\alpha(\varphi([]), t) = \mathrm{eval}^\alpha(\mathsf{E}, t) = \{t\} = \mathrm{eval}^\alpha_{\mathrm{s}}([], t)$.

- $\varsigma = I \rhd \varsigma'$; this means that $\mathrm{eval}^\alpha(\varphi(I \rhd \varsigma'), t) = \mathrm{eval}^\alpha(\mathsf{NF}(I, \varphi(\varsigma')), t) = \bigcup_{\varphi \in \mathrm{rewrf}^\beta(t, I)} \mathrm{eval}^\beta(\varphi(\varsigma'), t[\varphi]_I)$ and also that we have $\mathrm{eval}^\alpha_{\mathrm{s}}(I \rhd \varsigma', t) = \bigcup_{\psi \in \mathrm{rewrf}^\beta_{\mathrm{s}}(t, I)} \mathrm{eval}^\beta_{\mathrm{s}}(\varsigma', t[\psi]_I)$. By induction we have that $\mathrm{eval}^\beta(\varphi(\varsigma'), t[\varphi]_I) = \mathrm{eval}^\beta_{\mathrm{s}}(\varsigma', t[\psi]_I)$ for all ψ, so it remains to show that $\mathrm{rewrf}^\beta(t, I) = \mathrm{rewrf}^\beta_{\mathrm{s}}(t, I)$, which is trivial.

- $\varsigma = R \rhd \varsigma'$; this means that we get $\mathrm{eval}^\alpha(\varphi(R \rhd \varsigma'), t) = \mathrm{eval}^\alpha(\mathsf{T}(R, \varphi(\varsigma')), t)$. Now assume that $\mathrm{app}^\beta(R, t) = \emptyset$. In that case we have $\mathrm{eval}^\alpha(\mathsf{T}(R, \varphi(\varsigma')), t) = \mathrm{eval}^\beta(\varphi(\varsigma'), t)$ and the by induction equivalent $\mathrm{eval}^\alpha_{\mathrm{s}}(R \rhd \varsigma', t) = \mathrm{eval}^\beta_{\mathrm{s}}(\varsigma', t)$.

If $\mathrm{app}^\beta(R,t) \neq \emptyset$, then $\mathrm{eval}^\alpha(\mathsf{T}(R,\varphi(\varsigma')),t) = \bigcup_{u\in\mathrm{app}^\beta(R,t)} \mathrm{rewr}^\beta(u) = \bigcup_{u\in\mathrm{app}^\beta(R,t)} \mathrm{eval}^\beta(\varsigma^\perp_\varphi(\mathrm{hs}^\perp(u)),u)$ and on the other hand $\mathrm{eval}^\alpha_\mathrm{s}(R \rhd \varsigma', t) = \bigcup_{u\in\mathrm{app}^\beta(R,t)} \mathrm{rewr}^\beta_\mathrm{s}(u) = \bigcup_{u\in\mathrm{app}^\beta(R,t)} \mathrm{eval}^\beta_\mathrm{s}(\varsigma^\perp(\mathrm{hs}^\perp(u)),u)$. We trivially have that $\mathrm{app}^\beta(R,t) = \mathrm{app}^\beta(R,t)$ and by induction that $\mathrm{eval}^\beta(\varsigma^\perp_\varphi(\mathrm{hs}^\perp(u)),u) = \mathrm{eval}^\beta_\mathrm{s}(\varsigma^\perp(\mathrm{hs}^\perp(u)),u)$.

The case that α is a limit ordinal is trivial. This concludes the proof.

E.4 Theorem 5.2.8

We must prove that, given a strategy $\langle\varsigma,\varsigma_\mathrm{h}\rangle$ where all T nodes of all trees $\varsigma(f)$ and $\varsigma(f)$ (for all symbols f) have a set with at most one rewrite rule, $\mathrm{rewr}(t)$ and $\mathrm{rewr}_\mathrm{h}(t)$ have at most one element (for all terms t). We prove this using transfinite induction on approximation α of rewr and rewr_h.

For $\alpha = 0$ we trivially have that rewr and rewr_h return sets with at most one element. For the case that α is a limit ordinal we trivially have that there is at most one element by induction and the fact that $\mathrm{eval}^\alpha(T,t) \subseteq \mathrm{eval}^\beta(T,t)$ (and similarly for $\mathrm{eval}^\alpha_\mathrm{h}$) when $\alpha < \beta$ and for all strategy trees T and terms t.

What remains is the case that $\alpha = \beta + 1$. We have the following cases of $\mathrm{eval}^\alpha(T,t)$. Note that we only consider eval^α here; the cases for $\mathrm{eval}^\alpha_\mathrm{h}$ are very similar.

- If $T = \mathsf{X}$ or $T = \mathsf{E}$, then we have that there is at most one element per definition.
- If $T = \mathsf{F}(\pi,\varphi)$ with position π and function φ of function symbols to strategy trees, then, by induction, we trivially have there is at most one element.
- If $T = \mathsf{H}(\Pi,U)$ with set of positions Π and strategy tree U, then we are done (by induction) if we can show that for all $\varphi,\varphi' \in \mathrm{rewrf}^\alpha_\mathrm{h}(t,\Pi)$ it holds that $t[\varphi]_\Pi = t[\varphi']_\Pi$. We observe that in terms like $t[\varphi]_\Pi$ only $\varphi(\pi)$ is used if $\pi \in \Pi \cap \mathrm{pos}(t)$. That is, we need to show that $\varphi(\pi) = \varphi'(\pi)$ for all $\pi \in \Pi \cap \mathrm{pos}(t)$. Per definition we have that both $\varphi(\pi)$ and $\varphi'(\pi)$ are elements of $\mathrm{rewr}^\alpha_\mathrm{h}(t|_\pi)$. By induction we have that the latter has at most one element. Therefore $\varphi(\pi)$ and $\varphi'(\pi)$ must be the same element.
- The case for $T = \mathsf{NF}(\Pi,U)$, with set of positions Π and strategy tree U, is similar to the one above.
- Finally, if $T = \mathsf{T}(R,U)$, for some set of rewrite rules R and strategy tree U, we have two cases. Either $\mathrm{app}^\alpha(R,t)$ is empty or it is not. In the first case we done trivially by induction. Otherwise, we know that $\mathrm{app}^\alpha(R,t)$ contains at most one element because of the definition of app and the assumption that R contains at most one rule. With induction, this trivially means that also in this case there is at most one element in the resulting set.

E.5 Theorem 5.3.1

We must show that if a strategy is thorough, rewriting a term t with rewr or rewr_h results in a set of normal form, respectively head normal forms. Note that we cannot use the same approach as with just-in-time strategies because repeated passes through a tree may result in different paths (due to the possibility to let the strategy depend on a head symbol of a subterm).

We prove this together with the invariance of $P(T,t) = \exists_{S,R,\Pi}(t \in S \ \wedge P_1(t,R) \wedge P_2(t,\Pi) \wedge \text{thrgh}(T,S,R,\Pi))$ and $P_\text{h}(T,t) = \exists_{S,R,\Pi}(t \in S \wedge P_1(t,R) \wedge P_2(t,\Pi) \ \wedge \ \text{thrgh}_\text{h}(T,S,R,\Pi))$, where $P_1(t,R) = \neg\exists_{l\to r \textbf{ if } c\in R,\sigma}(t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma))$ and $P_2(t,\Pi) = \forall_{\pi\in\Pi\cap\text{pos}(t)}(t|_\pi \in \text{hnf}(t|_\pi))$, with respect to eval and eval_h, respectively. Instead of showing that $\text{rewr}(t) \subseteq \text{nf}(t)$ and $\text{rewr}_\text{h}(t) \subseteq \text{hnf}(t)$ hold we will prove the more general statements $\text{eval}(T,t) \subseteq \text{nf}(t)$ if $P(T,t)$ and $\text{eval}(T,t) \subseteq \text{hnf}(t)$ if $P_\text{h}(T,t)$. We use ordinal induction on the approximation α of the definition of eval and eval_h.

For the case that $\alpha = 0$ we have that $\text{eval}(T,t) = \emptyset$ and $\text{eval}_\text{h}(T,t) = \emptyset$. This trivially satisfies our goal.

We consider the case that $\alpha = \beta + 1$ with case analysis on the structure of T. First we consider the invariance of $P(T,t)$ and the statement $\text{eval}(T,t) \subseteq \text{nf}(t)$ if $P(T,t)$. For each case below, take a S, R and Π such that $t \in S$, $P_1(t,R)$, $P_2(t,\Pi)$ and $\text{thrgh}(T,S,R,\Pi)$. Note that the invariants trivially hold for occurrences of rewr^β and $\text{rewr}_\text{h}^\beta$ due to the completeness of the strategy.

- $T = \mathsf{F}(\pi,\psi)$, for some position π and function of symbols to strategy trees ψ; this means that we have that forall f, $\text{thrgh}(\psi(f), \{u \in S \ : \ \pi \in \text{pos}_\text{f}(t) \ \wedge \ \text{hs}(u|_\pi) = f\}, R \cup \{l \to r \textbf{ if } c \in R_f \ : \ \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \wedge \text{hs}(l\sigma|_\pi) = f)\}, \Pi)$ and that $\text{thrgh}(\psi(\bot), \{u \in S \ : \ \pi \notin \text{pos}_\text{f}(u)\}, R \cup \{l \to r \textbf{ if } c \in R_f \ : \ \pi \in \text{pos}_\text{f}(l)\}, \Pi \cup \{\pi\})$.

  Assume that $\pi \in \text{pos}_\text{f}(t)$. Then we have that $\text{thrgh}(\psi(\text{hs}(t|_\pi)), \{u \in S \ : \ \pi \in \text{pos}_\text{f}(t) \ \wedge \ \text{hs}(u|_\pi) = \text{hs}(t|_\pi)\}, R \cup \{l \to r \textbf{ if } c \in R_f \ : \ \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \ \wedge \ \text{hs}(l\sigma|_\pi) = \text{hs}(t|_\pi)\}, \Pi)$. We have that $P_1(t, R \cup \{l \to r \textbf{ if } c \in R_f \ : \ \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \wedge \text{hs}(l\sigma|_\pi) = \text{hs}(t|_\pi)\})$ is equivalent to $P_1(t,R) \wedge P_1(t, \{l \to r \textbf{ if } c \in R_f \ : \ \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \wedge \text{hs}(l\sigma|_\pi) = \text{hs}(t|_\pi)\})$ and know that $P_1(t,R)$ holds. To show that the other conjunct also holds we must show that $\neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \ \wedge \ \text{hs}(l\sigma|_\pi) = \text{hs}(t|_\pi)) = \forall_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \ \Rightarrow \ \text{hs}(l\sigma|_\pi) \neq \text{hs}(t|_\pi))$ implies that $\neg\exists_\sigma(t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma))$ for all rewrite rules $l \to r$ **if** c in R_f. If there would be a σ with $t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma)$, then we trivially get that both $\pi \in \text{pos}_\text{f}(l\sigma)$, $l\sigma|_\pi = t|_\pi$ and thus $\text{hs}(l\sigma|_\pi) = \text{hs}(t|_\sigma)$. This contradict with the antecedent of the implication and thus we have that $P_1(r, R \cup \{l \to r \textbf{ if } c \in R_f \ : \ \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \ \wedge \ \text{hs}(l\sigma|_\pi) = \text{hs}(t|_\pi)\})$. As t is trivially in $S \cap \{u \ : \ \text{hs}(u|_\pi) = \text{hs}(t|_\pi)\}$ and we still have $P_2(t,\Pi)$, this means that $P(\psi(\text{hs}(t)), t)$. By induction this trivially gives us that $\text{eval}^\beta(\psi(\text{hs}(t)), t) \subseteq \text{nf}(t)$. But then also $\text{eval}^\alpha(T,t) \subseteq \text{nf}(t)$.

  Now assume that $\pi \notin \text{pos}_\text{f}(t)$. This means we have that $\text{thrgh}(\psi(\bot)), \{u \in$

$S : \pi \notin \mathrm{pos_f}(u)\}, R, \Pi \cup \{\pi\})$. It is clear that $t \in \{u \in S : \pi \notin \mathrm{pos_f}(u)\}$ and for $P_2(t, \Pi \cup \{\pi\})$ we must show that if $\pi \in \mathrm{pos}(t)$ then also $t|_\pi \in \mathrm{hnf}(t|_\pi)$. As $\pi \notin \mathrm{pos_f}(t)$, we have that this means that we must show that if $\pi \in \mathrm{pos_v}(t)$ (i.e. $t|_\pi$ is a variable), then $t|_\pi$ is in head normal form, which holds trivially. Together with $P_1(t, R)$, we trivially get that $P(\psi(\bot), t)$ holds. Then, by induction, we get that $\mathrm{eval}^\beta(\psi(\bot), t) \subseteq \mathrm{nf}(t)$. But then also $\mathrm{eval}^\alpha(T, t) \subseteq \mathrm{nf}(t)$.

- $T = \mathsf{H}(\Pi', U)$, for some set of position Π' and strategy tree U; this means that we have that $\mathrm{thrgh}(U, \{u[\psi]_{\Pi'} : u \in S \wedge \forall_{\pi \in \Pi'}(\pi \in \mathrm{pos}(t) \Rightarrow \psi(\pi) \in \mathrm{rewr_h}(t|_\pi))\}, \{\rho \in R : \forall_{u \in S}(\partial(u, \rho) \setminus \Pi' \neq \emptyset) \vee \Pi' \cap \mathrm{esspos}(\rho) \subseteq \Pi\}, (\Pi \cup \Pi') \setminus \{\pi \cdot i \cdot \pi' :; \pi \in \Pi'\})\}$. Let $\psi' \in \{\psi'' : \forall_{\pi \in \Pi'}(\psi''(\pi) \in \mathrm{rewr_h}(t|_\pi)\}$.

  We trivially have that $t[\psi']_{\Pi'} \in \{u[\psi]_{\Pi'} : u \in S \wedge \forall_{\pi \in \Pi'}(\psi(\pi) \in \mathrm{rewr_h}(t|_\pi))\}$ for all ψ' with $\forall_{\pi \in \Pi'}(\psi'(\pi) \in \mathrm{rewr_h}(t|_\pi))$.

  From $P_1(t, R)$ it follows that $\neg\exists_{l \to r \textbf{ if } c \in R, \sigma}(t = l\sigma \wedge \mathrm{true} \in \mathrm{rewr}(c\sigma))$. We must show that $P_1(t[\psi']_{\Pi'}, \{\rho \in R : \forall_{u \in S}(\partial(u, \rho) \setminus \Pi' \neq \emptyset) \vee \Pi' \cap \mathrm{esspos}(\rho) \subseteq \Pi\})$, which is equivalent to $\neg\exists_{l \to r \textbf{ if } c \in R, \sigma}((\forall_{u \in S}(\partial(u, l) \setminus \Pi' \neq \emptyset) \vee \Pi' \cap \mathrm{esspos}(l \to r \textbf{ if } c) \subseteq \Pi) \wedge t[\psi']_{\Pi'} = l\sigma \wedge \mathrm{true} \in \mathrm{rewr}(c\sigma))$ and $\forall_{l \to r \textbf{ if } c \in R, \sigma}(t[\psi']_{\Pi'} = l\sigma \wedge \mathrm{true} \in \mathrm{rewr}(c\sigma) \Rightarrow (\exists_{u \in S}(\partial(u, l) \setminus \Pi' = \emptyset) \wedge \Pi' \cap \mathrm{esspos}(l \to r \textbf{ if } c) \not\subseteq \Pi))$. Let $l \to r \textbf{ if } c \in R$ and σ be a substitution such that $t[\psi']_{\Pi'} = l\sigma$ and $\mathrm{true} \in \mathrm{rewr}(c\sigma)$. From $P(t, R)$ we get that there is no substitution τ such that $t = l\tau$ and $\mathrm{true} \in \mathrm{rewr}(c\tau)$. Assume that $\Pi' \cap \mathrm{esspos}(l \to r \textbf{ if } c) \subseteq \Pi$. Then, as $t[\psi']_{\Pi'} = t[\psi']_{\Pi'' \setminus \Pi}$ by Lemma E.1.3 for non-overlapping Π'' (Lemma B.1.1) and $(\Pi'' \setminus \Pi) \cap \mathrm{esspos}(l \to r \textbf{ if } c) = \emptyset$, from Theorem 2.1.2 it follows that there must be a τ such that $t = l\tau$. From $P_1(t, R)$ we then get that $\mathrm{true} \notin \mathrm{rewr}(c\tau)$. As we have that for all $\pi \in \mathrm{pos_v}(l)$ with $l|_\pi \in \mathrm{var}(c)$ that $\pi \in \mathrm{esspos}(l \to r \textbf{ if } c)$ and thus $\pi \notin \Pi'$, we have that $c\tau = c\sigma$ and thus $\mathrm{true} \in \mathrm{rewr}(c\sigma)$. The latter is in contradiction with the $\mathrm{true} \notin \mathrm{rewr}(c\sigma)$ we got before and therefore that $\Pi' \cap \mathrm{esspos}(l \to r \textbf{ if } c) \not\subseteq \Pi)$. This leaves us to show that $\exists_{u \in S}(\partial(u, l) \setminus \Pi' = \emptyset)$. This follows trivially from the fact that $t \in S$ and, as $t = l\sigma$, $\partial(t, l) = \emptyset$ by Lemma E.1.1. We therefore we know that $P_1(t[\psi']_{\Pi'}, \{\rho \in R : \forall_{u \in S}(\partial(u, \rho) \setminus \Pi' \neq \emptyset) \vee \Pi' \cap \mathrm{esspos}(\rho) \subseteq \Pi\})$.

  Finally we must show that $P_2(t[\psi']_{\Pi'}, (\Pi \cup \Pi') \setminus \{\pi \cdot 1 \cdot \pi' : \pi \in \Pi'\})$, which is equivalent to $P_2(t[\psi']_{\Pi'}, \Pi \setminus \overline{(\Pi')}) \wedge P_2(t[\psi']_{\Pi'}, \Pi' \setminus \{\pi \cdot 1 \cdot \pi' : \pi \in \Pi\})$. The latter conjunct is trivially satisfied by induction. For the former conjunct we must show that for all $\pi \in \Pi \setminus \overline{\Pi'}$ we have that $t[\psi']_{\Pi'}|_\pi$ is a head normal form. From $P_2(t, \Pi)$ it follows that $t|_\pi$ is a head normal form and per definition we trivially have that if $t|_\pi \to^* u$ means that u is a head normal form as well. As we have that $t|_{\pi'} \to^* \psi'(\pi')$ for all $\pi' \in \Pi'$ by Theorem 5.2.1, Lemma E.1.2 and Lemma E.1.1 give us that $t[\psi']_{\Pi'}|_\pi = t|_\pi[\lambda x.\psi'(\pi \cdot x)]_{\{\pi' \,:\, \pi \cdot \pi' \in \Pi'\}}$ and $t|_\pi \to^* t|_\pi[\lambda x.\psi'(\pi \cdot x)]_{\{\pi' \,:\, \pi \cdot \pi' \in \Pi'\}}$.

  Thus, as $P(T, t[\psi']_{\Pi'})$, we get that $\mathrm{eval}^\beta(T, t[\psi']_{\Pi'}) \subseteq \mathrm{nf}(t[\psi']_{\Pi'})$ by induc-

tion. As Theorem 5.2.1 gives us that $t \to^* t[\psi']_{\Pi'}$, this concludes the proof of this case.

