Confluence and Normalisation for Higher-Order Rewriting

© 1996 Femke van Raamsdonk

VRIJE UNIVERSITEIT

Confluence and Normalisation for Higher-Order Rewriting

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit te Amsterdam,
op gezag van de rector magnificus
prof.dr E. Boeker,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der wiskunde en informatica
op maandag 13 mei 1996 te 13.45 uur
in het hoofdgebouw van de universiteit, De Boelelaan 1105

door

FEMKE VAN RAAMSDONK

geboren te Amsterdam

Promotor:	prof.dr J.W. Klop
Referent:	prof.dr P.-L. Curien

Preface

Having finished my thesis, I sit down with the intention to thank the ones who made the last four years of research into an, on the whole, enjoyable experience.

First of all, I have learned a lot from my supervisor Jan Willem Klop. In the beginning the basic questions he asked surprised me, later on I understood that it is necessary to ask these questions again and again in order to come to a deep understanding. His questions and intuitions have always been a source of inspiration to me. Moreover, he was always ready to listen to me, whether or not what I had to say was of a technical nature. For all this I wish to thank him.

I discovered the joy of working together in my collaboration with Vincent van Oostrom. His influence on the work reported in this thesis is enormous. Further, during the writing of this thesis he has been a great help to me, providing constructive remarks, appropriate criticism and comforting words. Shortly, without Vincent this thesis as it is now would not exist.

I am sincerely grateful to Pierre-Louis Curien for his willingness to referee this thesis. He has read my work in detail and has provided many very useful remarks. In fact his influence reaches even further. It was during the writing of my Master's Thesis under supervision of Pierre-Louis Curien and Roel de Vrijer that I discovered how enjoyable doing research can be.

I also wish to thank Roel de Vrijer. With great pleasure I think back to the period he was supervising my Master's Thesis, and also to our conversations concerning science, music and so many other things.

The last years Aart Middeldorp has always kept an eye on my work. I have learned a lot from him, and I am very grateful for his sensible advice, his interest in my work and his friendship.

Further I have very much enjoyed working together with Paula Severi and Zurab Khasidashvili. I shared an office with Fer-Jan de Vries, who always encouraged me, and later on with Inge Bethke. I wish to thank them all for the good times, and for their help in the preparation of this thesis.

I wish to thank Pierre-Louis Curien, Tobias Nipkow, Jaco de Bakker, Henk Barendregt, Rob Nederpelt, Roel de Vrijer, and Hans Zantema for their willingness to be a member of the promotion committee and for their interest in my work.

It has been a pleasure to work at CWI. In particular, I thank Krzysztof Apt for

his trust in me, and for his patience during the long time I had 'almost finished' writing my thesis.

My friends Astrid, Conchita, Daniëlle, Hans and Mechelien are always ready to discuss Everything, whether or not it concerns work. I wish to thank my parents and my brother Rutger for their confidence in me, but most importantly for always being there. Finally, I am very lucky to share my life with Pieter. His unconditional love and everything we share together make my life into a very happy one.

I gratefully acknowledge the financial support of NWO/SION in the form of grant 612-316-606, 'Extensions of Orthogonal Rewrite Systems: Syntactic Properties'.

The diagrams in this thesis are designed using the package Xy-pic of Kristoffer H. Rose.

Contents

Introduction

Rewriting. A basic concept in computing science is that of a transformation. A computation consists of transforming step by step an expression into another one. Often the aim of a computation is to obtain eventually an expression that is in some way simpler than the initial one. An example of a computation is

$$(3+5) \times 7 \rightarrow 8 \times 7 \rightarrow 56.$$

The expression $(3 + 5) \times 7$ is transformed via the simpler one 8×7 into the elementary one 56. There might be different ways to transform an expression. Another computation starting with the expression $(3+5) \times 7$ is

$$(3+5) \times 7 \rightarrow (5+3) \times 7 \rightarrow (3+5) \times 7 \rightarrow \ldots,$$

which, unlike the previous one, does not end.

The subject of this thesis is the theory of transformations. We will call a transformation step a *rewrite step*, and a sequence of consecutive rewrite steps a *rewrite sequence*.