- $T = \mathsf{NF}(\Pi', U)$, for some set of position Π' and strategy tree U; this case is the very similar to the case of $T = \mathsf{H}(\Pi', U)$.
- $T = \mathsf{T}(R', U)$, for some set of rewrite rules R' and strategy tree U; this means we have that $\mathrm{thrgh}(U, S \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}, R \cup R', \Pi)$. In the case that there are rewrite rules in R that can be applied to t, we trivially have that the invariant is satisfied. Otherwise, we have that there are no $l \to r$ **if** $c \in R'$ and σ such that $t = l\sigma$ and $\mathrm{true} \in \mathrm{rewr}(c\sigma)$. This trivially means that $t \in S \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}$ and $P_1(t, R \cup R')$. As we already had that $P_2(t, \Pi)$, the invariant is also satisfied in this case.
- $T = \mathsf{E}$; in this case the invariant is trivially satisfied due to the absence of eval/$\mathrm{eval_h}$ on the right-hand side of the definition. This means we must show that if $P(\mathsf{E}, t)$, then $t \in \mathrm{nf}(t)$. From $P(\mathsf{E}, t)$ it follows that $\mathrm{thrgh}(\mathsf{E}, S, R, \Pi) = \forall_{u \in S}(\mathrm{pos}(u) \subseteq \Pi \cup \{\epsilon\}) \ \wedge \ (S \neq \emptyset \ \Rightarrow \ R_f \subseteq R)$. The first conjunct together with $P_2(t, \Pi)$ gives us that for all positions $\pi \in \mathrm{pos}(t) \setminus \{\epsilon\}$ we have that $t|_\pi \in \mathrm{hnf}(t)$. With Theorem 2.1.1 this means that for all $i \in \mathrm{pos}(t)$ we have that $t|_i \in \mathrm{nf}(t)$. This, together with $P_1(t, R)$, $R_f \subseteq R$ (as $t \in S$ and thus $S \neq \emptyset$) and $\mathrm{hs}(t) = f$, gives us that $t \in \mathrm{nf}(t)$.
- $T = \mathsf{X}$; in this case the invariant is trivially satisfied due to the absence of eval/$\mathrm{eval_h}$ on the right-hand side of the definition. Also, as $\mathrm{eval}^\alpha(\mathsf{X}, t) = \emptyset$, we trivially have that $\mathrm{eval}^\alpha(T, t) \subseteq \mathrm{nf}(t)t$.

The cases for the invariance of $P_\mathrm{h}(T, t)$ and the statement $\mathrm{eval_h}(T, t) \subseteq \mathrm{hnf}(t)$ if $P_\mathrm{h}(T, t)$ are very similar. The only significant difference is in the case that $T = \mathsf{E}$. In that case the invariant is still trivially satisfied. For $\mathrm{eval_h}(T, t) \subseteq \mathrm{hnf}(t)$ if $P_\mathrm{h}(T, t)$ we have that if $P_\mathrm{h}(T, t)$, then also $\forall_{t \in S, u}(t \to^* u \ \Rightarrow \ \neg\exists_{l \to r \ \mathbf{if} \ c \in R_{\mathrm{hs}(u)}, \sigma}(u = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)))$ as all essential positions of relevant rewrite rules are in head normal form and are therefore the same in u as in t. This trivially means that t is a head normal form.

The case that α is a limit ordinal is trivial.

E.6 Theorem 5.3.2

We must show that if a strategy is thorough and we have that rewr or rewr_h return a empty set for a term t, then t must have a infinite reduction (i.e. $t \to^\omega$).

Let φ be defined by $\varphi(t, S) = (S = \emptyset) \ \Rightarrow \ t \to^\omega$. Then we are interested in $\psi(T, t) = \varphi(t, \mathrm{eval}(T, t))$ and $\psi_\mathrm{h}(T, t) = \varphi(t, \mathrm{eval_h}(T, t))$. We derive (in)equalities from these statements such that we can easily construct an equation system of which these (in)equalities show that ψ and ψ_h are a solution. In these derivations

we assume that $P(T,t)$ (Section E.5) holds as we are only interested in such cases. We then show that the minimal solution of this equation system is true, which means that $\psi(T,t)$ and $\psi_{\mathrm{h}}(T,t)$ hold.

Note that $\Rightarrow$ corresponds to the order on booleans. That is, $p \Rightarrow q$ if, and only if, $p \leq q$. Also note that φ is monotone in S.

With case analysis on T:

- Assume that $T = \mathsf{F}(\pi, \psi')$ for some position $\pi \in \mathrm{pos}_{\mathrm{f}}(t)$ and function ψ' of positions to terms.

$$\begin{aligned}
&\psi(\mathsf{F}(\pi,\psi'),t)\\
=\;&\\
&\varphi(t,\mathrm{eval}(\mathsf{F}(\pi,\psi'),t))\\
\geq\;&\\
&\varphi(t,\mathrm{eval}(\psi'(\mathrm{hs}(t|_\pi)),t))\\
=\;&\\
&\psi(\psi'(\mathrm{hs}(t|_\pi)),t)
\end{aligned}$$

  If $\pi \notin \mathrm{pos}_{\mathrm{f}}(t)$ we get the following.

$$\begin{aligned}
&\psi(\mathsf{F}(\pi,\psi'),t)\\
=\;&\\
&\varphi(t,\mathrm{eval}(\mathsf{F}(\pi,\psi'),t))\\
\geq\;&\\
&\varphi(t,\mathrm{eval}(\psi'(\bot),t))\\
=\;&\\
&\psi(\psi'(\bot),t)
\end{aligned}$$

- Assume that $T = \mathsf{H}(\Pi, U)$ for some set of positions Π and strategy tree U.

$$\begin{aligned}
&\psi(\mathsf{H}(\Pi,U),t)\\
=\;&\\
&\varphi(t,\mathrm{eval}(\mathsf{H}(\Pi,U),t))\\
\geq\;&\\
&\varphi(t,\textstyle\bigcup_{\psi''\in\{\psi' \,:\, \forall_{\pi\in\Pi}(\pi\in\mathrm{pos}(t)\,\Rightarrow\,\psi'(\pi)\in\mathrm{rewr}_{\mathrm{h}}(t|_\pi))\}}\mathrm{eval}(U,t[\psi'']_\Pi))\\
=\;&\\
&(\textstyle\bigcup_{\psi''\in\{\psi' \,:\, \forall_{\pi\in\Pi}(\pi\in\mathrm{pos}(t)\,\Rightarrow\,\psi'(\pi)\in\mathrm{rewr}_{\mathrm{h}}(t|_\pi))\}}\mathrm{eval}(U,t[\psi'']_\Pi)) = \emptyset \;\Rightarrow\\
&\quad t\rightarrow^{\omega}\\
=\;&
\end{aligned}$$

$$
\begin{aligned}
& \forall_{\psi'' \in \{\psi' \,:\, \forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \,\Rightarrow\, \psi'(\pi) \in \mathrm{rewr_h}(t|_\pi))\}} (\mathrm{eval}(U, t[\psi'']_\Pi) = \emptyset) \;\Rightarrow \\
& \quad t \rightarrow^\omega \\
= \\
& \exists_{\psi'' \in \{\psi' \,:\, \forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \,\Rightarrow\, \psi'(\pi) \in \mathrm{rewr_h}(t|_\pi))\}} (\mathrm{eval}(U, t[\psi'']_\Pi) = \emptyset \;\Rightarrow \\
& \quad t \rightarrow^\omega) \vee \\
& (\{\psi' \,:\, \forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \;\Rightarrow\; \psi'(\pi) \in \mathrm{rewr_h}(t|_\pi))\} = \emptyset \;\wedge\; t \rightarrow^\omega) \\
= \\
& \exists_{\psi'' \in \{\psi' \,:\, \forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \,\Rightarrow\, \psi'(\pi) \in \mathrm{rewr_h}(t|_\pi))\}} (\mathrm{eval}(U, t[\psi'']_\Pi) = \emptyset \;\Rightarrow \\
& \quad t \rightarrow^\omega) \vee \\
& (\forall_{\psi'}(\neg \forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \;\Rightarrow\; \psi'(\pi) \in \mathrm{rewr_h}(t|_\pi))) \;\wedge\; t \rightarrow^\omega) \\
= \\
& \exists_{\psi'' \in \{\psi' \,:\, \forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \,\Rightarrow\, \psi'(\pi) \in \mathrm{rewr_h}(t|_\pi))\}} (\mathrm{eval}(U, t[\psi'']_\Pi) = \emptyset \;\Rightarrow \\
& \quad t \rightarrow^\omega) \vee \\
& (\forall_{\psi'}(\exists_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \;\wedge\; \psi'(\pi) \notin \mathrm{rewr_h}(t|_\pi)) \;\wedge\; t \rightarrow^\omega) \\
= \quad & \{\;\} \\
& \exists_{\psi'' \in \{\psi' \,:\, \forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \,\Rightarrow\, \psi'(\pi) \in \mathrm{rewr_h}(t|_\pi))\}} (\mathrm{eval}(U, t[\psi'']_\Pi) = \emptyset \;\Rightarrow \\
& \quad t \rightarrow^\omega) \vee \\
& (\exists_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \;\wedge\; \mathrm{rewr_h}(t|_\pi) = \emptyset) \;\wedge\; t \rightarrow^\omega) \\
= \\
& \exists_{\psi'' \in \{\psi' \,:\, \forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \,\Rightarrow\, \psi'(\pi) \in \mathrm{rewr_h}(t|_\pi))\}} (\mathrm{eval}(U, t[\psi'']_\Pi) = \emptyset \;\Rightarrow \\
& \quad t \rightarrow^\omega) \vee \\
& \exists_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \;\wedge\; \mathrm{rewr_h}(t|_\pi) = \emptyset \;\wedge\; t \rightarrow^\omega) \\
\Leftarrow \\
& \exists_{\psi'' \in \{\psi' \,:\, \forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \,\Rightarrow\, \psi'(\pi) \in \mathrm{rewr_h}(t|_\pi))\}} (\mathrm{eval}(U, t[\psi'']_\Pi) = \emptyset \;\Rightarrow \\
& \quad (t \rightarrow^* t[\psi'']_\Pi \;\wedge\; t[\psi'']_\Pi \rightarrow^\omega)) \vee \\
& \exists_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \;\wedge\; \mathrm{eval_h}(\mathrm{hs}(t|_\pi), t|_\pi) = \emptyset \;\wedge\; t|_\pi \rightarrow^\omega) \\
= \quad & \{\;\} \\
& \exists_{\psi'' \in \{\psi' \,:\, \forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \,\Rightarrow\, \psi'(\pi) \in \mathrm{rewr_h}(t|_\pi))\}} (\mathrm{eval}(U, t[\psi'']_\Pi) = \emptyset \;\Rightarrow \\
& \quad t[\psi'']_\Pi \rightarrow^\omega) \vee \\
& \exists_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \;\wedge\; \mathrm{eval_h}(\mathrm{hs}(t|_\pi), t|_\pi) = \emptyset \;\wedge\; t|_\pi \rightarrow^\omega) \\
= \\
& \exists_{\psi'' \in \{\psi' \,:\, \forall_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \,\Rightarrow\, \psi'(\pi) \in \mathrm{rewr_h}(t|_\pi))\}} (\psi(U, t[\psi'']_\Pi)) \vee \\
& \exists_{\pi \in \Pi}(\pi \in \mathrm{pos}(t) \;\wedge\; \psi_\mathrm{h}(\varsigma(\mathrm{hs}(t|_\pi)), t|_\pi))
\end{aligned}
$$

Note that there is no Π such that the last expression is equivalent to false because this would mean that there are no positions in $\Pi \cap \mathrm{pos}(t)$ (second

disjunct) and in that case the set of ψ's is the set of all functions.

- Assume that $T = \mathsf{NF}(\Pi, U)$ for some set of positions Π and strategy tree U. With a similar derivation as for the case of $\psi(\mathsf{H}(\Pi, U), t)$ we get that $\psi(\mathsf{NF}(\Pi, U), t)$ is implied by $\exists_{\psi'' \in \{\psi' \,:\, \forall_{\pi \in \Pi}(\psi'(\pi) \in \mathrm{rewr}(t|_\pi))\}}(\psi(U, t[\psi'']_\Pi)) \;\vee\; \exists_{\pi \in \Pi}(\psi(\varsigma(\mathrm{hs}(t|_\pi)), t|_\pi))$.
- Assume that $T = \mathsf{T}(R, U)$ for some set of rewrite rules R and strategy tree U. Also assume that there is a rule in R that can be applied to t.

$$
\begin{aligned}
& \psi(\mathsf{T}(R, U), t) \\
= \;& \\
& \varphi(t, \mathrm{eval}(\mathsf{T}(R, U), t)) \\
\geq \;& \\
& \varphi(t, \textstyle\bigcup_{l \to r \;\mathbf{if}\; c \in R \,\wedge\, t = l\sigma \,\wedge\, \mathrm{true} \in \mathrm{rewr}(c\sigma)} \mathrm{rewr}(r\sigma)) \\
= \;& \\
& \textstyle\bigcup_{l \to r \;\mathbf{if}\; c \in R \,\wedge\, t = l\sigma \,\wedge\, \mathrm{true} \in \mathrm{rewr}(c\sigma)} \mathrm{rewr}(r\sigma) = \emptyset \;\Rightarrow\; t \to^\omega \\
= \;& \\
& \forall_{l \to r \;\mathbf{if}\; c \in R \,\wedge\, t = l\sigma \,\wedge\, \mathrm{true} \in \mathrm{rewr}(c\sigma)} (\mathrm{rewr}(r\sigma) = \emptyset) \;\Rightarrow\; t \to^\omega \\
= \;& \\
& \exists_{l \to r \;\mathbf{if}\; c \in R \,\wedge\, t = l\sigma \,\wedge\, \mathrm{true} \in \mathrm{rewr}(c\sigma)} (\mathrm{rewr}(r\sigma) = \emptyset \;\Rightarrow\; t \to^\omega) \\
= \;& \\
& \exists_{l \to r \;\mathbf{if}\; c \in R \,\wedge\, t = l\sigma \,\wedge\, \mathrm{true} \in \mathrm{rewr}(c\sigma)} (\mathrm{rewr}(r\sigma) = \emptyset \;\Rightarrow\; (t \to r\sigma \;\wedge\; r\sigma \to^\omega)) \\
= \;& \\
& \exists_{l \to r \;\mathbf{if}\; c \in R \,\wedge\, t = l\sigma \,\wedge\, \mathrm{true} \in \mathrm{rewr}(c\sigma)} (\mathrm{rewr}(r\sigma) = \emptyset \;\Rightarrow\; r\sigma \to^\omega) \\
= \;& \\
& \exists_{l \to r \;\mathbf{if}\; c \in R \,\wedge\, t = l\sigma \,\wedge\, \mathrm{true} \in \mathrm{rewr}(c\sigma)} (\mathrm{eval}(\varsigma(\mathrm{hs}(r\sigma)), r\sigma) = \emptyset \;\Rightarrow\; r\sigma \to^\omega) \\
= \;& \\
& \exists_{l \to r \;\mathbf{if}\; c \in R \,\wedge\, t = l\sigma \,\wedge\, \mathrm{true} \in \mathrm{rewr}(c\sigma)} (\psi(\varsigma(\mathrm{hs}(r\sigma)), r\sigma))
\end{aligned}
$$

Note that there is no R such that the last expression is equivalent to false because of the side condition for this case. Now assume there is no such rule.

$$
\begin{aligned}
& \psi(\mathsf{T}(R, U), t) \\
= \;& \\
& \varphi(t, \mathrm{eval}(\mathsf{T}(R, U), t)) \\
\geq \;& \\
& \varphi(t, \mathrm{eval}(U, t)) \\
= \;&
\end{aligned}
$$

$$\psi(U,t)$$

- Assume that $T = \mathsf{E}$.

$$\begin{array}{ll} & \psi(\mathsf{E},t) \\ = & \\ & \varphi(t,\mathrm{eval}(\mathsf{E},t)) \\ \geq & \\ & \varphi(t,\{t\}) \\ = & \\ & \{t\} = \emptyset \;\Rightarrow\; t \rightarrow^{\omega} \\ = & \\ & \mathrm{true} \end{array}$$

- Assume that $T = \mathsf{X}$.

$$\begin{array}{ll} & \psi(\mathsf{X},t) \\ = & \\ & \varphi(t,\mathrm{eval}(\mathsf{E},t)) \\ \geq & \\ & \varphi(t,\emptyset) \\ = & \\ & \emptyset = \emptyset \;\Rightarrow\; t \rightarrow^{\omega} \\ = & \\ & t \rightarrow^{\omega} \\ = & \quad \{\; P(T,t) \;\} \\ & \mathrm{true} \end{array}$$

In a similar way we can derive the (also similar) inequalities for ψ_{h}.

Now take a boolean equation system $\mathcal{E}$ with inequalities that correspond directly with the inequalities we derived but with all occurrences of $\psi(T,t)$ replaced by $X(T,t)$ and all occurrences of $\psi_{\mathrm{h}}(T,t)$ by $X_{\mathrm{h}}(T,t)$. Clearly ψ and ψ_{h} are valid solutions for X and X_{h}, respectively.

As the inequalities of $\mathcal{E}$ are built of only true and non-empty disjunctions, we trivially have that the minimal solution for both X and X' is the function $\lambda T, t.\mathrm{true}$. We trivially have that the minimal solution of $\mathcal{E}$ smaller than any other solution and as $\lambda T, t.\mathrm{true}$ is the biggest possible function, we know that ψ and ψ_{h} are also equal to $\lambda T, t.\mathrm{true}$. This concludes this proof.

E.7 Theorem 5.3.4

We must show that if a sequential strategy is full and in-time (see Section B.1 for a more formal definition), that its translation with φ (per Theorem 5.2.2) results in a thorough strategy. We do this by showing that for all s' and s'' with $s = s' +\!\!+ s''$ we have that $\mathrm{thrgh}(\varphi(s''), \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\}, \psi_\mathrm{r}(s'), \overline{\psi_\mathrm{i}(s')})$ with induction on the size of s''.

- If $s'' = []$, then we have that $s' = s$ and thus that $\psi_\mathrm{r}(s') = R_f$ and $\psi_\mathrm{i}(s') = \{1, \ldots, \mathrm{ar}(f)\}$. This means we get the following.

$$
\begin{aligned}
& \mathrm{thrgh}(\varphi([]), \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\}, R_f, \overline{\{1, \ldots, \mathrm{ar}(f)\}}) \\
= \\
& \forall_{t \in \{f(t_1, \ldots, t_{\mathrm{ar}(f)}\}}(\mathrm{pos}(t) \subseteq \overline{\{1, \ldots, \mathrm{ar}(f)\}} \cup \{\epsilon\}) \wedge \\
& (\{f(t_1, \ldots, t_{\mathrm{ar}(f)})\} \neq \emptyset \ \Rightarrow \ R_f \subseteq R_f) \\
= \\
& \mathrm{true}
\end{aligned}
$$

- If $s'' = I \rhd s'''$, then we have the following.

$$
\begin{aligned}
& \mathrm{thrgh}(\varphi(I \rhd s'''), \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\}, \psi_\mathrm{r}(s'), \overline{\psi_\mathrm{i}(s')}) \\
= \\
& \mathrm{thrgh}(\mathsf{NF}(I, \varphi(s''')), \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\}, \psi_\mathrm{r}(s'), \overline{\psi_\mathrm{i}(s')}) \\
= \\
& \mathrm{thrgh}(\varphi(s'''), \{t[\chi]_I \ : \ t \in \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\} \wedge \\
& \qquad \forall_{i \in I}(i \in \mathrm{pos}(t) \ \Rightarrow \ \chi(i) \in \mathrm{rewr}(t|_i))\}, i \\
& \quad \{\rho \in \psi_\mathrm{r}(s') \ : \ \forall_{t \in \psi_\mathrm{r}}(\partial(t, \rho) \setminus \overline{\psi_\mathrm{i}(s')} \neq \emptyset) \ \vee \ \overline{I} \cap \mathrm{esspos}(\rho) \subseteq \overline{\psi_\mathrm{i}(s')}\}, \\
& \overline{\psi_\mathrm{i}(s')} \cup \overline{I})
\end{aligned}
$$

  For the last two arguments we can derive the following.

$$
\begin{aligned}
& \{\rho \in \psi_\mathrm{r}(s') \ : \ \forall_{t \in \psi_\mathrm{r}}(\partial(t, \rho) \setminus \overline{\psi_\mathrm{i}(s')} \neq \emptyset) \ \vee \ \overline{I} \cap \mathrm{esspos}(\rho) \subseteq \overline{\psi_\mathrm{i}(s')}\} \\
= \quad & \{ \ \forall_\pi(\pi \in \Pi \cap \overline{\Pi'} \ \Leftrightarrow \ \pi \in \overline{\Pi} \cap \overline{\Pi'}) \ \} \\
& \{\rho \in \psi_\mathrm{r}(s') \ : \ \forall_{t \in \psi_\mathrm{r}}(\partial(t, \rho) \setminus \overline{\psi_\mathrm{i}(s')} \neq \emptyset) \ \vee \ I \cap \mathrm{esspos}(\rho) \subseteq \overline{\psi_\mathrm{i}(s')}\} \\
= \quad & \{ \ \forall_{\rho \in \psi_\mathrm{r}(s')}(\exists_{s'_1, R, s'_2}(s' = s'_1 +\!\!+ (R \rhd s'_2) \wedge \\
& \qquad \rho \in R \ \wedge \ (\{1, \ldots, \mathrm{ar}(f)\} \cap \textstyle\bigcup_{\rho' \in R} \mathrm{esspos}(\rho')) \subseteq \psi_\mathrm{i}(s'_1))) \ \ \} \\
& \{\rho \in \psi_\mathrm{r}(s') \ : \ \forall_{t \in \psi_\mathrm{r}}(\partial(t, \rho) \setminus \overline{\psi_\mathrm{i}(s')} \neq \emptyset) \ \vee \ \mathrm{true}\} \\
= \\
&
\end{aligned}
$$

$$
\begin{aligned}
& \{\rho \in \psi_{\mathrm{r}}(s') \ : \ \mathrm{true}\} \\
= & \\
& \psi_{\mathrm{r}}(s') \\
= & \\
& \psi_{\mathrm{r}}(s' +\!\!+ [I])
\end{aligned}
$$

And:

$$
\begin{aligned}
& \overline{\psi_{\mathrm{i}}(s')} \cup \overline{I} \\
= & \\
& \overline{\psi_{\mathrm{i}}(s') \cup I} \\
= & \\
& \overline{\psi_{\mathrm{i}}(s' +\!\!+ [I])}
\end{aligned}
$$

Now, as $\{t[\chi]_I \ : \ t \in \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\} \ \wedge \ \forall_{i \in I}(i \in \mathrm{pos}(t) \ \Rightarrow \ \chi(i) \in \mathrm{rewr}(t|_i))\}$ is obviously included in $\{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\}$, we get (with Lemma E.8.4) that our goal follows from $\mathrm{thrgh}(\varphi(s'''), \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\}, \psi_{\mathrm{r}}(s' +\!\!+ [I]), \overline{\psi_{\mathrm{i}}(s' +\!\!+ [I])})$, which hold by induction.

- If $s'' = R \rhd s'''$, then we have the following.

$$
\begin{aligned}
& \mathrm{thrgh}(\varphi(R \rhd s'''), \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\}, \psi_{\mathrm{r}}(s'), \overline{\psi_{\mathrm{i}}(s')}) \\
= & \\
& \mathrm{thrgh}(\mathsf{T}(R, \varphi(s''')), \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\}, \psi_{\mathrm{r}}(s'), \overline{\psi_{\mathrm{i}}(s')}) \\
= & \\
& \mathrm{thrgh}(\varphi(s'''), \\
& \quad \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\} \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}, \\
& \quad \psi_{\mathrm{r}}(s') \cup R, \overline{\psi_{\mathrm{i}}(s')}) \\
= & \\
& \mathrm{thrgh}(\varphi(s'''), \\
& \quad \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\} \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}, \\
& \quad \psi_{\mathrm{r}}(s' +\!\!+ [R]), \overline{\psi_{\mathrm{i}}(s' +\!\!+ [R])}) \\
\Rightarrow & \quad \{ \text{ Lemma E.8.4 } \} \\
& \mathrm{thrgh}(\varphi(s'''), \{f(t_1, \ldots, t_{\mathrm{ar}(f)}) \ : \ \mathrm{true}\}, \psi_{\mathrm{r}}(s' +\!\!+ [R]), \overline{\psi_{\mathrm{i}}(s' +\!\!+ [R])}) \\
= & \\
& \mathrm{true}
\end{aligned}
$$

E.8 Theorem 5.4.1

We must show that, if f is a function symbol such that $\bigcup_{l\to r \textbf{ if } c\in R_f} \mathrm{pos}(l)$ is finite, we have that $\mathrm{stgen}(R_f)$ is thorough w.r.t. f and, similarly, that $\mathrm{stgen}_\mathrm{h}(R_f)$ is head thorough w.r.t. f.

Lemma E.8.1

$$\mathrm{stready}(R,\Pi) \subseteq R$$

Proof We trivially have that this is the case per definition of stready. □

Lemma E.8.2

$$\mathrm{need}^w_\mathrm{f}(R,\Pi) \subseteq \Pi$$

Proof For the defintion of need_f it trivially follows that $\mathrm{need}^w_\mathrm{f}(R,\Pi) \subseteq \{\pi \ : \langle \pi, n\rangle \in w(R,\Pi,\mathrm{need}_\mathrm{f})\}$. The latter set is a subset of Π per the requirement on w. □

Lemma E.8.3

$$\mathrm{need}^w_\mathrm{v}(R,\Pi) \subseteq \Pi$$

Proof For the defintion of need_v it trivially follows that $\mathrm{need}^w_\mathrm{v}(R,\Pi) \subseteq \{\pi \ : \langle \pi, n\rangle \in w(R,\Pi,\mathrm{need}_\mathrm{v})\}$. The latter set is a subset of Π per the requirement on w. □

Lemma E.8.4

$$\mathrm{thrgh}(T, S\cup S', R, \Pi) \Rightarrow \mathrm{thrgh}(T, S, R, \Pi)$$

Proof With induction on the structure of T. First the base cases:

- $T = \mathsf{E}$; this means we have

$$\begin{aligned}
&\mathrm{thrgh}(\mathsf{E}, S\cup S', R, \Pi)\\
=\ &\forall_{t\in S\cup S'}(\mathrm{pos}(t)\subseteq \Pi\cup\{\epsilon\}) \ \wedge\ (S\cup S'\neq\emptyset \ \Rightarrow\ R_f\subseteq R)\\
=\ &\forall_{t\in S}(\mathrm{pos}(t)\subseteq \Pi\cup\{\epsilon\}) \ \wedge\ \forall_{t\in S'}(\mathrm{pos}(t)\subseteq \Pi\cup\{\epsilon\}) \wedge\\
&(S\cup S'\neq\emptyset \ \Rightarrow\ R_f\subseteq R)\\
\Rightarrow\ &\forall_{t\in S}(\mathrm{pos}(t)\subseteq \Pi\cup\{\epsilon\}) \ \wedge\ (S\cup S'\neq\emptyset \ \Rightarrow\ R_f\subseteq R)\\
=\ &
\end{aligned}$$

$$
\begin{array}{ll}
& \forall_{t\in S}(\mathrm{pos}(t) \subseteq \Pi \cup \{\epsilon\}) \ \wedge \ (S \neq \emptyset \vee S' \neq \emptyset \ \Rightarrow \ R_f \subseteq R) \\
= & \\
& \forall_{t\in S}(\mathrm{pos}(t) \subseteq \Pi \cup \{\epsilon\}) \ \wedge \\
& (S \neq \emptyset \ \Rightarrow \ R_f \subseteq R) \ \wedge \ (S' \neq \emptyset \ \Rightarrow \ R_f \subseteq R) \\
\Rightarrow & \\
& \forall_{t\in S}(\mathrm{pos}(t) \subseteq \Pi \cup \{\epsilon\}) \ \wedge \ (S \neq \emptyset \ \Rightarrow \ R_f \subseteq R) \\
= & \\
& \mathrm{thrgh}(\mathsf{E}, S, R, \Pi)
\end{array}
$$

- $T = \mathsf{X}$; this means we have

$$
\begin{array}{ll}
& \mathrm{thrgh}(\mathsf{X}, S \cup S', R, \Pi) \\
= & \\
& \forall_{t\in S\cup S'}(t \rightarrow^{\omega}) \\
= & \\
& \forall_{t\in S}(t \rightarrow^{\omega}) \ \wedge \ \forall_{t\in S}(t \rightarrow^{\omega}) \\
\Rightarrow & \\
& \forall_{t\in S}(t \rightarrow^{\omega}) \\
= & \\
& \mathrm{thrgh}(\mathsf{X}, S, R, \Pi)
\end{array}
$$

Next the other cases:

- $T = \mathsf{F}(\pi, \psi)$ for some position π and function ψ of positions to strategy trees; this means we have

$$
\begin{array}{ll}
& \mathrm{thrgh}(\mathsf{F}(\pi, \psi), S \cup S', R, \Pi) \\
= & \\
& \forall_{f'}(\mathrm{thrgh}(\psi(f'), \{t \in S \cup S' \ : \ \pi \in \mathrm{pos_f}(t) \ \wedge \ \mathrm{hs}(t|_\pi) = f'\}, \\
& \quad R \cup \{l \rightarrow r \ \mathbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge \ \mathrm{hs}(l\sigma|_\pi) = f')\}, \Pi)) \ \wedge \\
& \mathrm{thrgh}(\psi(\bot), \{t \in S \cup S' \ : \ \pi \notin \mathrm{pos_f}(t)\}, \\
& \quad R \cup \{l \rightarrow r \ \mathbf{if} \ c \in R_f \ : \ \pi \in \mathrm{pos_f}(l)\}, \Pi \cup \{\pi\}) \\
= & \\
& \forall_{f'}(\mathrm{thrgh}(\psi(f'), \{t \in S \ : \ \pi \in \mathrm{pos_f}(t) \ \wedge \ \mathrm{hs}(t|_\pi) = f'\} \cup \\
& \qquad \{t \in S' \ : \ \pi \in \mathrm{pos_f}(t) \ \wedge \ \mathrm{hs}(t|_\pi) = f'\}, \\
& \quad R \cup \{l \rightarrow r \ \mathbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge \ \mathrm{hs}(l\sigma|_\pi) = f')\}, \Pi)) \ \wedge \\
& \mathrm{thrgh}(\psi(\bot), \{t \in S \ : \ \pi \notin \mathrm{pos_f}(t)\} \ \cup \ \{t \in S' \ : \ \pi \notin \mathrm{pos_f}(t)\}, \\
& \quad R \cup \{l \rightarrow r \ \mathbf{if} \ c \in R_f \ : \ \pi \in \mathrm{pos_f}(l)\}, \Pi \cup \{\pi\}) \\
\Rightarrow & \quad \{ \text{ Induction Hypothesis } \}
\end{array}
$$

$$
\begin{array}{l}
\quad \forall_{f'}(\mathrm{thrgh}(\psi(f'), \{t \in S \ : \ \pi \in \mathrm{pos_f}(t) \ \wedge \ \mathrm{hs}(t|_\pi) = f'\}, \\
\qquad R \cup \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge \ \mathrm{hs}(l\sigma|_\pi) = f')\}, \Pi)) \wedge \\
\quad \mathrm{thrgh}(\psi(\bot), \{t \in S \ : \ \pi \notin \mathrm{pos_f}(t)\}, \\
\qquad R \cup \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \pi \in \mathrm{pos_f}(l)\}, \Pi \cup \{\pi\}) \\
= \\
\quad \mathrm{thrgh}(\mathsf{F}(\pi, \psi), S, R, \Pi)
\end{array}
$$

- $T = \mathsf{H}(\Pi', U)$ for some set of positions Π' and strategy tree U; this means we have

$$
\begin{array}{ll}
& \mathrm{thrgh}(\mathsf{H}(\Pi', U), S \cup S', R, \Pi) \\
= & \\
& \mathrm{thrgh}(U, \{t[\psi]_{\Pi'} \ : \ t \in S \cup S' \wedge \\
& \qquad \forall_{\pi \in \Pi'}(\pi \in \mathrm{pos}(t) \ \Rightarrow \ \psi(\pi) \in \mathrm{rewr_h}(t|_\pi))\}, \\
& \quad \{\rho \in R \ : \ \Pi' \cap \mathrm{esspos}(\rho) \subseteq \Pi\}, (\Pi \cup \Pi') \setminus \{\pi \cdot i \cdot \pi' \ : \ \pi \in \Pi'\}) \\
= & \\
& \mathrm{thrgh}(U, \{t[\psi]_{\Pi'} \ : \ t \in S \wedge \\
& \qquad \forall_{\pi \in \Pi'}(\pi \in \mathrm{pos}(t) \ \Rightarrow \ \psi(\pi) \in \mathrm{rewr_h}(t|_\pi))\} \cup \\
& \qquad \{t[\psi]_{\Pi'} \ : \ t \in S' \ \wedge \ \forall_{\pi \in \Pi'}(\pi \in \mathrm{pos}(t) \ \Rightarrow \ \psi(\pi) \in \mathrm{rewr_h}(t|_\pi))\}, \\
& \quad \{\rho \in R \ : \ \Pi' \cap \mathrm{esspos}(\rho) \subseteq \Pi\}, (\Pi \cup \Pi') \setminus \{\pi \cdot i \cdot \pi' \ : \ \pi \in \Pi'\}) \\
\Rightarrow & \quad \{ \text{ Induction Hypothesis } \} \\
& \mathrm{thrgh}(U, \{t[\psi]_{\Pi'} \ : \ t \in S \ \wedge \ \forall_{\pi \in \Pi'}(\pi \in \mathrm{pos}(t) \ \Rightarrow \ \psi(\pi) \in \mathrm{rewr_h}(t|_\pi))\}, \\
& \quad \{\rho \in R \ : \ \Pi' \cap \mathrm{esspos}(\rho) \subseteq \Pi\}, (\Pi \cup \Pi') \setminus \{\pi \cdot i \cdot \pi' \ : \ \pi \in \Pi'\}) \\
= & \\
& \mathrm{thrgh}(\mathsf{H}(\Pi', U), S, R, \Pi)
\end{array}
$$

- $T = \mathsf{NF}(\Pi', U)$ for some set of positions Π' and strategy tree U; this case is similar to the previous case.

- $T = \mathsf{T}(R', U)$ for some set of rewrite rules R' and strategy tree U; this means we have

$$
\begin{array}{ll}
& \mathrm{thrgh}(\mathsf{T}(R', U), S \cup S', R, \Pi) \\
= & \\
& \mathrm{thrgh}(U, (S \cup S') \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}, \\
& \quad R \cup R', \Pi) \\
= &
\end{array}
$$

$$
\begin{array}{ll}
 & \mathrm{thrgh}(U, (S \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}) \cup \\
 & \qquad (S' \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}), R \cup R', \Pi) \\
\Rightarrow & \quad \{ \text{ Induction Hypothesis } \} \\
 & \mathrm{thrgh}(U, S \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}, R \cup R', \Pi) \\
= & \\
 & \mathrm{thrgh}(\mathsf{T}(R', U), S, R, \Pi)
\end{array}
$$

□

Definition stlim**.**

$$
\begin{array}{lcl}
\mathrm{stlim}(\mathsf{F}(\pi, \psi), \Pi) & = & \forall_{f_\perp}(\mathrm{stlim}(\psi(f_\perp), \Pi)) \\
\mathrm{stlim}(\mathsf{H}(\Pi', U), \Pi) & = & \Pi' \subseteq \Pi \ \wedge \ \mathrm{stlim}(U, \Pi) \\
\mathrm{stlim}(\mathsf{NF}(\Pi', U), \Pi) & = & \Pi' \subseteq \Pi \ \wedge \ \mathrm{stlim}(U, \Pi) \\
\mathrm{stlim}(\mathsf{T}(R, U), \Pi) & = & \mathrm{stlim}(U, \Pi) \\
\mathrm{stlim}(\mathsf{E}, \Pi) & = & \mathrm{true} \\
\mathrm{stlim}(\mathsf{X}, \Pi) & = & \mathrm{true}
\end{array}
$$

Lemma E.8.5

$$\mathrm{stlim}(T, \Pi) \Rightarrow \mathrm{stlim}(T, \Pi \cup \Pi')$$

Proof This follows trivially with induction on the structure of T. □

Lemma E.8.6 *Let* Π *be non-overlapping and* $R \subseteq R_f$ *for some* f*. We have the following:*

$$\mathrm{stlim}(\mathrm{stgen}'(R, \Pi), \overline{\Pi})$$

Proof We prove this with induction on the size of $\overline{\Pi} \cap \bigcup_{l \to r \ \mathbf{if} \ c \in R_f} \mathrm{pos}(l)$. With case distinction on $R = \emptyset$. If $R = \emptyset$, then we have

$$
\begin{array}{ll}
 & \mathrm{stlim}(\mathrm{stgen}'(\emptyset, \Pi), \overline{\Pi}) \\
= & \\
 & \mathrm{stlim}(\mathsf{NF}(\Pi, \mathsf{E}), \overline{\Pi}) \\
= & \\
 & \Pi \subseteq \overline{\Pi} \ \wedge \ \mathrm{stlim}(\mathsf{E}, \overline{\Pi}) \\
= & \\
 & \mathrm{true} \ \wedge \ \mathrm{true} \\
= & \\
 & \mathrm{true}
\end{array}
$$

Otherwise, if $R \neq \emptyset$, we have the following cases:

- $R' \neq \emptyset$ where $R' = \text{stready}(R, \Pi)$; which means that

$$
\begin{array}{ll}
& \text{stlim}(\text{stgen}'(R, \Pi), \overline{\Pi}) \\
= & \\
& \text{stlim}(\mathsf{T}(R', \text{stgen}'(R \setminus R', \Pi)), \overline{\Pi}) \\
= & \\
& \text{stlim}(\text{stgen}'(R \setminus R', \Pi), \overline{\Pi}) \\
= & \quad \{ \text{ Induction Hypothesis } \} \\
& \text{true}
\end{array}
$$

- $\text{stready}(R, \Pi) = \emptyset \ \wedge \ \text{need}^w_{\text{f}}(R, \Pi) \neq \emptyset \ \wedge \ \pi = \iota.\text{need}^w_{\text{f}}(R, \Pi)$; which means that

$$
\begin{array}{ll}
& \text{stlim}(\text{stgen}'(R, \Pi), \overline{\Pi}) \\
= & \\
& \text{stlim}(\mathsf{H}(\{\pi\}, \mathsf{F}(\pi, \text{stfunc}(\pi, R, \Pi))), \overline{\Pi}) \\
= & \\
& \{\pi\} \subseteq \overline{\Pi} \ \wedge \ \text{stlim}(\mathsf{F}(\pi, \text{stfunc}(\pi, R, \Pi)), \overline{\Pi}) \\
= & \\
& \{\pi\} \subseteq \overline{\Pi} \ \wedge \ \forall_{f_\perp}(\text{stlim}(\text{stfunc}(\pi, R, \Pi)(f_\perp), \overline{\Pi})) \\
= & \\
& \{\pi\} \subseteq \overline{\Pi} \ \wedge \ \forall_f(\text{stlim}(\text{stfunc}(\pi, R, \Pi)(f), \overline{\Pi})) \wedge \\
& \text{stlim}(\text{stfunc}(\pi, R, \Pi)(\perp), \overline{\Pi}) \\
= & \\
& \{\pi\} \subseteq \overline{\Pi} \wedge \\
& \forall_f(\text{stlim}(\text{stgen}'(\text{stfilter}_{\text{f}}(\pi, f, R), (\Pi \setminus \{\pi\}) \cup \{\pi \cdot i \ : \ 1 \leq i \leq \text{ar}(f)\}), \\
& \quad \overline{\Pi})) \wedge \\
& \text{stlim}(\text{stgen}'(\text{stfilter}_{\text{v}}(\pi, R), \Pi \setminus \{\pi\}), \overline{\Pi}) \\
\Leftarrow & \quad \{ \text{ Lemma E.8.5 } \} \\
& \{\pi\} \subseteq \overline{\Pi} \wedge \\
& \forall_f(\text{stlim}(\text{stgen}'(\text{stfilter}_{\text{f}}(\pi, f, R), (\Pi \setminus \{\pi\}) \cup \{\pi \cdot i \ : \ 1 \leq i \leq \text{ar}(f)\}), \\
& \quad \overline{\Pi} \setminus (\{\pi\} \cup \{\pi \cdot i \cdot \pi' \ : \ i > \text{ar}(f)\}))) \wedge \\
& \text{stlim}(\text{stgen}'(\text{stfilter}_{\text{v}}(\pi, R), \Pi \setminus \{\pi\}), \overline{\Pi} \setminus \{\pi \cdot \pi' \ : \ \text{true}\}) \\
= & \quad \{ \ \pi \in \Pi \text{ as } \text{need}^w_{\text{v}}(R, \Pi) \subseteq \Pi \ \}
\end{array}
$$

$$
\begin{array}{ll}
& \text{true} \wedge \\
& \forall_f(\text{stlim}(\text{stgen}'(\text{stfilter}_\text{f}(\pi, f, R), (\Pi \setminus \{\pi\}) \cup \{\pi \cdot i \ : \ 1 \leq i \leq \text{ar}(f)\}), \\
& \quad \overline{\Pi} \setminus (\{\pi\} \cup \{\pi \cdot i \cdot \pi' \ : \ i > \text{ar}(f)\}))) \wedge \\
& \text{stlim}(\text{stgen}'(\text{stfilter}_\text{v}(\pi, R), \Pi \setminus \{\pi\}), \overline{\Pi \setminus \{\pi\}}) \\
= & \quad \{\ \pi \in \Pi \text{ as } \text{need}^w_\text{v}(R, \Pi) \subseteq \Pi,\ \Pi \text{ is non-overlapping}\ \} \\
& \forall_f(\text{stlim}(\text{stgen}'(\text{stfilter}_\text{f}(\pi, f, R), (\Pi \setminus \{\pi\}) \cup \{\pi \cdot i \ : \ 1 \leq i \leq \text{ar}(f)\}), \\
& \quad \overline{(\Pi \setminus \{\pi\}) \cup \{\pi \cdot i \cdot \pi' \ : \ 1 \leq i \leq \text{ar}(f)\}})) \wedge \\
& \text{stlim}(\text{stgen}'(\text{stfilter}_\text{v}(\pi, R), \Pi \setminus \{\pi\}), \overline{\Pi \setminus \{\pi\}}) \\
= & \quad \{\ \text{Induction Hypothesis}\ \} \\
& \text{true} \ \wedge \ \forall_f(\text{true}) \ \wedge \ \text{true} \\
= & \\
& \text{true}
\end{array}
$$