Often a result of a computation is modelled as an expression that cannot be rewritten any further. Such an expression is called a *normal form*. In a normal form, nothing remains to be computed, so there is no way to gain more information. A natural question is whether a computation eventually yields a result, or, whether a rewrite sequence always ends in finitely many rewrite steps in a normal form. The example illustrates that this may depend on the way of rewriting. If an expression can be rewritten in such a way that eventually a normal form is reached, then it is said to be *weakly normalising*. If there is no infinite rewrite sequence starting at an expression a, then a is said to be *strongly normalising*. Strong normalisation is a stronger property than weak normalisation, as illustrated by the example.

Another natural question is whether the result of a computation, if it exists, is unique. We say that an expression has the *property of unique normal forms* if any two normal forms that can be reached from it, are equal. Uniqueness of normal forms is guaranteed by an important property in rewriting called *confluence*. An expression a is said to be confluent if any two finite rewrite sequences issuing from a can be continued in such a way that they end in the same expression. Writing

$a \twoheadrightarrow a'$ for a rewrite sequence from a to a' consisting of zero, one or more rewrite steps, this amounts to the following: for every b and b' such that $a \twoheadrightarrow b$ and $a \twoheadrightarrow b'$ there is a c such that $b \twoheadrightarrow c$ and $b' \twoheadrightarrow c$. In a picture:

$$\begin{tikzcd} & a \arrow[dl, two heads] \arrow[dr, two heads] & \\ b \arrow[dr, two heads] & & b' \arrow[dl, two heads] \\ & c & \end{tikzcd}$$

It is easy to see that a confluent expression has the property of unique normal forms: suppose a is confluent and can be rewritten to normal forms b and b'. Then there is an expression that can be reached both from b and from b'. Since b nor b' can perform any step, it must be the case that b and b' are equal.

We will now discuss how rewrite steps arise quite naturally, namely when computing with equations. First we stress the important point of the difference between *representation* on the one hand and *meaning* on the other hand. This is illustrated by the example on the previous page as follows. All expressions in the rewrite sequence $(3 + 5) \times 7$ have the same meaning, namely fifty-six. In each rewrite step, a representation of fifty-six is transformed into another, simpler, one.

For many different purposes theories are considered in which equality is defined by a set of equations. Two expressions are considered to be equal, or to have the same meaning in the sense as above, if this is a consequence of equational reasoning, which consists of assigning arbitrary expressions to variables and replacing equals by equals. Such a theory is called an equational theory. As an example we consider the equational theory for natural numbers with addition, defined by the following axioms:

$$\begin{aligned} 0 + x &= x, \\ S(x) + y &= S(x + y). \end{aligned}$$

Here 0 models the natural number zero, and S the successor operation of adding one. Further, x and y are variables standing for an arbitrary expression in the theory. A first natural question in an equational theory is whether the theory does not identify too much. This question motivates to consider a notion of consistency, expressing that there are two expressions not containing variables, that are not equal in the theory. Further, it makes sense to ask whether two expressions are equal in the theory. In the equational theory of the example, we have for instance $S(0) + S(0) = S(0 + S(0)) = S(S(0))$.

By orienting the equations they are turned into *rewrite rules*. Orienting the equations of the example from left to right, we obtain the following two rewrite rules:

$$\begin{array}{rcl} 0 + x & \to & x, \\ S(x) + y & \to & S(x+y). \end{array}$$

Rewrite rules induce a *rewrite relation* on the set of expressions as follows. If a part of an expression is an instance of the left-hand side of a rewrite rule, then this part is replaced by the corresponding instance of the right-hand side of the rewrite rule. An instance is obtained by assigning expressions to variables. For example, the expression S(0) + S(0) is of the form $S(x) + y$, with $x = 0$ and $y = S(0)$, and hence it can be rewritten to the expression $S(0 + S(0))$. The set of expressions together with the rewrite rules form a *rewriting system.* A rewriting system forms an implementation of an equational theory if the equivalence closure of its rewrite relation coincides with the equality of the theory. In that case, the rewrite system can be used to answer questions concerning the equational theory. For instance, if the rewriting system is confluent and strongly normalising, which it is if all the expressions are confluent and strongly normalising, then it yields a procedure to decide whether two expressions are equal in the theory: just rewrite them to normal form and check whether the two normal forms are syntactically equal. If the rewriting system is confluent, then the equational theory is consistent in a strong sense, since every two normal forms not containing variables are equal in the theory only if they are syntactically equal. Equality of the expressions $S(0) + S(0)$ and $S(S(0))$ follows since $S(0) + S(0)$ can be rewritten to $S(S(0))$: $S(0) + S(0) \to S(0 + S(0)) \to S(S(0))$.