- $\text{stready}(R, \Pi) = \emptyset \ \wedge \ \text{need}^w_\text{f}(R, \Pi) = \emptyset \ \wedge \ \pi = \iota.\text{need}^w_\text{v}(R, \Pi)$; which means that

$$
\begin{array}{ll}
& \text{stlim}(\text{stgen}'(R, \Pi), \overline{\Pi}) \\
= & \\
& \text{stlim}(\mathsf{NF}(\{\pi\}, \text{stgen}'(R, \Pi \setminus \{\pi\})), \overline{\Pi}) \\
= & \\
& \{\pi\} \subseteq \overline{\Pi} \ \wedge \ \text{stlim}(\text{stgen}'(R, \Pi \setminus \{\pi\}), \overline{\Pi}) \\
\Leftarrow & \quad \{\ \text{Lemma E.8.5}\ \} \\
& \{\pi\} \subseteq \overline{\Pi} \ \wedge \ \text{stlim}(\text{stgen}'(R, \Pi \setminus \{\pi\}), \overline{\Pi \setminus \{\pi\}}) \\
= & \quad \{\ \pi \in \Pi \text{ as } \text{need}^w_\text{v}(R, \Pi) \subseteq \Pi, \text{ Induction Hypothesis}\ \} \\
& \text{true} \ \wedge \ \text{true} \\
= & \\
& \text{true}
\end{array}
$$

□

Lemma E.8.7

$\text{thrgh}(T, S, R, \Pi \cup \Pi') \wedge \text{stlim}(T, (\Pi')^{-1}) \wedge \forall_{t \in S}(\text{pos}(t) \cap \Pi' = \emptyset) \Rightarrow \text{thrgh}(T, S, R, \Pi)$

Proof With induction on the structure of T. First the base cases:

- $T = \mathsf{E}$; this means we have

$$
\begin{array}{ll}
& \text{thrgh}(\mathsf{E}, S, R, \Pi) \\
= &
\end{array}
$$

$$
\begin{array}{ll}
& \forall_{t\in S}(\mathrm{pos}(t) \subseteq \Pi \cup \{\epsilon\}) \ \wedge \ (S \neq \emptyset \ \Rightarrow \ R_f \subseteq R) \\
= & \quad \{ \text{ Assumption } \forall_{t\in S}(\mathrm{pos}(t) \cap \Pi' = \emptyset) \ \} \\
& \forall_{t\in S}(\mathrm{pos}(t) \setminus \Pi' \subseteq \Pi \cup \{\epsilon\}) \ \wedge \ (S \neq \emptyset \ \Rightarrow \ R_f \subseteq R) \\
= & \\
& \forall_{t\in S}(\mathrm{pos}(t) \subseteq \Pi \cup \Pi' \cup \{\epsilon\}) \ \wedge \ (S \neq \emptyset \ \Rightarrow \ R_f \subseteq R) \\
= & \\
& \mathrm{thrgh}(\mathsf{E}, S, R, \Pi \cup \Pi') \\
= & \quad \{ \text{ Assumption } \} \\
& \mathrm{true}
\end{array}
$$

- $T = \mathsf{X}$; this means we have

$$
\begin{array}{ll}
& \mathrm{thrgh}(\mathsf{X}, S, R, \Pi) \\
= & \\
& \forall_{t\in S}(t \rightarrow^{\omega}) \\
= & \\
& \mathrm{thrgh}(\mathsf{X}, S, R, \Pi \cup \Pi') \\
= & \quad \{ \text{ Assumption } \} \\
& \mathrm{true}
\end{array}
$$

Next the other cases:

- $T = \mathsf{F}(\pi, \psi)$ for some position π and function ψ of positions to strategy trees. As $\forall_{t\in S'}(\mathrm{pos}(t) \cap \Pi' = \emptyset$ trivially follows from the assumption $\forall_{t\in S}(\mathrm{pos}(t) \cap \Pi' = \emptyset)$ if $S' = \{t \in S \ : \ \varphi(t)\}$ for some φ and $\mathrm{stlim}(\psi(f_\perp), (\Pi')^{-1})$ follows trivially from $\mathrm{stlim}(T, (\Pi')^{-1}$ for all $f_\perp$, this means we have the following.

$$
\begin{array}{ll}
& \mathrm{thrgh}(\mathsf{F}(\pi, \psi), S, R, \Pi) \\
= & \\
& \forall_{f'}(\mathrm{thrgh}(\psi(f'), \{t \in S \ : \ \pi \in \mathrm{pos_f}(t) \ \wedge \ \mathrm{hs}(t|_\pi) = f'\}, \\
& \quad R \cup \{l \rightarrow r \textbf{ if } c \in R_f \ : \ \neg\exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge \ \mathrm{hs}(l\sigma|_\pi) = f')\}, \Pi)) \wedge \\
& \mathrm{thrgh}(\psi(\perp), \{t \in S \ : \ \pi \notin \mathrm{pos_f}(t)\}, \\
& \quad R \cup \{l \rightarrow r \textbf{ if } c \in R_f \ : \ \pi \in \mathrm{pos_f}(l)\}, \Pi \cup \{\pi\}) \\
\Leftarrow & \quad \{ \text{ Induction Hypothesis } \}
\end{array}
$$

$$
\begin{array}{ll}
& \forall_{f'}(\mathrm{thrgh}(\psi(f'), \{t \in S \ : \ \pi \in \mathrm{pos_f}(t) \ \wedge \ \mathrm{hs}(t|_\pi) = f'\}, \\
& \quad R \cup \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge \ \mathrm{hs}(l\sigma|_\pi) = f')\}, \\
& \quad \Pi \cup \Pi')) \wedge \\
& \mathrm{thrgh}(\psi(\bot), \{t \in S \ : \ \pi \notin \mathrm{pos_f}(t)\}, \\
& \quad R \cup \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \pi \in \mathrm{pos_f}(l)\}, \Pi \cup \Pi' \cup \{\pi\}) \\
= & \\
& \mathrm{thrgh}(\mathsf{F}(\pi, \psi), S, R, \Pi \cup \Pi') \\
= & \quad \{ \text{ Assumption } \} \\
& \mathrm{true}
\end{array}
$$

- $T = \mathsf{H}(\Pi'', U)$ for some set of positions Π'' and strategy tree U. As we have $\mathrm{stlim}(\mathsf{H}(\Pi'', U), (\Pi')^{-1})$, we also have that $\Pi'' \subseteq (\Pi')^{-1}$ and $\mathrm{stlim}(U, (\Pi')^{-1})$. From the former it follows that $\Pi'' \cap \Pi' = \emptyset$ and for the later it follows that $\mathrm{stlim}(U, (\Pi' \setminus \Pi''')^{-1})$ for any set Π''' (by Lemma E.8.5). Also, in order to be able to apply the induction hypothesis later on, we need that we have $\forall_{t' \in \{t[\psi]_{\Pi''} \, : \, t \in S \wedge \forall_{\pi \in \Pi''}(\pi \in \mathrm{pos}(t) \Rightarrow \psi(\pi) \in \mathrm{rewr_h}(t|_\pi))\}}(\mathrm{pos}(t') \cap (\Pi' \setminus \{\pi \cdot i \cdot \pi' \ : \ \pi \in \Pi''\}) = \emptyset)$. It is easy to see that this follows from the obvious fact that $\mathrm{pos}(t') \cap (\{\pi \cdot i \cdot \pi' \ : \ \pi \in \Pi''\})^{-1} \subseteq \mathrm{pos}(t)$ and the assumption that $\forall_{t \in S}(\mathrm{pos}(t) \cap \Pi' = \emptyset)$. We then have the following.

$$
\begin{array}{ll}
& \mathrm{thrgh}(\mathsf{H}(\Pi'', U), S, R, \Pi) \\
= & \\
& \mathrm{thrgh}(U, \{t[\psi]_{\Pi''} \ : \ t \in S \ \wedge \ \forall_{\pi \in \Pi''}(\pi \in \mathrm{pos}(t) \ \Rightarrow \ \psi(\pi) \in \mathrm{rewr_h}(t|_\pi))\}, \\
& \quad \{\rho \in R \ : \ \Pi'' \cap \mathrm{esspos}(\rho) \subseteq \Pi\}, (\Pi \cup \Pi'') \setminus \{\pi \cdot i \cdot \pi' \ : \ \pi \in \Pi''\}) \\
\Leftarrow & \quad \{ \text{ Induction Hypothesis } \} \\
& \mathrm{thrgh}(U, \{t[\psi]_{\Pi''} \ : \ t \in S \ \wedge \ \forall_{\pi \in \Pi''}(\pi \in \mathrm{pos}(t) \ \Rightarrow \ \psi(\pi) \in \mathrm{rewr_h}(t|_\pi))\}, \\
& \quad \{\rho \in R \ : \ \Pi'' \cap \mathrm{esspos}(\rho) \subseteq \Pi\}, \\
& \quad ((\Pi \cup \Pi'') \setminus \{\pi \cdot i \cdot \pi' \ : \ \pi \in \Pi''\}) \ \cup \ (\Pi' \setminus \{\pi \cdot i \cdot \pi' \ : \ \pi \in \Pi''\})) \\
= & \\
& \mathrm{thrgh}(U, \{t[\psi]_{\Pi''} \ : \ t \in S \ \wedge \ \forall_{\pi \in \Pi''}(\pi \in \mathrm{pos}(t) \ \Rightarrow \ \psi(\pi) \in \mathrm{rewr_h}(t|_\pi))\}, \\
& \quad \{\rho \in R \ : \ \Pi'' \cap \mathrm{esspos}(\rho) \subseteq \Pi\}, \\
& \quad (\Pi \cup \Pi' \cup \Pi'') \setminus \{\pi \cdot i \cdot \pi' \ : \ \pi \in \Pi''\}) \\
= & \quad \{ \ \Pi'' \cap \Pi' = \emptyset \ \} \\
& \mathrm{thrgh}(U, \{t[\psi]_{\Pi''} \ : \ t \in S \ \wedge \ \forall_{\pi \in \Pi''}(\pi \in \mathrm{pos}(t) \ \Rightarrow \ \psi(\pi) \in \mathrm{rewr_h}(t|_\pi))\}, \\
& \quad \{\rho \in R \ : \ \Pi'' \cap \mathrm{esspos}(\rho) \subseteq \Pi \cup \Pi'\}, \\
& \quad (\Pi \cup \Pi' \cup \Pi'') \setminus \{\pi \cdot i \cdot \pi' \ : \ \pi \in \Pi''\}) \\
= &
\end{array}
$$

$$
\begin{array}{ll}
& \mathrm{thrgh}(\mathsf{H}(\Pi'', U), S, R, \Pi \cup \Pi') \\
= & \quad \{ \text{ Assumption } \} \\
& \mathrm{true}
\end{array}
$$

- $T = \mathsf{NF}(\Pi', U)$ for some set of positions Π' and strategy tree U; this case is similar to the previous case.
- $T = \mathsf{T}(R', U)$ for some set of rewrite rules R' and strategy tree U. As $\forall_{t \in S'}(\mathrm{pos}(t) \cap \Pi' = \emptyset$ trivially follows from the assumption $\forall_{t \in S}(\mathrm{pos}(t) \cap \Pi' = \emptyset)$ if $S' \subseteq S$ and $\mathrm{stlim}(U, (\Pi')^{-1})$ trivially follows from $\mathrm{stlim}(T, (\Pi')^{-1})$, this means we have the following.

$$
\begin{array}{ll}
& \mathrm{thrgh}(\mathsf{T}(R', U), S, R, \Pi) \\
= & \\
& \mathrm{thrgh}(U, S \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}, R \cup R', \Pi) \\
\Leftarrow & \quad \{ \text{ Induction Hypothesis } \} \\
& \mathrm{thrgh}(U, S \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}, R \cup R', \Pi \cup \Pi') \\
= & \\
& \mathrm{thrgh}(\mathsf{T}(R', U), S, R, \Pi \cup \Pi') \\
= & \quad \{ \text{ Assumption } \} \\
& \mathrm{true}
\end{array}
$$

□

We show that under the assumption that $R \subseteq R_f$, $\epsilon \notin \Pi$, Π is non-overlapping and $\forall_{t \in \mathfrak{S}(R), l \to r \ \mathbf{if} \ c \in \mathfrak{R}(R)}(\partial(t, l) \cap \mathfrak{P}(\Pi) \neq \emptyset \ \vee \ (\partial(t, l) = \emptyset \ \wedge \ \mathrm{esspos}(l \to r \ \mathbf{if} \ c) \subseteq \Pi \ \wedge \ \neg\exists_\sigma(t = l\sigma \wedge \mathrm{true} \in \mathrm{rewr}(c\sigma))))$, it holds that $\mathrm{thrgh}(\mathrm{stgen}'(R, \Pi), \mathfrak{S}(R), \mathfrak{R}(R), \mathfrak{P}(\Pi))$, where

$$
\begin{array}{lcl}
\mathfrak{S}(R) & = & \{t \ : \ \neg\exists_{l \to r \ \mathbf{if} \ c \in R_f \setminus R}(t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma))\} \\
\mathfrak{R}(R) & = & R_f \setminus R \\
\mathfrak{P}(\Pi) & = & (\overline{\Pi})^{-1} \setminus \{\epsilon\}
\end{array} .
$$

We do this with induction on the size of $\overline{\Pi} \cap \bigcup_{l \to r \ \mathbf{if} \ c \in R_f} \mathrm{pos}(l)$. With this we trivially have that $\mathrm{thrgh}(\mathrm{stgen}(R_f), \{f(t_1, \ldots, t_n) \ : \ \mathrm{true}\}, \emptyset, \emptyset)$ holds. Note that we do not explicitly show that the assumptions (except the last) hold when applying the induction hypothesis.

First the case that $R = \emptyset$. Here we have the following.

$$
\begin{array}{ll}
& \mathrm{thrgh}(\mathrm{stgen}'(\emptyset, \Pi), \mathfrak{S}(\emptyset), \mathfrak{R}(\emptyset), \mathfrak{P}(\Pi)) \\
= &
\end{array}
$$

$$
\begin{aligned}
& \mathrm{thrgh}(\mathsf{NF}(\Pi,\mathsf{E}),\mathfrak{S}(\emptyset),\mathfrak{R}(\emptyset),\mathfrak{P}(\Pi)) \\
= & \\
& \mathrm{thrgh}(\mathsf{E},\{t'[\varphi]_\Pi \;:\; t'\in\mathfrak{S}(\emptyset) \;\wedge\; \forall_{\pi\in\Pi}(\pi\in\mathrm{pos}(t') \;\Rightarrow\; \psi(\pi)\in\mathrm{rewr}(t'|_\pi))\}, \\
& \quad \{\rho\in\mathfrak{R}(\emptyset) \;:\; \forall_{t\in\mathfrak{S}(\emptyset)}(\partial(t,\rho)\setminus\Pi\neq\emptyset) \;\vee\; \overline{\Pi}\cap\mathrm{esspos}(\rho)\subseteq\mathfrak{P}(\Pi)\}, \\
& \quad \mathfrak{P}(\Pi)\cup\overline{\Pi}) \\
= & \\
& \forall_{t\in\{t'[\varphi]_\Pi\,:\,t'\in\mathfrak{S}(\emptyset)\,\wedge\,\forall_{\pi\in\Pi}(\pi\in\mathrm{pos}(t')\,\Rightarrow\,\psi(\pi)\in\mathrm{rewr}(t'|_\pi))\}}(\\
& \qquad \mathrm{pos}(t)\subseteq\mathfrak{P}(\Pi)\cup\overline{\Pi}\cup\{\epsilon\}) \;\wedge \\
& \quad R_f\subseteq\{\rho\in\mathfrak{R}(\emptyset) \;:\; \forall_{t\in\mathfrak{S}(\emptyset)}(\partial(t,\rho)\setminus\Pi\neq\emptyset) \;\vee\; \overline{\Pi}\cap\mathrm{esspos}(\rho)\subseteq\mathfrak{P}(\Pi)\} \\
= & \\
& \forall_{t\in\{t'[\varphi]_\Pi\,:\,t'\in\mathfrak{S}(\emptyset)\,\wedge\,\forall_{\pi\in\Pi}(\pi\in\mathrm{pos}(t')\,\Rightarrow\,\psi(\pi)\in\mathrm{rewr}(t'|_\pi))\}}(\\
& \qquad \mathrm{pos}(t)\subseteq((\overline{\Pi})^{-1}\setminus\{\epsilon\})\cup\overline{\Pi}\cup\{\epsilon\}) \;\wedge \\
& \quad R_f\subseteq\{\rho\in\mathfrak{R}(\emptyset) \;:\; \forall_{t\in\mathfrak{S}(\emptyset)}(\partial(t,\rho)\setminus\Pi\neq\emptyset) \;\vee\; \overline{\Pi}\cap\mathrm{esspos}(\rho)\subseteq\mathfrak{P}(\Pi)\} \\
= & \\
& \forall_{t\in\{t'[\varphi]_\Pi\,:\,t'\in\mathfrak{S}(\emptyset)\,\wedge\,\forall_{\pi\in\Pi}(\pi\in\mathrm{pos}(t')\,\Rightarrow\,\psi(\pi)\in\mathrm{rewr}(t'|_\pi))\}}(\\
& \qquad \mathrm{pos}(t)\subseteq(\overline{\Pi})^{-1}\cup\overline{\Pi}\cup\{\epsilon\}) \;\wedge \\
& \quad R_f\subseteq\{\rho\in\mathfrak{R}(\emptyset) \;:\; \forall_{t\in\mathfrak{S}(\emptyset)}(\partial(t,\rho)\setminus\Pi\neq\emptyset) \;\vee\; \overline{\Pi}\cap\mathrm{esspos}(\rho)\subseteq\mathfrak{P}(\Pi)\} \\
= & \\
& \forall_{t\in\{t'[\varphi]_\Pi\,:\,t'\in\mathfrak{S}(\emptyset)\,\wedge\,\forall_{\pi\in\Pi}(\pi\in\mathrm{pos}(t')\,\Rightarrow\,\psi(\pi)\in\mathrm{rewr}(t'|_\pi))\}}(\\
& \qquad \mathrm{pos}(t)\subseteq\{\pi \;:\; \mathrm{true}\}) \;\wedge \\
& \quad R_f\subseteq\{\rho\in\mathfrak{R}(\emptyset) \;:\; \forall_{t\in\mathfrak{S}(\emptyset)}(\partial(t,\rho)\setminus\Pi\neq\emptyset) \;\vee\; \overline{\Pi}\cap\mathrm{esspos}(\rho)\subseteq\mathfrak{P}(\Pi)\} \\
= & \\
& \mathrm{true} \;\wedge\; R_f\subseteq\{\rho\in R_f\setminus\emptyset \;:\; \forall_{t\in\mathfrak{S}(\emptyset)}(\partial(t,\rho)\setminus\mathfrak{P}(\Pi)\neq\emptyset)\vee \\
& \qquad \Pi\cap\mathrm{esspos}(\rho)\subseteq\mathfrak{P}(\Pi)\} \\
= & \\
& \forall_{\rho\in R_f}(\forall_{t\in\mathfrak{S}(\emptyset)}(\partial(t,\rho)\setminus\Pi\neq\emptyset) \;\vee\; \overline{\Pi}\cap\mathrm{esspos}(\rho)\subseteq\mathfrak{P}(\Pi)) \\
= & \\
& \forall_{\rho\in R_f}(\forall_{t\in\mathfrak{S}(\emptyset)}(\partial(t,\rho)\setminus\Pi\neq\emptyset) \;\vee\; \overline{\Pi}\cap\mathrm{esspos}(\rho)=\emptyset) \\
= & \quad \{\ \partial(t,l)\cap\mathfrak{P}(\Pi)\neq\emptyset \text{ implies } \partial(t,l)\setminus(\Pi\cup\{\epsilon\})\neq\emptyset, \text{ Assumption }\} \\
& \mathrm{true}
\end{aligned}
$$

Next the case $R\neq\emptyset$. With case distinction on the values of $\mathrm{stready}(R,\Pi)$ and $\mathrm{need}^w_{\mathrm{f}}(R,\Pi)$ we get the following cases for $\mathrm{thrgh}(\mathrm{stgen}'(R,\Pi),\mathfrak{S}(R),\mathfrak{R}(R),\mathfrak{P}(\Pi))$.

- $R'\neq\emptyset$ where $R'=\mathrm{stready}(R,\Pi)$; then we have

$$
\begin{aligned}
& \mathrm{thrgh}(\mathrm{stgen}'(R,\Pi),\mathfrak{S}(R),\mathfrak{R}(R),\mathfrak{P}(\Pi)) \\
= &
\end{aligned}
$$

$$
\begin{array}{ll}
& \mathrm{thrgh}(\mathsf{T}(R', \mathrm{stgen}'(R \setminus R', \Pi)), \mathfrak{S}(R), \mathfrak{R}(R), \mathfrak{P}(\Pi)) \\
= & \\
& \mathrm{thrgh}(\mathrm{stgen}'(R \setminus R', \Pi), \\
& \quad \mathfrak{S}(R) \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}, \\
& \quad \mathfrak{R}(R) \cup R', \mathfrak{P}(\Pi))
\end{array}
$$

If we can show that $\forall_{t \in \mathfrak{S}(R) \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\}, \rho \in R_f \setminus (R \setminus R')} (\partial(t, l) \cap \mathfrak{P}(\Pi) \neq \emptyset \ \vee \ (\partial(t, l) = \emptyset \ \wedge \ \mathrm{esspos}(l \to r \ \mathbf{if} \ c) \subseteq \Pi \ \wedge \ \neg \exists_\sigma (t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma))))$, $\mathfrak{S}(R) \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\} = \mathfrak{S}(R \setminus R')$ and $\mathfrak{R}(R) \cup R' = \mathfrak{R}(R \setminus R')$, then by induction we have that this is true. The former, using the assumption and $\mathrm{stready}(R, \Pi) \subseteq R$ (Lemma E.8.1), follows from $\forall_{t \in \{t' \ : \ \neg \exists_{l \to r \ \mathbf{if} \ c \in R'} (t' = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma))\}, \rho \in R'} (\partial(t, l) \cap \mathfrak{P}(\Pi) \neq \emptyset \ \vee \ (\partial(t, l) = \emptyset \ \wedge \ \mathrm{esspos}(l \to r \ \mathbf{if} \ c) \subseteq \Pi \ \wedge \ \neg \exists_\sigma (t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma))))$, which trivially holds per the definition of stready. For the second statement we have the following.

$$
\begin{array}{ll}
& \mathfrak{S}(R) \setminus \{l\sigma \ : \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\} \\
= & \\
& \{t \ : \ \neg \exists_{l \to r \ \mathbf{if} \ c \in R_f \setminus R} (t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma))\} \setminus \\
& \quad \{t \ : \ t = l\sigma \ \wedge \ l \to r \ \mathbf{if} \ c \in R' \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)\} \\
= & \\
& \{t \ : \ \neg \exists_{l \to r \ \mathbf{if} \ c \in R_f \setminus R} (t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)) \wedge \\
& \quad \neg \exists_{l \to r \ \mathbf{if} \ c \in R'} (t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma))\} \\
= & \\
& \{t \ : \ \neg \exists_{l \to r \ \mathbf{if} \ c \in (R_f \setminus R) \cup R'} (t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma))\} \\
= & \quad \{ \ R' \subseteq R \text{ by Lemma E.8.1 and assumption } R \subset R_f \ \} \\
& \{t \ : \ \neg \exists_{l \to r \ \mathbf{if} \ c \in R_f \setminus (R \setminus R')} (t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma))\} \\
= & \\
& \mathfrak{S}(R \setminus R')
\end{array}
$$

The following derivation concludes this case.

$$
\begin{array}{ll}
& \mathfrak{R}(R) \cup R' \\
= & \\
& (R_f \setminus R) \cup R' \\
= & \\
& (R_f \cup R') \setminus (R \setminus R') \\
= & \quad \{ \ R' \subseteq R \text{ by Lemma E.8.1 and assumption } R \subset R_f \ \}
\end{array}
$$

$$
\begin{array}{ll}
& R_f \setminus (R \setminus R') \\
= & \\
& \mathfrak{R}(R \setminus R')
\end{array}
$$

- $\text{stready}(R, \Pi) = \emptyset \ \wedge \ \text{need}^w_{\text{f}}(R, \Pi) \neq \emptyset \ \wedge \ \pi = \iota.\text{need}^w_{\text{f}}(R, \Pi)$; this means

$$
\begin{array}{ll}
& \text{thrgh}(\text{stgen}'(R, \Pi), \mathfrak{S}(R), \mathfrak{R}(R), \mathfrak{P}(\Pi)) \\
= & \\
& \text{thrgh}(\mathsf{H}(\{\pi\}, \mathsf{F}(\pi, \text{stfunc}(\pi, R, \Pi))), \mathfrak{S}(R), \mathfrak{R}(R), \mathfrak{P}(\Pi)) \\
= & \\
& \text{thrgh}(\mathsf{F}(\pi, \text{stfunc}(\pi, R, \Pi)), \\
& \quad \{t[\psi]_{\{\pi\}} \ : \ t \in \mathfrak{S}(R) \ \wedge \ \forall_{\pi' \in \{\pi\}}(\pi' \in \text{pos}(t) \ \Rightarrow \ \psi(\pi') \in \text{rewr}(t|_{\pi'}))\}, \\
& \quad \{\rho \in \mathfrak{R}(R) \ : \ \{\pi\} \cap \text{esspos}(\rho) \subseteq \mathfrak{P}(\Pi)\}, \\
& \quad (\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi' \cdot i \cdot \pi'' \ : \ \pi' \in \{\pi\}\}) \\
= & \\
& \text{thrgh}(\mathsf{F}(\pi, \text{stfunc}(\pi, R, \Pi)), \\
& \quad \{t[\psi(\pi)]_{\pi} \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \text{pos}(t) \ \Rightarrow \ \psi(\pi) \in \text{rewr}(t|_{\pi}))\}, \\
& \quad \{\rho \in \mathfrak{R}(R) \ : \ \pi \notin \text{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\}, \\
& \quad (\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\}) \\
= & \\
& \text{thrgh}(\mathsf{F}(\pi, \text{stfunc}(\pi, R, \Pi)), \\
& \quad \{t[u]_{\pi} \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \text{pos}(t) \ \Rightarrow \ u \in \text{rewr}(t|_{\pi}))\}, \\
& \quad \{\rho \in \mathfrak{R}(R) \ : \ \pi \notin \text{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\}, \\
& \quad (\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\}) \\
= &
\end{array}
$$

$$
\begin{array}{l}
\forall_{f'}(\mathrm{thrgh}(\mathrm{stfunc}(\pi, R, \Pi)(f'),\\
\quad \{v \in \{t[u]_\pi \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi))\}\\
\qquad : \ \pi \in \mathrm{pos}_{\mathrm{f}}(v) \ \wedge \ \mathrm{hs}(v|_\pi) = f'\},\\
\quad \{\rho \in \mathfrak{R}(R) \ : \ \pi \notin \mathrm{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \cup\\
\qquad \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \mathrm{pos}_{\mathrm{f}}(l\sigma) \ \wedge \ \mathrm{hs}(l\sigma|_\pi) = f')\},\\
\quad (\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \mathrm{true}\})) \wedge\\
\mathrm{thrgh}(\mathrm{stfunc}(\pi, R, \Pi)(\bot),\\
\quad \{v \in \{t[u]_\pi \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi))\}\\
\qquad : \ \pi \notin \mathrm{pos}_{\mathrm{f}}(v)\},\\
\quad \{\rho \in \mathfrak{R}(R) \ : \ \pi \notin \mathrm{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \cup\\
\qquad \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \pi \in \mathrm{pos}_{\mathrm{f}}(l)\},\\
\quad ((\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \mathrm{true}\}) \cup \{\pi\})
\end{array}
$$

We separately continue with each of the above conjuncts. With the first we get the following.

$$
\begin{array}{l}
\forall_{f'}(\mathrm{thrgh}(\mathrm{stfunc}(\pi, R, \Pi)(f'),\\
\quad \{v \in \{t[u]_\pi \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi))\}\\
\qquad : \ \pi \in \mathrm{pos}_{\mathrm{f}}(v) \ \wedge \ \mathrm{hs}(v|_\pi) = f'\},\\
\quad \{\rho \in \mathfrak{R}(R) \ : \ \pi \notin \mathrm{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \cup\\
\qquad \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \mathrm{pos}_{\mathrm{f}}(l\sigma) \ \wedge \ \mathrm{hs}(l\sigma|_\pi) = f')\},\\
\quad (\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \mathrm{true}\}))\\
=\\
\forall_{f'}(\mathrm{thrgh}(\mathrm{stfunc}(\pi, R, \Pi)(f'),\\
\quad \{t[u]_\pi \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi)) \wedge\\
\qquad \pi \in \mathrm{pos}_{\mathrm{f}}(t[u]_\pi) \ \wedge \ \mathrm{hs}(u) = f'\},\\
\quad \{\rho \in \mathfrak{R}(R) \ : \ \pi \notin \mathrm{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \cup\\
\qquad \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \mathrm{pos}_{\mathrm{f}}(l\sigma) \ \wedge \ \mathrm{hs}(l\sigma|_\pi) = f')\},\\
\quad (\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \mathrm{true}\}))\\
=\\
\forall_{f'}(\mathrm{thrgh}(\mathrm{stgen}'(\mathrm{stfilter}_{\mathrm{f}}(\pi, f', R), (\Pi \setminus \{\pi\}) \cup \{\pi \cdot i \ : \ 1 \leq i \leq \mathrm{ar}(f')\}),\\
\quad \{t[u]_\pi \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi)) \wedge\\
\qquad \pi \in \mathrm{pos}_{\mathrm{f}}(t[u]_\pi) \ \wedge \ \mathrm{hs}(u) = f'\},\\
\quad \{\rho \in \mathfrak{R}(R) \ : \ \pi \notin \mathrm{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \cup\\
\qquad \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \mathrm{pos}_{\mathrm{f}}(l\sigma) \ \wedge \ \mathrm{hs}(l\sigma|_\pi) = f')\},\\
\quad (\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \mathrm{true}\}))
\end{array}
$$

Next we derive relations between the arguments of the thrgh above and $\mathfrak{S}(\text{stfilter}_\text{f}(\pi, f', R))$, $\mathfrak{R}(\text{stfilter}_\text{f}(\pi, f', R))$ and $\mathfrak{P}((\Pi \setminus \{\pi\}) \cup \{\pi \cdot i \ : \ 1 \leq i \leq \text{ar}(f')\})$.

$$
\begin{aligned}
&\{t[u]_\pi \ : \ t \in \mathfrak{S}(R) \wedge \\
&\quad (\pi \in \text{pos}(t) \ \Rightarrow \ u \in \text{rewr}(t|_\pi)) \ \wedge \ \pi \in \text{pos}_\text{f}(t[u]_\pi) \ \wedge \ \text{hs}(u) = f'\} \\
=\ &\{t[u]_\pi \ : \ t \in \{t' \ : \ \neg\exists_{l \to r \ \textbf{if} \ c \in R_f \setminus R}(t' = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma))\} \wedge \\
&\quad (\pi \in \text{pos}(t) \ \Rightarrow \ u \in \text{rewr}(t|_\pi)) \ \wedge \ \pi \in \text{pos}_\text{f}(t[u]_\pi) \ \wedge \ \text{hs}(u) = f'\} \\
=\ &\{t[u]_\pi \ : \ \neg\exists_{l \to r \ \textbf{if} \ c \in R_f \setminus R}(t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma)) \wedge \\
&\quad (\pi \in \text{pos}(t) \ \Rightarrow \ u \in \text{rewr}(t|_\pi)) \ \wedge \ \pi \in \text{pos}_\text{f}(t[u]_\pi) \ \wedge \ \text{hs}(u) = f'\} \\
\subseteq\ &\{t[u]_\pi \ : \ \neg\exists_{l \to r \ \textbf{if} \ c \in R_f \setminus R}(t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma)) \wedge \\
&\quad \pi \in \text{pos}_\text{f}(t[u]_\pi) \ \wedge \ \text{hs}(u) = f'\} \\
=\ &\quad \{\ \pi \in \Pi \text{ (Lemma E.8.2)}, \\
&\qquad \text{assumption } \forall_{\rho \in R_f \setminus R}(\text{esspos}(\rho) \cap \Pi = \emptyset)\ \} \\
&\{t \ : \ \neg\exists_{l \to r \ \textbf{if} \ c \in R_f \setminus R}(t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma)) \wedge \\
&\quad \pi \in \text{pos}_\text{f}(t) \wedge \text{hs}(t|_\pi) = f'\} \\
\subseteq\ &\{t \ : \ \neg\exists_{l \to r \ \textbf{if} \ c \in R_f \setminus R}(t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma)) \wedge \\
&\quad ((\pi \in \text{pos}_\text{f}(t) \wedge \text{hs}(t|_\pi) = f') \vee \\
&\qquad \neg\exists_{l \to r \ \textbf{if} \ c \in \{l \to r \ \textbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \wedge \text{hs}(l\sigma|_\pi) = f')\}}(\\
&\qquad\quad t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma)))\} \\
=\ &\{t \ : \ \neg\exists_{l \to r \ \textbf{if} \ c \in R_f \setminus R}(t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma)) \wedge \\
&\quad \neg\exists_{l \to r \ \textbf{if} \ c \in \{l \to r \ \textbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \wedge \text{hs}(l\sigma|_\pi) = f')\}}(\\
&\qquad \neg(\pi \in \text{pos}_\text{f}(t) \wedge \text{hs}(t|_\pi) = f') \ \wedge \ t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma))\} \\
=\ &\{t \ : \ \neg\exists_{l \to r \ \textbf{if} \ c \in R_f \setminus R}(t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma)) \wedge \\
&\quad \neg\exists_{l \to r \ \textbf{if} \ c \in \{l \to r \ \textbf{if} \ c \in R_f \ : \ \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \wedge \text{hs}(l\sigma|_\pi) = f')\}}(\\
&\qquad t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma))\} \\
=\ &
\end{aligned}
$$

$$
\begin{aligned}
&\{t \;:\; \neg\exists_{l\to r \textbf{ if } c\in R_f\setminus R}(t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma)) \wedge \\
&\quad \neg\exists_{l\to r \textbf{ if } c\in R_f\cap\{l\to r \textbf{ if } c\,:\,\neg\exists_\sigma(\pi\in\text{pos}_\text{f}(l\sigma)\,\wedge\,\text{hs}(l\sigma|_\pi)=f')\}}(\\
&\qquad t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma))\} \\
=\;& \\
&\{t \;:\; \neg\exists_{l\to r \textbf{ if } c\in(R_f\setminus R)\cup(R_f\cap\{l\to r \textbf{ if } c\,:\,\neg\exists_\sigma(\pi\in\text{pos}_\text{f}(l\sigma)\,\wedge\,\text{hs}(l\sigma|_\pi)=f')\})}(\\
&\qquad t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma))\} \\
=\;& \\
&\{t \;:\; \neg\exists_{l\to r \textbf{ if } c\in R_f\setminus(R\setminus\{l\to r \textbf{ if } c\,:\,\neg\exists_\sigma(\pi\in\text{pos}_\text{f}(l\sigma)\,\wedge\,\text{hs}(l\sigma|_\pi)=f')\})}(\\
&\qquad t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma))\} \\
=\;& \\
&\{t \;:\; \neg\exists_{l\to r \textbf{ if } c\in R_f\setminus\{l\to r \textbf{ if } c\in R\,:\,\exists_\sigma(\pi\in\text{pos}_\text{f}(l\sigma)\,\wedge\,\text{hs}(l\sigma|_\pi)=f')\}}(\\
&\qquad t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma))\} \\
=\;& \\
&\mathfrak{S}(\{l \to r \textbf{ if } c \in R \;:\; \exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \;\wedge\; \text{hs}(l\sigma|_\pi) = f')\}) \\
=\;& \\
&\mathfrak{S}(\text{stfilter}_\text{f}(\pi, f, R))
\end{aligned}
$$

Next for $\mathfrak{R}$.