The above discussion illustrates that confluence and both weak and strong normalisation are essential properties in rewriting. They constitute two main themes of this thesis.

History. We will now briefly discuss the history of rewriting which will lead us automatically to the subject of higher-order rewriting.

Combinatory Logic and *λ-calculus* have been studied since the 1920s in order to provide a general theory of functions. A formal system forming the basis of what is known nowadays as Combinatory Logic was introduced in 1924 by Schönfinkel in [Sch24]. His aim was to eliminate bound variables from logic, that appear in quantifiers like 'for all' and 'there exists'. Schönfinkel shows that every function can be written as an expression built from application and the combinators S and K satisfying the axioms $Kxy = x$ and $Sxyz = xz(yz)$. Application is written as juxtaposition with association to the left. Curry rediscovered Combinatory Logic in 1929 (see [CFC58]) and showed a weak form of consistency for a system extending the one of Schönfinkel.

Around 1932, Church developed λ-calculus (see [Chu41]), in which there is besides a binary operator for application, like in Combinatory Logic, also an operator λ used to abstract a function value over a variable. For instance, the function $f : x \mapsto x^2$ is written in λ-notation as $\lambda x.x^2$. The main axiom for λ-calculus is $(\lambda x.M)N =_\beta M[x := N]$ where $[x := N]$ denotes the simultaneous substitution of N for all free occurrences of x in M. Again, we write application as juxtaposition. The original system was shown to be inconsistent, therefore Church introduced a subtheory called the λI-calculus. Consistency of the λI-calculus was proved by by Church and Rosser in [CR36], by showing that the rewriting system obtained by orienting the axiom for β from left to right, is confluent. Kleene has shown that every recursive function is definable in λ-calculus. This supported what is now known as 'Church's Thesis': every effective computable function from natural numbers to natural numbers is λ-definable.

The history of *abstract rewriting systems* goes back to Newman. He presents in [New42] a classical result in rewriting, nowadays called Newman's Lemma, stating that confluence is guaranteed by strong normalisation and a property called *local confluence*. A rewriting system is said to be locally confluent if whenever $a \to b$ and $a \to b'$, then there exists a c such that $b \twoheadrightarrow c$ and $b' \twoheadrightarrow c$. Newman's Lemma is an important method to prove confluence of a rewriting system. It is obtained without specifying the structure of the expressions that are rewritten. We will call such a rewriting system an abstract rewriting system. The advantage of abstract rewriting systems is their generality: results obtained for abstract rewriting systems hold for any other rewriting system.

In a *term rewriting system*, the expressions that are rewritten have a term structure. The rewrite relation on terms is induced by rewrite rules that act as schemes, as in the case of Combinatory Logic and λ-calculus. For the moment we restrict attention to first-order term rewriting, where terms are built over a first-order alphabet. An example of a first-order term rewriting system is the rewriting system obtained by orienting the equations specifying addition for natural numbers given above. Another example is the rewriting system obtained by orienting the axioms for Combinatory Logic as $Kxy \to x$ and $Sxyz \to xz(yz)$. Often moreover the rewrite rule $Ix \to x$ is added. As discussed above, a term can be rewritten if a rewrite rule $\mathtt{l} \to \mathtt{r}$ can be applied to a subterm. More concretely, the rewrite relation generated by a rewrite rule $\mathtt{l} \to \mathtt{r}$ is defined by $C[\mathtt{l}^\theta] \to C[\mathtt{r}^\theta]$ for an assignment θ mapping variables to terms, and a context (which is a term with a hole) $C\Box$. An example of a rewrite sequence in Combinatory Logic is

$$SII(SII) \to I(SII)(I(SII)) \to (SII)(I(SII)) \to (SII)(SII),$$

which shows that Combinatory Logic is not strongly normalising. Term rewriting systems appear as subtree replacement systems in an article by Rosen [Ros73]. He formulates conditions that imply confluence of a term rewriting system. In general, the properties confluence and strong normalisation are undecidable for first-order term rewriting. Surveys on first-order term rewriting are [HO80], [DJ90] and [Klo92].