$$
\begin{aligned}
&\{\rho \in \mathfrak{R}(R) \;:\; \pi \notin \text{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\}\cup \\
&\quad \{l \to r \textbf{ if } c \in R_f \;:\; \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \;\wedge\; \text{hs}(l\sigma|_\pi) = f')\} \\
=\;& \\
&\{\rho \in R_f \setminus R \;:\; \pi \notin \text{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\}\cup \\
&\quad \{l \to r \textbf{ if } c \in R_f \;:\; \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \;\wedge\; \text{hs}(l\sigma|_\pi) = f')\} \\
=\;& \\
&(\{\rho \in R_f \;:\; \pi \notin \text{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \setminus \{\rho \in R \;:\; \text{true}\})\cup \\
&\quad \{l \to r \textbf{ if } c \in R_f \;:\; \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \;\wedge\; \text{hs}(l\sigma|_\pi) = f')\} \\
=\;& \\
&(\{\rho \in R_f \;:\; \pi \notin \text{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\}\cup \\
&\quad \{l \to r \textbf{ if } c \in R_f \;:\; \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \;\wedge\; \text{hs}(l\sigma|_\pi) = f')\})\setminus \\
&(\{\rho \in R \;:\; \text{true}\}\setminus \\
&\quad \{l \to r \textbf{ if } c \in R_f \;:\; \neg\exists_\sigma(\pi \in \text{pos}_\text{f}(l\sigma) \;\wedge\; \text{hs}(l\sigma|_\pi) = f')\}) \\
=\;& \quad \{\text{ Assumption } R \subseteq R_f \;\}
\end{aligned}
$$

$$
\begin{aligned}
&\{l \to r \textbf{ if } c \in R_f \ :\ \pi \notin \mathrm{esspos}(l \to r \textbf{ if } c) \setminus \mathfrak{P}(\Pi) \vee \\
&\quad \neg\exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge\ \mathrm{hs}(l\sigma|_\pi) = f')\} \setminus \\
&\{l \to r \textbf{ if } c \in R \ :\ \exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge\ \mathrm{hs}(l\sigma|_\pi) = f')\} \\
=& \\
&\{l \to r \textbf{ if } c \in R_f \ :\ \pi \notin \mathrm{esspos}(l \to r \textbf{ if } c) \setminus \mathfrak{P}(\Pi) \vee \\
&\quad \neg\exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge\ \mathrm{hs}(l\sigma|_\pi) = f') \vee \\
&\quad (l \to r \textbf{ if } c \in R \ \wedge\ \exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge\ \mathrm{hs}(l\sigma|_\pi) = f'))\} \setminus \\
&\{l \to r \textbf{ if } c \in R \ :\ \exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge\ \mathrm{hs}(l\sigma|_\pi) = f')\} \\
=& \\
&\{l \to r \textbf{ if } c \in R_f \ :\ \pi \notin \mathrm{esspos}(l \to r \textbf{ if } c) \setminus \mathfrak{P}(\Pi) \vee \\
&\quad \neg\exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge\ \mathrm{hs}(l\sigma|_\pi) = f') \vee \\
&\quad l \to r \textbf{ if } c \in R\} \setminus \\
&\{l \to r \textbf{ if } c \in R \ :\ \exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge\ \mathrm{hs}(l\sigma|_\pi) = f')\} \\
=& \\
&\{l \to r \textbf{ if } c \in R_f \ :\ l \to r \textbf{ if } c \notin R \ \Rightarrow\ \pi \notin \mathrm{esspos}(l \to r \textbf{ if } c) \setminus \mathfrak{P}(\Pi) \vee \\
&\quad \neg\exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge\ \mathrm{hs}(l\sigma|_\pi) = f')\} \setminus \\
&\{l \to r \textbf{ if } c \in R \ :\ \exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge\ \mathrm{hs}(l\sigma|_\pi) = f')\} \\
=& \quad \{\ \pi \in \Pi \text{ (Lemma E.8.2)}, \\
&\qquad \text{assumption } \forall_{\rho \in R_f \setminus R}(\mathrm{esspos}(\rho) \cap \Pi = \emptyset)\ \} \\
&\{l \to r \textbf{ if } c \in R_f \ :\ \mathrm{true}\} \setminus \\
&\quad \{l \to r \textbf{ if } c \in R \ :\ \exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge\ \mathrm{hs}(l\sigma|_\pi) = f')\} \\
=& \\
&R_f \setminus \{l \to r \textbf{ if } c \in R \ :\ \exists_\sigma(\pi \in \mathrm{pos_f}(l\sigma) \ \wedge\ \mathrm{hs}(l\sigma|_\pi) = f')\} \\
=& \\
&R_f \setminus \mathrm{stfilter_f}(\pi, f', R) \\
=& \\
&\mathfrak{R}(\mathrm{stfilter_f}(\pi, f', R))
\end{aligned}
$$

Finally for $\mathfrak{P}$.

$$
\begin{aligned}
&(\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ :\ \mathrm{true}\} \\
=& \\
&(((\overline{\Pi})^{-1} \setminus \{\epsilon\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ :\ \mathrm{true}\} \\
=& \quad \{\ \pi \in \Pi \text{ (Lemma E.8.2)}\ \} \\
&(((\overline{(\Pi \setminus \{\pi\}) \cup \{\pi\}})^{-1} \setminus \{\epsilon\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ :\ \mathrm{true}\} \\
=&
\end{aligned}
$$

$$
\begin{aligned}
& (((\overline{(\Pi \setminus \{\pi\})} \cup \{\pi \cdot \pi' \ : \ \text{true}\})^{-1} \setminus \{\epsilon\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
= & \\
& ((((\overline{(\Pi \setminus \{\pi\})})^{-1} \setminus \{\pi \cdot \pi' \ : \ \text{true}\}) \setminus \{\epsilon\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
= & \\
& ((((\overline{(\Pi \setminus \{\pi\})})^{-1} \setminus \{\epsilon\}) \setminus \{\pi \cdot \pi' \ : \ \text{true}\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
= & \\
& ((((\overline{(\Pi \setminus \{\pi\})})^{-1} \setminus \{\epsilon\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
= & \\
& (((\overline{(\Pi \setminus \{\pi\})})^{-1} \setminus \{\epsilon\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
= & \\
& (((\overline{(\Pi \setminus \{\pi\})})^{-1} \cup \{\pi\}) \setminus \{\epsilon\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
= & \quad \{ \ \pi \in \Pi \ \text{(Lemma E.8.2)}, \\
& \qquad \text{assumption that } \Pi \text{ is non-overlapping } \} \\
& ((\overline{(\Pi \setminus \{\pi\})})^{-1} \setminus \{\epsilon\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
\subseteq & \\
& ((\overline{(\Pi \setminus \{\pi\})})^{-1} \setminus \{\epsilon\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ 1 \leq i \leq \text{ar}(f')\} \\
= & \\
& ((\overline{(\Pi \setminus \{\pi\})})^{-1} \setminus \{\pi \cdot i \cdot \pi' \ : \ 1 \leq i \leq \text{ar}(f')\}) \setminus \{\epsilon\} \\
= & \\
& (\overline{(\Pi \setminus \{\pi\})} \cup \{\pi \cdot i \cdot \pi' \ : \ 1 \leq i \leq \text{ar}(f')\})^{-1} \setminus \{\epsilon\} \\
= & \\
& \overline{((\Pi \setminus \{\pi\}) \cup \{\pi \cdot i \ : \ 1 \leq i \leq \text{ar}(f')\})}^{-1} \setminus \{\epsilon\} \\
= & \\
& \mathfrak{P}((\Pi \setminus \{\pi\}) \cup \{\pi \cdot i \ : \ 1 \leq i \leq \text{ar}(f')\})
\end{aligned}
$$

Now, with Lemma E.8.4 and Lemma E.8.7 (with $\Pi' = \{\pi \cdot i \cdot \pi' \ : \ i > \text{ar}(f')\}$ and using Lemma E.8.6), we have that our original thrgh term is implied by $\text{thrgh}(\text{stgen}'(\text{stfilter}_\text{f}(\pi, f', R), (\Pi \setminus \{\pi\}) \cup \{\pi \cdot i \ : \ 1 \leq i \leq \text{ar}(f')\}), \mathfrak{S}(\text{stfilter}_\text{f}(\pi, f', R)), \mathfrak{R}(\text{stfilter}_\text{f}(\pi, f', R)), \mathfrak{P}((\Pi \setminus \{\pi\}) \cup \{\pi \cdot i \ : \ 1 \leq i \leq \text{ar}(f')\}))$, which holds by induction using the following. We have $\forall_{t \in \mathfrak{S}(\text{stfilter}_\text{f}(\pi, f', R)), \rho \in \mathfrak{R}(\text{stfilter}_\text{f}(\pi, f', R))}(\partial(t, l) \cap \mathfrak{P}(\Pi) \neq \emptyset \ \vee \ (\partial(t, l) = \emptyset \ \wedge \ \text{esspos}(l \to r \ \textbf{if} \ c) \subseteq \Pi \ \wedge \ \neg\exists_\sigma(t = l\sigma \ \wedge \ \text{true} \in \text{rewr}(c\sigma))))$ per definition of stfilter_f.

For the other conjunct we have the following.

$$
\begin{array}{l}
\quad \mathrm{thrgh}(\mathrm{stfunc}(\pi, R, \Pi)(\bot), \\
\qquad \{v \in \{t[u]_\pi \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi))\} \\
\qquad\quad : \ \pi \notin \mathrm{pos_f}(v)\}, \\
\qquad \{\rho \in \mathfrak{R}(R) \ : \ \pi \notin \mathrm{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \cup \\
\qquad\quad \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \pi \in \mathrm{pos_f}(l)\}, \\
\qquad ((\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \mathrm{true}\}) \cup \{\pi\}) \\
= \\
\quad \mathrm{thrgh}(\mathrm{stfunc}(\pi, R, \Pi)(\bot), \\
\qquad \{t[u]_\pi \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi)) \wedge \\
\qquad\quad \pi \notin \mathrm{pos_f}(t[u]_\pi)\}, \\
\qquad \{\rho \in \mathfrak{R}(R) \ : \ \pi \notin \mathrm{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \cup \\
\qquad\quad \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \pi \in \mathrm{pos_f}(l)\}, \\
\qquad (\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \mathrm{true}\}) \\
= \\
\quad \mathrm{thrgh}(\mathrm{stgen}'(\mathrm{stfilter_v}(\pi, R), \Pi \setminus \{\pi\}), \\
\qquad \{t[u]_\pi \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi)) \wedge \\
\qquad\quad \pi \notin \mathrm{pos_f}(t[u]_\pi)\}, \\
\qquad \{\rho \in \mathfrak{R}(R) \ : \ \pi \notin \mathrm{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \cup \\
\qquad\quad \{l \to r \ \mathbf{if} \ c \in R_f \ : \ \pi \in \mathrm{pos_f}(l)\}, \\
\qquad (\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \mathrm{true}\})
\end{array}
$$

We derive relations between the arguments of the thrgh expressions above and $\mathfrak{S}(\mathrm{stfilter_v}(\pi, R))$, $\mathfrak{R}(\mathrm{stfilter_v}(\pi, R))$ and $\mathfrak{P}(\Pi \setminus \{\pi\})$.

$$
\begin{array}{l}
\quad \{t[u]_\pi \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi)) \wedge \\
\qquad \pi \notin \mathrm{pos_f}(t[u]_\pi)\} \\
= \\
\quad \{t[u]_\pi \ : \ t \in \{t' \ : \ \neg \exists_{l \to r \ \mathbf{if} \ c \in R_f \setminus R} (t' = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma))\} \wedge \\
\qquad (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi)) \ \wedge \ \pi \notin \mathrm{pos_f}(t[u]_\pi)\} \\
= \\
\quad \{t[u]_\pi \ : \ \neg \exists_{l \to r \ \mathbf{if} \ c \in R_f \setminus R} (t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)) \wedge \\
\qquad (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi)) \ \wedge \ \pi \notin \mathrm{pos_f}(t[u]_\pi)\} \\
\subseteq \\
\quad \{t[u]_\pi \ : \ \neg \exists_{l \to r \ \mathbf{if} \ c \in R_f \setminus R} (t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)) \wedge \\
\qquad \pi \notin \mathrm{pos_f}(t[u]_\pi)\} \\
= \quad \{ \ \pi \in \Pi \ (\text{Lemma E.8.2}), \\
\qquad\quad \text{assumption } \forall_{\rho \in R_f \setminus R} (\mathrm{esspos}(\rho) \cap \Pi = \emptyset) \ \ \}
\end{array}
$$

$$
\begin{array}{ll}
 & \{t \;:\; \neg\exists_{l\to r \text{ if } c\in R_f\setminus R}(t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma)) \;\wedge\; \pi \notin \text{pos}_\text{f}(t)\} \\
\subseteq & \\
 & \{t \;:\; \neg\exists_{l\to r \text{ if } c\in R_f\setminus R}(t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma)) \wedge \\
 & \quad (\pi \notin \text{pos}_\text{f}(t) \;\vee\; \neg\exists_{l\to r \text{ if } c\in\{l\to r \text{ if } c\in R \,:\, \pi\in\text{pos}_\text{f}(l)\}}(t = l\sigma \wedge \\
 & \qquad \text{true} \in \text{rewr}(c\sigma)))\} \\
= & \\
 & \{t \;:\; \neg\exists_{l\to r \text{ if } c\in R_f\setminus R}(t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma)) \wedge \\
 & \quad \neg\exists_{l\to r \text{ if } c\in\{l\to r \text{ if } c\in R \,:\, \pi\in\text{pos}_\text{f}(l)\}}(\pi \in \text{pos}_\text{f}(t) \;\wedge\; t = l\sigma \wedge \\
 & \qquad \text{true} \in \text{rewr}(c\sigma))\} \\
= & \\
 & \{t \;:\; \neg\exists_{l\to r \text{ if } c\in R_f\setminus R}(t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma)) \wedge \\
 & \quad \neg\exists_{l\to r \text{ if } c\in\{l\to r \text{ if } c\in R \,:\, \pi\in\text{pos}_\text{f}(l)\}}(t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma))\} \\
= & \\
 & \{t \;:\; \neg\exists_{l\to r \text{ if } c\in R_f\setminus R}(t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma)) \wedge \\
 & \quad \neg\exists_{l\to r \text{ if } c\in R_f\cap\{l\to r \text{ if } c \,:\, \pi\in\text{pos}_\text{f}(l)\}}(t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma))\} \\
= & \\
 & \{t \;:\; \neg\exists_{l\to r \text{ if } c\in(R_f\setminus R)\cup(R_f\cap\{l\to r \text{ if } c \,:\, \pi\in\text{pos}_\text{f}(l)\})}(t = l\sigma \wedge \\
 & \quad \text{true} \in \text{rewr}(c\sigma))\} \\
= & \\
 & \{t \;:\; \neg\exists_{l\to r \text{ if } c\in R_f\setminus(R\setminus\{l\to r \text{ if } c \,:\, \pi\in\text{pos}_\text{f}(l)\})}(t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma))\} \\
= & \\
 & \{t \;:\; \neg\exists_{l\to r \text{ if } c\in R_f\setminus\{l\to r \text{ if } c\in R \,:\, \pi\notin\text{pos}_\text{f}(l)\}}(t = l\sigma \;\wedge\; \text{true} \in \text{rewr}(c\sigma))\} \\
= & \\
 & \mathfrak{S}(\{l \to r \textbf{ if } c \in R \;:\; \pi \notin \text{pos}_\text{f}(l)\}) \\
= & \\
 & \mathfrak{S}(\text{stfilter}_\text{v}(\pi, R))
\end{array}
$$

Next for $\mathfrak{R}$.

$$
\begin{array}{ll}
 & \{\rho \in \mathfrak{R}(R) \;:\; \pi \notin \text{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \cup \{l \to r \textbf{ if } c \in R_f \;:\; \pi \in \text{pos}_\text{f}(l)\} \\
= & \\
 & \{\rho \in R_f \setminus R \;:\; \pi \notin \text{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \cup \{l \to r \textbf{ if } c \in R_f \;:\; \pi \in \text{pos}_\text{f}(l)\} \\
= & \\
 & (\{\rho \in R_f \;:\; \pi \notin \text{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \setminus \{\rho \in R \;:\; \text{true}\}) \cup \\
 & \{l \to r \textbf{ if } c \in R_f \;:\; \pi \in \text{pos}_\text{f}(l)\} \\
= &
\end{array}
$$

$$
\begin{aligned}
& (\{\rho \in R_f \ : \ \pi \notin \mathrm{esspos}(\rho) \setminus \mathfrak{P}(\Pi)\} \cup \{l \to r \ \mathbf{if}\ c \in R_f \ : \ \pi \in \mathrm{pos_f}(l)\}) \setminus \\
& (\{\rho \in R \ : \ \mathrm{true}\} \setminus \{l \to r \ \mathbf{if}\ c \in R_f \ : \ \pi \in \mathrm{pos_f}(l)\}) \\
= \quad & \{ \text{ Assumption } R \subseteq R_f \ \} \\
& \{l \to r \ \mathbf{if}\ c \in R_f \ : \ \pi \notin \mathrm{esspos}(l \to r \ \mathbf{if}\ c) \setminus \mathfrak{P}(\Pi) \ \vee \ \pi \in \mathrm{pos_f}(l)\} \setminus \\
& \{l \to r \ \mathbf{if}\ c \in R \ : \ \pi \notin \mathrm{pos_f}(l)\} \\
= \quad & \\
& \{l \to r \ \mathbf{if}\ c \in R_f \ : \ \pi \notin \mathrm{esspos}(l \to r \ \mathbf{if}\ c) \setminus \mathfrak{P}(\Pi) \ \vee \ \pi \in \mathrm{pos_f}(l) \vee \\
& \quad (l \to r \ \mathbf{if}\ c \in R \ \wedge \ \pi \notin \mathrm{pos_f}(l))\} \setminus \\
& \{l \to r \ \mathbf{if}\ c \in R \ : \ \pi \notin \mathrm{pos_f}(l)\} \\
= \quad & \\
& \{l \to r \ \mathbf{if}\ c \in R_f \ : \ \pi \notin \mathrm{esspos}(l \to r \ \mathbf{if}\ c) \setminus \mathfrak{P}(\Pi) \ \vee \ \pi \in \mathrm{pos_f}(l) \vee \\
& \quad l \to r \ \mathbf{if}\ c \in R\} \setminus \\
& \{l \to r \ \mathbf{if}\ c \in R \ : \ \pi \notin \mathrm{pos_f}(l)\} \\
= \quad & \\
& \{l \to r \ \mathbf{if}\ c \in R_f \ : \ l \to r \ \mathbf{if}\ c \notin R \ \Rightarrow \ \pi \notin \mathrm{esspos}(l \to r \ \mathbf{if}\ c) \setminus \mathfrak{P}(\Pi) \vee \\
& \quad \pi \in \mathrm{pos_f}(l)\} \setminus \\
& \{l \to r \ \mathbf{if}\ c \in R \ : \ \pi \notin \mathrm{pos_f}(l)\} \\
= \quad & \{ \ \pi \in \Pi \text{ and assumption } \forall_{\rho \in R_f \setminus R}(\mathrm{esspos}(\rho) \cap \Pi = \emptyset) \ \} \\
& \{l \to r \ \mathbf{if}\ c \in R_f \ : \ \mathrm{true}\} \setminus \{l \to r \ \mathbf{if}\ c \in R \ : \ \pi \notin \mathrm{pos_f}(l)\} \\
= \quad & \\
& R_f \setminus \{l \to r \ \mathbf{if}\ c \in R \ : \ \pi \notin \mathrm{pos_f}(l)\} \\
= \quad & \\
& R_f \setminus \mathrm{stfilter_v}(\pi, R) \\
= \quad & \\
& \mathfrak{R}(\mathrm{stfilter_v}(\pi, R))
\end{aligned}
$$

Finally for $\mathfrak{P}$.

$$
\begin{aligned}
& (\mathfrak{P}(\Pi) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \mathrm{true}\} \\
= \quad & \\
& (((\overline{\Pi})^{-1} \setminus \{\epsilon\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \mathrm{true}\} \\
= \quad & \{ \ \pi \in \Pi \text{ (Lemma E.8.2)} \ \} \\
& (((\overline{(\Pi \setminus \{\pi\}) \cup \{\pi\}})^{-1} \setminus \{\epsilon\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \mathrm{true}\} \\
= \quad & \\
& (((\overline{\overline{\Pi \setminus \{\pi\}} \cup \{\pi \cdot \pi' \ : \ \mathrm{true}\}})^{-1} \setminus \{\epsilon\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \mathrm{true}\} \\
= \quad &
\end{aligned}
$$

$$
\begin{aligned}
& (((((\overline{(\Pi \setminus \{\pi\})})^{-1} \setminus \{\pi \cdot \pi' \ : \ \text{true}\}) \setminus \{\epsilon\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
= & \\
& (((((\overline{(\Pi \setminus \{\pi\})})^{-1} \setminus \{\epsilon\}) \setminus \{\pi \cdot \pi' \ : \ \text{true}\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
= & \\
& (((((\overline{(\Pi \setminus \{\pi\})})^{-1} \setminus \{\epsilon\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
= & \\
& ((((\overline{(\Pi \setminus \{\pi\})})^{-1} \setminus \{\epsilon\}) \cup \{\pi\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
= & \\
& ((((\overline{(\Pi \setminus \{\pi\})})^{-1} \cup \{\pi\}) \setminus \{\epsilon\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
= & \quad \{\ \pi \in \Pi \text{ (Lemma E.8.2)}, \\
& \qquad \text{assumption that } \Pi \text{ is non-overlapping } \} \\
& ((\overline{(\Pi \setminus \{\pi\})})^{-1} \setminus \{\epsilon\}) \setminus \{\pi \cdot i \cdot \pi' \ : \ \text{true}\} \\
\subseteq & \\
& (\overline{\Pi \setminus \{\pi\}})^{-1} \setminus \{\epsilon\} \\
= & \\
& \mathfrak{P}(\Pi \setminus \{\pi\})
\end{aligned}
$$

As with the previous case, Lemma E.8.4 and Lemma E.8.7 (with $\Pi' = \{\pi \cdot i \cdot \pi' \ : \ \text{true}\}$ and Lemma E.8.6) give us the original thrgh term is implied by $\text{thrgh}(\text{stgen}'(\text{stfilter}_{\text{v}}(\pi, R), \Pi \setminus \{\pi\}), \mathfrak{S}(\text{stfilter}_{\text{v}}(\pi, R)), \mathfrak{R}(\text{stfilter}_{\text{v}}(\pi, R)), \mathfrak{P}(\Pi \setminus \{\pi\}))$, which holds by induction (using the assumption which holds per definition of $\text{stfilter}_{\text{v}}$).

- $\text{stready}(R, \Pi) = \emptyset \ \wedge \ \text{need}^w_{\text{f}}(R, \Pi) = \emptyset \ \wedge \ \pi = \iota.\text{need}^w_{\text{v}}(R, \Pi)$; this means

$$
\begin{aligned}
& \text{thrgh}(\text{stgen}'(R, \Pi), \mathfrak{S}(R), \mathfrak{R}(R), \mathfrak{P}(\Pi)) \\
= & \\
& \text{thrgh}(\mathsf{NF}(\{\pi\}, \text{stgen}'(R, \Pi \setminus \{\pi\})), \mathfrak{S}(R), \mathfrak{R}(R), \mathfrak{P}(\Pi)) \\
= & \\
& \text{thrgh}(\text{stgen}'(R, \Pi \setminus \{\pi\}), \\
& \quad \{t[\psi]_{\{\pi\}} \ : \ t \in \mathfrak{S}(R) \ \wedge \ \forall_{\pi' \in \{\pi\}}(\pi' \in \text{pos}(t) \ \Rightarrow \ \psi(\pi') \in \text{rewr}(t|_{\pi'}))\}, \\
& \quad \{\rho \in \mathfrak{R}(R) \ : \ \{\pi\} \cap \text{esspos}(\rho) \subseteq \{\pi' \ : \ \overline{\{\pi'\}} \subseteq \mathfrak{P}(\Pi)\}\}, \\
& \quad \mathfrak{P}(\Pi) \cup \overline{\{\pi\}})
\end{aligned}
$$

We derive relations between the arguments of the thrgh above and $\mathfrak{S}(R)$, $\mathfrak{R}(R)$ and $\mathfrak{P}(\Pi \setminus \{\pi\})$.

$$
\begin{aligned}
& \{t[\psi]_{\{\pi\}} \ : \ t \in \mathfrak{S}(R) \ \wedge \ \forall_{\pi' \in \{\pi\}}(\pi' \in \text{pos}(t) \ \Rightarrow \ \psi(\pi') \in \text{rewr}(t|_{\pi'}))\} \\
= &
\end{aligned}
$$

$$
\begin{array}{ll}
& \{t[\psi(\pi)]_\pi \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \mathrm{pos}(t) \ \Rightarrow \ \psi(\pi) \in \mathrm{rewr}(t|_\pi))\} \\
= & \\
& \{t[u]_\pi \ : \ t \in \mathfrak{S}(R) \ \wedge \ (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi))\} \\
= & \\
& \{t[u]_\pi \ : \ t \in \{t' \ : \ \neg\exists_{l \to r \ \mathbf{if} \ c \in R_f \setminus R}(t' = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma))\} \wedge \\
& \quad (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi))\} \\
= & \\
& \{t[u]_\pi \ : \ \neg\exists_{l \to r \ \mathbf{if} \ c \in R_f \setminus R}(t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)) \wedge \\
& \quad (\pi \in \mathrm{pos}(t) \ \Rightarrow \ u \in \mathrm{rewr}(t|_\pi))\} \\
\subseteq & \\
& \{t \ : \ \neg\exists_{l \to r \ \mathbf{if} \ c \in R_f \setminus R}(t' = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma))\} \\
= & \\
& \mathfrak{S}(R)
\end{array}
$$

Next of $\mathfrak{S}$.

$$
\begin{array}{ll}
& \{\rho \in \mathfrak{R}(R) \ : \ \{\pi\} \cap \mathrm{esspos}(\rho) \subseteq \{\pi' \ : \ \overline{\{\pi'\}} \subseteq \mathfrak{P}(\Pi)\}\} \\
= & \\
& \{\rho \in R_f \setminus R \ : \ \pi \notin \mathrm{esspos}(\rho) \setminus \{\pi' \ : \ \overline{\{\pi'\}} \subseteq \mathfrak{P}(\Pi)\}\} \\
= & \quad \{ \ \pi \in \Pi \text{ (Lemma E.8.2),} \\
& \quad \text{assumption } \forall_{\rho \in R_f \setminus R}(\mathrm{esspos}(\rho) \cap \Pi = \emptyset) \ \} \\
& R_f \setminus R \\
= & \\
& \mathfrak{R}(R)
\end{array}
$$

Finally for $\mathfrak{P}$.

$$
\begin{array}{ll}
& \mathfrak{P}(\Pi) \cup \overline{\{\pi\}} \\
= & \\
& ((\overline{\Pi})^{-1} \setminus \{\epsilon\}) \cup \overline{\{\pi\}} \\
= & \quad \{ \ \pi \neq \epsilon \text{ (via } \pi \in \Pi\text{, Lemma E.8.3, and assumption } \epsilon \notin \Pi) \ \} \\
& ((\overline{\Pi})^{-1} \cup \overline{\{\pi\}}) \setminus \{\epsilon\} \\
= & \\
& (\overline{\Pi} \cap (\overline{\{\pi\}})^{-1})^{-1} \setminus \{\epsilon\} \\
= & \\
& (\overline{\Pi} \setminus \overline{\{\pi\}})^{-1} \setminus \{\epsilon\} \\
= & \quad \{ \text{ Assumption that } \Pi \text{ is non-overlapping } \}
\end{array}
$$

$$
\begin{array}{l}
\quad (\overline{\Pi \setminus \{\pi\}})^{-1} \setminus \{\epsilon\} \\
= \\
\quad \mathfrak{P}(\Pi \setminus \{\pi\})
\end{array}
$$

With Lemma E.8.4 we get that our original thrgh expression is implied by $\mathrm{thrgh}(\mathrm{stgen}'(R, \Pi \setminus \{\pi\}), \mathfrak{S}(R), \mathfrak{R}(R), \mathfrak{P}(\Pi \setminus \{\pi\}))$, which holds by induction.

The case for $\mathrm{thrgh}_{\mathrm{h}}(\mathrm{stgen}_{\mathrm{h}}(R_f), \{f(t_1, \ldots, t_n) \ : \ \mathrm{true}\}, \emptyset, \emptyset)$ is the similar. We only look at the base case.

$$
\begin{array}{ll}
 & \mathrm{thrgh}_{\mathrm{h}}(\mathrm{stgen}'_{\mathrm{h}}(\emptyset, \Pi), \mathfrak{S}(\emptyset), \mathfrak{R}(\emptyset), \mathfrak{P}(\Pi)) \\
= & \\
 & \mathrm{thrgh}_{\mathrm{h}}(\mathsf{E}, \mathfrak{S}(\emptyset), \mathfrak{R}(\emptyset), \mathfrak{P}(\Pi)) \\
= & \\
 & \forall_{t \in \mathfrak{S}(\emptyset), l \to r \ \mathbf{if} \ c \in \mathfrak{R}(\emptyset)} (\partial(t, l) \cap \mathfrak{P}(\Pi) \neq \emptyset \vee \\
 & \quad (\partial(t, l) = \emptyset \ \wedge \ \mathrm{esspos}(l \to r \ \mathbf{if} \ c) \subseteq \mathfrak{P}(\Pi) \wedge \\
 & \qquad \neg \exists_\sigma (t = l\sigma \ \wedge \ \mathrm{true} \in \mathrm{rewr}(c\sigma)))) \wedge \\
 & (\mathfrak{S}(\emptyset) \neq \emptyset \ \Rightarrow \ R_f \subseteq \mathfrak{R}(R)) \\
= & \quad \{ \text{ Assumption } \} \\
 & \mathrm{true}
\end{array}
$$

Appendix F

Benchmarks

In this appendix we give the specifications of the benchmarks we use in Chapter 6. Here we leave the signature implicit as it is trivially deduced from the rewrite rules. Note that we write 0 instead of 0() even though it is a function (of arity 0).

F.1 Prioritised *eq*

The rewrite rules for this benchmark are from [BBKW89] and are as follows and the priority is such that when both can be applied the first will be applied.

$$\begin{array}{lcl} eq(x,x) & \to & \text{true} \\ eq(x,y) & \to & \text{false} \end{array}$$

F.2 Prioritised *fac*

The rewrite rules for this benchmark are from [BBKW89] and are as follows and the priority is such that when both can be applied the first will be applied.

$$\begin{array}{lcl} fac(0) & \to & S(0) \\ fac(x) & \to & mult(x, fac(P(x))) \end{array}$$

F.3 *fib*(15)

The set of rewrite rules for this benchmark is as follows.

$$\begin{array}{lcl} plus(n,0) & \to & n \\ plus(n,S(m)) & \to & S(plus(n,m)) \end{array}$$

(continued on next page)

$$\begin{array}{lcl} \mathit{fib}(0) & \to & 0 \\ \mathit{fib}(S(0)) & \to & S(0) \\ \mathit{fib}(S(S(n))) & \to & \mathit{plus}(\mathit{fib}(n), \mathit{fib}(S(n))) \end{array}$$

The term that is rewritten to normal form for this benchmark is the following.

$$\mathit{fib}(S(S(S(S(S(\ S(S(S(S(S(\ S(S(S(S(S(0))))))))))))))))$$

F.4 $\mathit{evalexp}(5)$

For this benchmark we separate the rewrite rules in three sets such that they can be easily reused in other benchmarks. First we have the "basic eval" rules.

$$\begin{array}{lcl} \mathit{if}(\mathit{true}(), x, y) & \to & x \\ \mathit{if}(\mathit{false}(), x, y) & \to & y \\ \mathit{if}(b, x, x) & \to & x \\ \\ \mathit{and}(\mathit{true}(), x) & \to & x \\ \mathit{and}(\mathit{false}(), x) & \to & \mathit{false}() \\ \mathit{and}(x, \mathit{true}()) & \to & x \\ \mathit{and}(x, \mathit{false}()) & \to & \mathit{false}() \\ \\ \mathit{eq}(x, x) & \to & \mathit{true}() \\ \mathit{eq}(Z(), S(n)) & \to & \mathit{false}() \\ \mathit{eq}(S(n), Z()) & \to & \mathit{false}() \\ \mathit{eq}(S(n), S(m)) & \to & \mathit{eq}(n, m) \\ \\ \mathit{succ}17(n) & \to & \mathit{if}(\mathit{eq}(n, S(S(S(S(S(\ S(S(S(S(\\ & & \quad S(S(S(S(S(\ S(Z())))))))))))))))), Z(), S(n)) \\ \mathit{plus}17(n, Z()) & \to & n \\ \mathit{plus}17(n, S(m)) & \to & \mathit{succ}17(\mathit{plus}17(n, m)) \\ \mathit{mult}17(n, Z()) & \to & Z() \\ \mathit{mult}17(n, S(m)) & \to & \mathit{plus}17(n, \mathit{mult}17(n, m)) \\ \mathit{exp}17(n, Z()) & \to & \mathit{succ}17(Z()) \\ \mathit{exp}17(n, S(m)) & \to & \mathit{mult}17(n, \mathit{exp}17(n, m)) \end{array}$$

Secondly, we have the "expression" rules.

$$\begin{array}{lcl} \mathit{eq}(\mathit{Exz}, \mathit{Exs}(e)) & \to & \mathit{false}() \\ \mathit{eq}(\mathit{Exz}, \mathit{Explus}(e, e2)) & \to & \mathit{false}() \\ \mathit{eq}(\mathit{Exz}, \mathit{Exmult}(e, e2)) & \to & \mathit{false}() \\ \mathit{eq}(\mathit{Exz}, \mathit{Exexp}(e, e2)) & \to & \mathit{false}() \\ \mathit{eq}(\mathit{Exs}(f), \mathit{Exz}()) & \to & \mathit{false}() \end{array}$$

$$
\begin{array}{lcl}
eq(Exs(f), Exs(e)) & \rightarrow & eq(f, e) \\
eq(Exs(f), Explus(e, e2)) & \rightarrow & false() \\
eq(Exs(f), Exmult(e, e2)) & \rightarrow & false() \\
eq(Exs(f), Exexp(e, e2)) & \rightarrow & false() \\
eq(Explus(f, f2), Exz()) & \rightarrow & false() \\
eq(Explus(f, f2), Exs(e)) & \rightarrow & false() \\
eq(Explus(f, f2), Explus(e, e2)) & \rightarrow & and(eq(f, e), eq(f2, e2)) \\
eq(Explus(f, f2), Exmult(e, e2)) & \rightarrow & false() \\
eq(Explus(f, f2), Exexp(e, e2)) & \rightarrow & false() \\
eq(Exmult(f, f2), Exz()) & \rightarrow & false() \\
eq(Exmult(f, f2), Exs(e)) & \rightarrow & false() \\
eq(Exmult(f, f2), Explus(e, e2)) & \rightarrow & false() \\
eq(Exmult(f, f2), Exmult(e, e2)) & \rightarrow & and(eq(f, e), eq(f2, e2)) \\
eq(Exmult(f, f2), Exexp(e, e2)) & \rightarrow & false() \\
eq(Exexp(f, f2), Exz()) & \rightarrow & false() \\
eq(Exexp(f, f2), Exs(e)) & \rightarrow & false() \\
eq(Exexp(f, f2), Explus(e, e2)) & \rightarrow & false() \\
eq(Exexp(f, f2), Exmult(e, e2)) & \rightarrow & false() \\
eq(Exexp(f, f2), Exexp(e, e2)) & \rightarrow & and(eq(f, e), eq(f2, e2))
\end{array}
$$

$$
\begin{array}{lcl}
eval17(Exz()) & \rightarrow & Z() \\
eval17(Exs(n)) & \rightarrow & succ17(eval17(n)) \\
eval17(Explus(n, m)) & \rightarrow & plus17(eval17(n), eval17(m)) \\
eval17(Exmult(n, m)) & \rightarrow & mult17(eval17(n), eval17(m)) \\
eval17(Exexp(n, m)) & \rightarrow & exp17(eval17(n), eval17(m))
\end{array}
$$

The last set of rules is specific to this benchmark and consist of the following rules.

$$
\begin{array}{lcl}
expand(Exz()) & \rightarrow & Exz() \\
expand(Exs(n)) & \rightarrow & Explus(Exs(Exz()), n) \\
expand(Explus(n, m)) & \rightarrow & Explus(expand(n), expand(m)) \\
expand(Exmult(n, Exz())) & \rightarrow & Exz() \\
expand(Exmult(n, Exs(Exz()))) & \rightarrow & expand(n) \\
expand(Exmult(n, Exs(Exs(m)))) & \rightarrow & expand(Exmult(n, \\
 & & \quad expand(Exs(Exs(m))))) \\
expand(Exmult(n, Exs(Explus(m, o)))) & \rightarrow & expand(Exmult(n, \\
 & & \quad expand(Exs(Explus(m, o))))) \\
expand(Exmult(n, Exs(Exmult(m, o)))) & \rightarrow & expand(Exmult(n, \\
 & & \quad expand(Exs(Exmult(m, o))))) \\
expand(Exmult(n, Exs(Exexp(m, o)))) & \rightarrow & expand(Exmult(n, \\
 & & \quad expand(Exs(Exexp(m, o))))) \\
expand(Exmult(n, Explus(m, o))) & \rightarrow & expand(Explus(Exmult(n, m), \\
 & & \quad Exmult(n, o)))
\end{array}
$$

(continued on next page)

$$
\begin{array}{lcl}
expand(Exmult(n, Exmult(m, o))) & \to & expand(Exmult(n, \\
 & & \quad expand(Exmult(m, o)))) \\
expand(Exmult(n, Exexp(m, o))) & \to & expand(Exmult(n, \\
 & & \quad expand(Exexp(m, o)))) \\
expand(Exexp(n, Exz())) & \to & Exs(Exz()) \\
expand(Exexp(n, Exs(Exz()))) & \to & expand(n) \\
expand(Exexp(n, Exs(Exs(m)))) & \to & expand(Exexp(n, \\
 & & \quad expand(Exs(Exs(m))))) \\
expand(Exexp(n, Exs(Explus(m, o)))) & \to & expand(Exexp(n, \\
 & & \quad expand(Exs(Explus(m, o))))) \\
expand(Exexp(n, Exs(Exmult(m, o)))) & \to & expand(Exexp(n, \\
 & & \quad expand(Exs(Exmult(m, o))))) \\
expand(Exexp(n, Exs(Exexp(m, o)))) & \to & expand(Exexp(n, \\
 & & \quad expand(Exs(Exexp(m, o))))) \\
expand(Exexp(n, Explus(m, o))) & \to & expand(Exmult(Exexp(n, m), \\
 & & \quad Exexp(n, o))) \\
expand(Exexp(n, Exmult(m, o))) & \to & expand(Exexp(n, \\
 & & \quad expand(Exmult(m, o)))) \\
expand(Exexp(n, Exexp(m, o))) & \to & expand(Exexp(n, \\
 & & \quad expand(Exexp(m, o)))) \\
 & & \\
evalexp17(n) & \to & eval17(expand(n)) \\
 & & \\
two() & \to & Exs(Exs(Exz())) \\
f(x) & \to & eq(eval17(Exexp(two(), x)), \\
 & & \quad evalexp17(Exexp(two(), x)))
\end{array}
$$

The term that is rewritten to normal form for this benchmark is the following.

$$f(Exs(Exs(Exs(Exs(Exs(Exs(Exz())))))))$$

F.5 *evaltree*(5)

This benchmark uses the "basic eval" and "expression" rules of Section F.4 in combination with the following rules.

$$
\begin{array}{lcl}
evaltree17(n) & \to & mult17(exp17(S(S(Z())), pred17(n)), \\
 & & \quad pred17(exp17(S(S(Z())), n))) \\
 & & \\
getval(leaf(val)) & \to & val \\
getval(node(val, max, t, u)) & \to & val
\end{array}
$$

$$
\begin{array}{lcl}
getmax(leaf(val)) & \to & val \\
getmax(node(val, max, t, u)) & \to & max \\
\\
buildtree(Z(), n) & \to & leaf(n) \\
\\
tmp_4(left, leftval, right, & & \\
\quad rightval, max) & \to & node(plus17(leftval, rightval), max, \\
 & & \quad left, right) \\
tmp_3(left, leftval, right) & \to & tmp_4(left, leftval, right, getval(right), \\
 & & \quad getmax(right)) \\
tmp_2(n, left, leftmax, leftval) & \to & tmp_3(left, leftval, \\
 & & \quad buildtree(n, succ17(leftmax))) \\
tmp_1(n, left) & \to & tmp_2(n, left, getmax(left), \\
 & & \quad getval(left)) \\
buildtree(S(n), m) & \to & tmp_1(n, buildtree(n, m)) \\
f(x) & \to & eq(evaltree17(x), \\
 & & \quad getval(buildtree(x, Z())))
\end{array}
$$

The term that is rewritten to normal form for this benchmark is the following.

$$f(S(S(S(S(S(Z()))))))$$

F.6 *evalsym*(5)

This benchmark uses the "basic eval" and "expression" rules of Section F.4 in combination with the following rules.

$$
\begin{array}{ll}
evalsym17(Exz()) & \to Z() \\
evalsym17(Exs(n)) & \to succ17(evalsym17(n)) \\
evalsym17(Explus(n, m)) & \to plus17(evalsym17(n), evalsym17(m)) \\
evalsym17(Exmult(n, Exz())) & \to Z() \\
evalsym17(Exmult(n, Exs(m))) & \to evalsym17(Explus(Exmult(n, m), n)) \\
evalsym17(Exmult(n, Explus(m, o))) & \to evalsym17(Explus(Exmult(n, m), \\
 & \qquad Exmult(n, o))) \\
evalsym17(Exmult(n, Exmult(m, o))) & \to evalsym17(Exmult(Exmult(n, m), o)) \\
evalsym17(Exmult(n, Exexp(m, o))) & \to evalsym17(Exmult(n, dec(Exexp(m, o)))) \\
evalsym17(Exexp(n, Exz())) & \to succ17(Z()) \\
evalsym17(Exexp(n, Exs(m))) & \to evalsym17(Exmult(Exexp(n, m), n)) \\
evalsym17(Exexp(n, Explus(m, o))) & \to evalsym17(Exmult(Exexp(n, m), \\
 & \qquad Exexp(n, o))) \\
evalsym17(Exexp(n, Exmult(m, o))) & \to evalsym17(Exexp(Exexp(n, m), o)) \\
evalsym17(Exexp(n, Exexp(m, o))) & \to evalsym17(Exexp(n, dec(Exexp(m, o))))
\end{array}
$$