Higher-Order Rewriting. Many specifications, like for instance natural numbers with addition as considered above, can be defined in λ-calculus. Nevertheless, it is often more convenient to add them explicitly. Moreover, not every combination of λ-calculus with first-order rewrite rules is definable in λ-calculus. For instance, Barendregt has shown in [Bar74] that λ-calculus plus surjective pairing cannot be defined in λ-calculus. This motivates the study of rewriting systems that contain both λ-calculus and term rewriting. It is natural not to restrict the term rewriting part to the first-order case, but to incorporate as well higher-order rewrite rules, containing bound variables. First, many equations occurring in mathematics and computing science contain bound variables. A well-known example is the law concerning the derivative of a sum of two functions: $\mathsf{dif}(f+g)(x) = \mathsf{dif}(f)(x) + \mathsf{dif}(g)(x)$. If we express a function f using λ-notation as $\lambda x.fx$, then this law takes the following form:

$$\mathsf{dif}(\lambda x.fx + \lambda x.gx)(y) = \mathsf{dif}(\lambda x.fx)(y) + \mathsf{dif}(\lambda x.gx)(y).$$

We can answer questions concerning equational theories defined by a set of equalities that possibly contain bound variables by considering the corresponding rewriting system, as is done for the first-order case. Such a rewriting system may contain rewrite rules that contain bound variables. Second, functional programming languages may contain specifications of higher-order functions, like

$$\begin{array}{rcl} \mathsf{map}\, f\, \mathsf{nil} & = & \mathsf{nil}, \\ \mathsf{map}\, f\, (\mathsf{cons}\, x\, l) & = & \mathsf{cons}\,(f\,x)(\mathsf{map}\, f\, l) \end{array}$$

specifying an operation map that given a function f applies this function to all elements of a list. Also in this case, the specification can be studied by considering the rewriting system obtained by orienting the equations. Moreover, it is often convenient to consider $\lambda x.fx$ instead of f only. In that case the rewriting system also contains bound variables.

This motivates the study of rewriting systems containing bound variables. Such rewriting systems are called *higher-order rewriting systems*. A first important step in the theory of higher-order rewriting was made by Klop, who introduced in [Klo80] the class of Combinatory Reduction Systems, in the sequel abbreviated as CRSs. The class of CRSs contains both λ-calculus and first-order term rewriting. CRSs generalise $\lambda(a)$-reductions as introduced by Hindley in [Hin78], and contraction schemes as introduced by Aczel in [Acz78]. The class of $\lambda(a)$-reductions is an extension of λ-calculus with operators (called atoms) and axiom schemes for them. The class of contraction schemes contains λ-calculus and a subclass of first-order term rewriting systems.

In subsequent years, more formalisms of higher-order rewriting were introduced. Nipkow defines in [Nip91] the class of Higher-Order Rewrite Systems, in the sequel abbreviated as HRSs, with the aim to investigate the meta-theory of systems like λProlog and Isabelle. HRSs have simply typed λ-calculus as a meta-language. In this respect they are different from CRSs, that are equipped with an untyped meta-language. In [Kha90], Khasidashvili introduces the class of Expression Reduction Systems, in the sequel abbreviated as ERSs. ERSs are similar to CRSs, but have a different syntax. Further, Wolfram considers in [Wol91] Higher-Order Term Rewriting Systems, that are similar to HRSs in the sense that they also use simply typed λ-calculus as a meta-language, but more general. Finally we mention the class of Interaction Systems, in the sequel abbreviated as ISs, introduced by Asperti and Laneve in [Lan93] and [AL94]. ISs form a subclass of CRSs, in fact, they can roughly be thought of as CRSs satisfying a constructor discipline. They were introduced to study the theory of optimality as defined by Lévy in [Lév78] in a setting more general than the one of λ-calculus. These systems can all be seen as a uniform framework for λ-calculus and term rewriting.

Another approach is taken by studying λ-calculus combined with term rewriting. This is done for instance by Breazu-Tannen and Gallier, who show in [BTG90] and [BTG94] that combining polymorphic λ-calculus with a confluent term rewriting system yields a confluent rewriting system, and that a combination of polymorphic λ-calculus with a strongly normalising term rewriting system is again strongly normalising. Jouannaud and Okada present in [JO91] systems combining type theory and higher-order rewriting. They provide conditions on the higher-order rewrite rules that imply strong normalisation. Combinations of λ-calculus and higher-order rewriting are studied for various typing disciplines. In this thesis we will be primarily concerned with the uniform approach to higher-order rewriting.