(continued on next page)

$$
\begin{array}{ll}
dec(Exexp(n, Exz())) & \to Exs(Exz()) \\
dec(Exexp(n, Exs(m))) & \to Exmult(Exexp(n, m), n) \\
dec(Exexp(n, Explus(m, o))) & \to Exmult(Exexp(n, m), Exexp(n, o)) \\
dec(Exexp(n, Exmult(m, o))) & \to dec(Exexp(Exexp(n, m), o)) \\
dec(Exexp(n, Exexp(m, o))) & \to dec(Exexp(n, dec(Exexp(m, o)))) \\
 & \\
two() & \to Exs(Exs(Exz())) \\
f(x) & \to eq(eval17(Exexp(two(), x)), \\
 & \qquad evalsym17(Exexp(two(), x)))
\end{array}
$$

The term that is rewritten to normal form for this benchmark is the following.

$$f(Exs(Exs(Exs(Exs(Exs(Exz()))))))$$

F.7 set add

The set of rewrite rules for this benchmark is as follows.

$$
\begin{array}{lcl}
if(true(), x, y) & \to & x \\
if(false(), x, y) & \to & y \\
if(b, x, x) & \to & x \\
 & & \\
le(0, n) & \to & true() \\
le(S(n), 0) & \to & false() \\
le(S(n), S(m)) & \to & le(n, m) \\
 & & \\
insert(n, empty()) & \to & cons(n, empty()) \\
insert(n, cons(m, l)) & \to & if(le(n, m), \\
 & & \quad if(le(m, n), \\
 & & \quad\quad cons(m, l), \\
 & & \quad\quad cons(n, cons(m, l))), \\
 & & \quad cons(m, insert(n, l)))
\end{array}
$$

The term that is rewritten to normal form for this benchmark is the following.

$$
\begin{array}{l}
insert(S(S(S(S(S(S(S(S(0)))))))),\\
\quad cons(0,\\
\quad cons(S(0),\\
\quad cons(S(S(0)),\\
\quad cons(S(S(S(0))),\\
\quad cons(S(S(S(S(0)))),\\
\quad cons(S(S(S(S(S(0))))),\\
\quad cons(S(S(S(S(S(S(0)))))),\\
\quad cons(S(S(S(S(S(S(S(0))))))),\\
\quad cons(S(S(S(S(S(S(S(S(S(S(S(S(0))))))))))))),\\
\quad empty())))))))))))
\end{array}
$$

F.8 all even

The set of rewrite rules for this benchmark is as follows.

$$
\begin{array}{lcl}
and(true(), b) & \rightarrow & b\\
and(false(), b) & \rightarrow & false()\\
\\
even(0) & \rightarrow & true()\\
even(S(n)) & \rightarrow & odd(n)\\
\\
odd(0) & \rightarrow & false()\\
odd(S(n)) & \rightarrow & even(n)\\
\\
alleven(empty()) & \rightarrow & true()\\
alleven(cons(n, l)) & \rightarrow & and(even(n), alleven(l))
\end{array}
$$

The term that is rewritten to normal form for this benchmark is the following.

$$
\begin{array}{l}
alleven(\\
\quad cons(S(S(0)),\\
\quad cons(S(S(S(S(0)))),\\
\quad cons(S(S(S(S(S(0))))),\\
\quad cons(0,\\
\quad cons(S(S(0)),\\
\quad cons(S(S(S(0))),\\
\quad cons(S(S(S(S(S(S(0)))))),\\
\quad empty()))))))))
\end{array}
$$

F.9 exp peano

The set of rewrite rules for this benchmark is as follows.

$$\begin{array}{lcl} exp(n, 0) & \to & S(0) \\ exp(n, S(m)) & \to & mult(exp(n, m), n) \\ \\ mult(n, 0) & \to & 0 \\ mult(n, S(m)) & \to & plus(mult(n, m), n) \\ \\ plus(n, 0) & \to & n \\ plus(n, S(m)) & \to & S(plus(n, m)) \end{array}$$

The term that is rewritten to normal form for this benchmark is the following.

$$exp(S(S(0())), S(S(S(0()))))$$

F.10 exp binary

The set of rewrite rules for this benchmark is as follows.

$$\begin{array}{lcl} not(true()) & \to & false() \\ not(false()) & \to & true() \\ \\ succ(1) & \to & dub(false(), 1) \\ succ(dub(false(), p)) & \to & dub(true(), p) \\ succ(dub(true(), p)) & \to & dub(false(), succ(p)) \\ \\ addc(false(), 1, p) & \to & succ(p) \\ addc(true(), 1, p) & \to & succ(succ(p)) \\ addc(false(), p, 1) & \to & succ(p) \\ addc(true(), p, 1) & \to & succ(succ(p)) \\ addc(b, dub(c, p), dub(c, q)) & \to & dub(b, addc(c, p, q)) \\ addc(b, dub(false(), p), dub(true(), q)) & \to & dub(not(b), addc(b, p, q)) \\ addc(b, dub(true(), p), dub(false(), q)) & \to & dub(not(b), addc(b, p, q)) \\ \\ multir(false(), p, 1, q) & \to & q \\ multir(true(), p, 1, q) & \to & addc(false(), p, q) \\ multir(b, p, dub(false(), q), r) & \to & multir(b, p, q, dub(false(), r)) \\ multir(false(), p, dub(true(), q), r) & \to & multir(true(), r, q, dub(false(), r)) \\ multir(true(), p, dub(true(), q), r) & \to & multir(true(), addc(false(), p, r), \\ & & \quad q, dub(false(), r)) \\ \\ exp(p, 0) & \to & 1 \end{array}$$

$$
\begin{array}{lcl}
exp(p, nat(1)) & \rightarrow & p \\
exp(p, nat(dub(false(), q))) & \rightarrow & exp(multir(false(), 1, p, p), nat(q)) \\
exp(p, nat(dub(true(), q))) & \rightarrow & multir(false(), 1, p, \\
 & & \quad exp(multir(false(), 1, p, p), nat(q)))
\end{array}
$$

The term that is rewritten to normal form for this benchmark is the following.

$$exp(dub(false(), 1), nat(dub(true(), dub(false(), dub(true(), dub(false(), 1))))))$$

F.11 higher-order binary search

The mCRL2 data specification for this benchmark is given below. The evaluated term is $bs(3435, f, 100000)$.

```
map f: Nat -> Nat;
    bs: Nat#(Nat -> Nat)#Nat -> Nat;
    bs2: Nat#(Nat -> Nat)#Nat#Nat -> Nat;

var n,m,x,y: Nat;
    g: Nat -> Nat;
    b: Bool;
eqn f(n) = n*n;
    bs(n,g,m) = bs2(n,g,0,m);
    bs2(n,g,x,y) = if(
                     x+1 == y,
                     x,
                     if(
                       f(h) < n,
                       bs2(n,g,h,y),
                       bs2(n,g,x,h),
                     ) whr h = (x+y) div 2 end
                   );
```

Bibliography

[Aug85] L. Augustsson. Compiling pattern matching. In J.-P. Jouannaud, editor, *Proceedings of a Conference on Functional Programming Languages and Computer Architecture (FPCA)*, volume 523 of *Lecture Notes in Computer Science*, pages 368–381, 1985. [2, 3, 19]

[BBK87] J.C.M. Baeten, J.A. Bergsta, and J.W. Klop. Term rewriting systems with priorities. In Pierre Lescanne, editor, *Rewriting Techniques and Applications, proceedings (RTA'87)*, volume 256 of *Lecture Notes in Computer Science*, pages 83–94, 1987. [37]

[BBKW89] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, and W.P. Weijland. Term-rewriting systems with rule priorities. *Theoretical Computer Science*, 67(2-3):283–301, 1989. [80, 155]

[BFG+01] S.C.C. Blom, W.J. Fokkink, J.F. Groote, I. van Langevelde, B. Lisser, and J.C. van de Pol. μCRL: A toolset for analysing algebraic specifications. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification: 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, volume 2102 of *Lecture Notes in Computer Science*, pages 250–254, 2001. [2, 85]

[BK86] J.A. Bergstra and J.W. Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32(3):323–362, 1986. [9]

[BvEG+87] H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer, and M.R. Sleep. Term graph rewriting. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *PARLE Parallel Architectures and Languages Europe*, volume 259 of *Lecture Notes in Computer Science*, pages 141–158, 1987. [2]

[CDE+02] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J.F. Quesada. Maude: specification and program-

ming in rewriting logic. *Theoretical Computer Science*, 285(2):187–243, 2002. [85]

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 243–320. MIT Press, 1990. [9]

[FKW00] W.J. Fokkink, J. Kamperman, and P. Walters. Lazy rewriting on eager machinery. *ACM Transactions on Programming Languages and Systems*, 22(1):45–86, 2000. [60]

[FW76] D.P. Friedman and D.S. Wise. Cons should not evaluate its arguments. In S. Michaelson and R. Milner, editors, *International Colloquium on Automata, Languages and Programming (ICALP '76)*, pages 257–284, 1976. [3]

[GK04] J.F. Groote and M. Keinänen. Solving disjunctive/conjunctive boolean equation systems with alternating fixed points. In K. Jensen and A. Podelski, editors, *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *Lecture Notes in Computer Science*, pages 436–450, 2004. [56]

[GLM02] H. Garavel, F. Lang, and R. Mateescu. An overview of CADP 2001. *European Association for Software Science and Technology (EASST) Newsletter*, 4, 2002. [85]

[GMR+08] J.F. Groote, A.H.J. Mathijssen, M.A. Reniers, Y.S. Usenko, and M.J. van Weerdenburg. Analysis of distributed systems with mCRL2. In M. Alexander and W. Gardner, editors, *Process Algebra for Parallel and Distributed Processing*, Computational Science Series, chapter 4, pages 99–128. Chapman and Hall/CRC, 2008. [2]

[GMvWU07] J.F. Groote, A.H.J. Mathijssen, M.J. van Weerdenburg, and Y.S. Usenko. The formal specification language mCRL2. In E. Brinksma, D. Harel, A. Mader, P. Stevens, and R. Wieringa, editors, *Methods for Modelling Software Systems (MMOSS)*, number 06351 in Dagstuhl Seminar Proceedings, 2007. [2]

[GWJMJ00] J.A. Goguen, T. Winkler, and K. Futatsugi J. Meseguer, and J.-P. Jouannaud. Introducing obj. In J.A. Goguen and G. Malcolm, editors, *Software Engineering with OBJ: Algebraic Specification in Action*, Advances in Formal Methods, pages 3–167. Kluwer Academic Publishers, 2000. [4, 5, 60]

[GWM+93] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In Joseph Goguen, editor, *Applications of Algebraic Specification using OBJ*. Cambridge, 1993. [4]

[HFA+96] P.H. Hartel, M. Feeley, M. Alt, L. Augustsson, P. Baumann, M. Beemster, E. Chailloux, C.H. Flood, W. Grieskamp, J.H.G. Groningen, and K. Hammond. Benchmarking implementations of functional languages with "pseudoknot", a float-intensive benchmark. *Journal of Functional Programming*, 6:621–655, 1996. [3]

[HHPW07] P. Hudak, J. Hughes, S.L. Peyton Jones, and P. Wadler. A history of Haskell: being lazy with class. In *The Third ACM SIGPLAN History of Programming Languages Conference (HOPL-III)*, pages 12–1–12–55, 2007. [2]

[HJ76] P. Henderson and J.H. Morris Jr. A lazy evaluator. In *POPL '76: Proceedings of the 3rd ACM SIGACT-SIGPLAN symposium on Principles on programming languages*, pages 95–103, 1976. [3]

[HL91] G.P. Huet and J.-J. Lévy. Computations in orthogonal rewriting systems, I and II. In *Computational Logic - Essays in Honor of Alan Robinson*, pages 395–443, 1991. [54]

[LNS82] J.-L. Lassez, V.L. Nguyen, and E.A. Sonenberg. Fixed point theorems and semantics: a folk tale. *Information Processing Letters*, 14(3):112–116, 1982. [92]

[LS93] J. Launchbury and P.M. Sansom, editors. *The Glasgow Haskell Compiler: A Retrospective.*, Workshops in Computing, 1993. [2, 85]

[Mad97] A. Mader. *Verification of Modal Properties using Boolean Equation Systems.* PhD thesis, Technical University of Munich, 1997. [55, 56]

[Mar92] L. Maranget. Compiling lazy pattern matching. In *Proceedings of the 1992 conference on Lisp and Functional Programming*, pages 21–31, 1992. [3, 19]

[Mat03] R. Mateescu. A generic on-the-fly solver for alternation-free boolean equation systems. In H. Garavel and J. Hatcliff, editors, *Proceedings of the 9th International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 81–96, 2003. [56]

[Nip89] T. Nipkow. Term rewriting and beyond – theorem proving in Isabelle. *Formal Aspects of Computing*, 1/1:320–338, 1989. [2]

[OF97] K. Ogata and K. Futatsugi. Implementation of term rewritings with the evaluation strategy. In H. Glaser, P.H. Hartel, and H. Kuchen, editors, *Proceedings of the 9th International Symposium on Programming Languages: Implementations, Logics, and*

Programs, PLILP'97, volume 1292 of *Lecture Notes in Computer Science*, pages 225–239, 1997. [4, 60]

[OF00] Kazuhiro Ogata and Kockichi Futatsugi. Operational semantics of rewriting with the on-demand evaluation strategy. In *Proceedings of the 2000 ACM symposium on Applied computing (SAC '00)*, pages 756–763, 2000. [5]

[Oli00] P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. PhD thesis, University of Amsterdam, 2000. [81]

[Pey87] S.L. Peyton Jones. *The implementation of functional programming languages*. Prentice-Hall, 1987. [2, 3, 19]

[Pey03] S.L. Peyton Jones, editor. *Haskell 98 Language and Libraries: The Revised Report*. Cambridge, 2003. [2]

[Pla95] M.J. Plasmeijer. Clean: a programming environment based on term graph rewriting. *Electronic Notes in Theoretical Computer Science*, 2:215–221, 1995. [2, 85]

[PvE93] R. Plasmeijer and M. van Eekelen. *Functional Programming and Parallel Graph Rewriting*. Addison-Wesley Longman Publishing Co., Inc., 1993. [4, 54]

[Sch88] Ph. Schnoebelen. Refined compilation of pattern-matching for functional languages. *Science of Computer Programming*, 11(2):133–159, 1988. [2, 3, 19]

[Too] mCRL2 Toolset. http://www.mcrl2.org/. [85]

[vdBHKO02] M.G.J. van den Brand, J. Heering, P. Klint, and P.A. Olivier. Compiling language definitions: the ASF+SDF compiler. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 24(4):334–368, 2002. [18, 19]

[vdBvDH+01] M.G.J. van den Brand, A. van Deursen, J. Heering, H.A. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P.A. Olivier, J. Scheerder, J.J. Vinju, E. Visser, and J. Visser. The ASF+SDF Meta-environment: A component-based language development environment. In R. Wilhelm, editor, *Compiler Construction: 10th International Conference, CC 2001*, volume 2027 of *Lecture Notes in Computer Science*, pages 365–370, 2001. [2, 19, 85]

[vdP01] J.C. van de Pol. Just-in-time: On strategy annotations. In Bernhard Gramlich and Salvador Lucas, editors, *WRS 2001, 1st International Workshop on Reduction Strategies in Rewriting and Programming*, volume 57 of *Electronic Notes in Theoretical Computer Science*, pages 41–63, 2001. [2, 4, 10, 12, 13, 89]

[vdP02] J.C. van de Pol. JITty: a rewriter with strategy annotations. In S. Tison, editor, *Rewriting Techniques and Applications : 13th International Conference, RTA 2002, Copenhagen, Denmark, July 22-24, 2002. Proceedings*, volume 2378 of *Lecture Notes in Computer Science*, pages 367–370, 2002. [11, 13, 51]

[Vis04] Eelco Visser. Program transformation with Stratego/XT: Rules, strategies, tools, and systems in StrategoXT-0.9. In C. Lengauer, D. Batory, C. Consel, and M. Odersky, editors, *Domain-Specific Program Generation*, volume 3016 of *Lecture Notes in Computer Science*, pages 216–238, 2004. [2]

[Vis05] E. Visser. A survey of strategies in rule-based program transformation systems. *Journal of Symbolic Computation*, 40:831–873, 2005. [2]

[Vit96] M. Vittek. A compiler for nondeterministic term rewriting systems. In *Rewriting Techniques and Applications, 7th International Conference, RTA-96, New Brunswick, NJ, USA, July 27-30, 1996, Proceedings*, volume 1103 of *Lecture Notes in Computer Science*, pages 154–167, 1996. [18, 19]

[vW07] M.J. van Weerdenburg. An account of implementing applicative term rewriting. In S. Antoy, editor, *Proceedings of the Sixth International Workshop on Reduction Strategies in Rewriting and Programming (WRS 2006)*, volume 174/10 of *Electronic Notes in Theoretical Computer Science*, pages 139–155, 2007. [2, 3, 4, 10, 12, 18, 79, 81, 82, 85, 86, 87]

Index

Summary

This thesis considers three aspects of the (efficient) implementation of term rewrite systems. For efficient matching of terms against rules we introduce a formal notion of match trees. These match trees can be used to simultaneously match a term against multiple rewrite rules.

The second aspect is that of temporary-term construction. After each application of a rewrite rule, a new (often temporary) term is constructed. In order to make rewriting as efficient as possible, it is shown how to annotate these temporary terms such that the information about which (sub)terms are already rewritten to normal form is preserved. This allows strategies to be written such that these subterms, known to be in normal form, will not be considered a second time. To avoid needless construction of temporary terms, it is also shown how to determine what subterms will be rewritten later on for sure, allowing for immediate rewriting of such terms.

Finally, the notion of strategy trees is introduced. These strategy trees allow for a flexible specification of lazy rewrite strategies. Conditions are given for strategy trees that guarantee that rewriting a term results in a normal form of that term. Also, a method is given to automatically generate strategy trees that satisfy these conditions.

Curriculum Vitae

In 1981, on the 8th of August, Muck Joost van Weerdenburg was born in 's-Hertogenbosch, the Netherlands. He graduated at the Stedelijk Gymnasium 's-Hertogenbosch in 1999, after which he studied Technische Informatica (Computer Science) at the Eindhoven University of Technology (TU/e). There he obtained his Master's degree in 2004 (cum laude) with his thesis on the GenSpect Process Algebra, which formed the basis of the process part of the language mCRL2. The following four years he stayed at the TU/e as a Ph.D. student. During this time he made contributions to the development and implementation of the mCRL2 language and toolset and researched various subjects such as (timed) process algebra, aspects of structural operational semantics and the implementation of rewrite systems.

Titles in the IPA Dissertation Series since 2002

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects.* Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity.* Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing.* Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing.* Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining.* Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL.* Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand.* Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations.* Faculty of Natural Sciences,

Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks.* Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing.* Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions.* Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components.* Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components.* Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method.* Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving.* Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems.* Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation.* Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments.* Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications.* Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications.* Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems.* Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms*

and Proofs. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality.* Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications.* Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting.* Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs.* Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations.* Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data.* Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space.* Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices.* Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization.* Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML.* Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models.* Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications.* Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols.* Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition.* Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains*. Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use*. Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems*. Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML*. Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery*. Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J. Mathijssen. *Logical Calculi for Reasoning with Binding*. Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks*. Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data*. Faculty of Mathematics and Computer Science, TU/e. 2007-19

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications.* Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications.* Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation.* Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems.* Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification.* Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development.* Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation.* Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean.* Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems.* Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques.* Faculty of Mathematics and Computer Science, TU/e. 2009-06