The main reason to study CRSs, ERSs and HRSs is to extend the rewriting theory as developed for first-order term rewriting and for λ-calculus to the higher-order case. It is hence of obvious interest to understand the relationship between these three classes. The main example of a result concerning first-order rewriting that has been lifted to the higher-order case, is that a condition on the rewrite rules, called *orthogonality*, is a sufficient condition to obtain confluence. This has been shown for CRSs, ERSs and HRSs. The question arises whether this result needs to be proved for CRSs, ERSs and HRSs separately.

As a first step in the investigation of the relationships between different formats of higher-order rewriting, together with Vincent van Oostrom the relationship between CRSs and HRSs was studied. This study, reported in [OR94a, OR93], revealed that CRSs and HRSs are roughly speaking equivalent as far as rewriting is concerned, and that the main difference between these two classes lies in the meta-language used to instantiate left- and right-hand sides of rewrite rules to terms. In CRSs, the meta-language is in fact untyped λ-calculus with developments, whereas in HRSs it is simply typed λ-calculus with β-reduction and restricted η-expansion, and rewrite sequences to $\beta\overline{\eta}$-normal form. We presented a translation from CRSs into HRSs and vice versa, such that a rewrite step in a CRS is mapped into a rewrite step in its associated HRS, and a rewrite step in a HRS is mapped into a rewrite step in its associated CRS, possibly followed by a finite β-reduction sequence. For the reader familiar with HRSs and CRSs, we illustrate this by an example. Consider the HRS rewrite rule $f(\lambda x.z(\lambda y.xy)) \to z(\lambda x.x)$. It induces, with an assignment $z \mapsto \lambda x.xa$, the rewrite step

$$f(\lambda x.xa) \to a.$$

The corresponding CRS rewrite rule is $f([x]Z(x)) \to Z([x]x)$. With the assignment $Z \mapsto \boldsymbol{\lambda} x.\mathsf{app}(x,a)$, it induces the rewrite step

$$f([x]\mathsf{app}(x,a)) \to \mathsf{app}([x]x,a).$$

In order to simulate the rewrite step in the HRS, an explicit β-reduction step $\mathsf{app}([x]x,a) \to a$, using a β-reduction rule that is added to the CRS, must be performed.

The fact that formats of higher-order rewriting may differ in the meta-language employed suggests that in order to capture the concept of higher-order rewriting one should parametrise over the meta-language. This is not necessary to obtain a framework containing both HRSs and CRSs, since as explained above, CRSs can be coded as HRSs, but one might wish to consider other meta-languages as well.

Also in collaboration with Vincent van Oostrom, we introduced *higher-order rewriting systems*. A higher-order rewriting system is specified by a meta-language, an alphabet and a set of rewrite rules. Since the meta-language prescribes the structure of the expressions that are rewritten, the alphabet and the rewrite rules depend on the meta-language. Because the meta-language does not only specify the way expressions are built, but is also used to calculate substitutions of expressions for variables, we call it a *substitution calculus*. CRSs are obtained by specifying the substitution calculus to be untyped λ-calculus with developments, and HRSs are obtained by specifying the substitution calculus to be simply typed λ-calculus with β-reduction and restricted η-expansion, denoted by $\lambda_{\overline{\eta}}^{\to}$. The prime example of a substitution calculus is $\lambda_{\overline{\eta}}^{\to}$. The substitution calculus permits to define the rewrite relation in a different way, using instead of contexts and assignments as usual only contexts. Therefore we need to abstract left- and right-hand sides of rewrite rules over their free variables. As an example, we consider the rewrite rule specifying the multiplication of zero with some natural number. In the usual format of term rewriting, this rule is written as $\mathsf{M}(0,x) \to x$. In the format of higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\to}$ as substitution calculus, this rule is represented as

$$x.\mathsf{M}0x \to x.0.$$

Now the rewrite step $\mathsf{M02} \to \mathsf{0}$ is obtained as follows:

$$\begin{array}{rcl} \mathsf{M02} & {}_{\beta\overline{\eta}}\!\leftarrow & (x.\mathsf{M0}x)\mathsf{2} \\ & \to & (x.\mathsf{0})\mathsf{2} \\ & \to_{\beta\overline{\eta}} & \mathsf{0}. \end{array}$$

So a rewrite step is defined as an expansion in the substitution calculus, followed by a replacement of the left-hand side of a rewrite rule by its right-hand side, followed by a reduction in the substitution calculus.

A first presentation of higher-order rewriting systems is given in [Oos94], and another, slightly different one in [OR94b]. Since then, the presentation of the syntax of higher-order rewriting systems has been subject to some changes although the essential underlying concepts are identical. In Chapter 4 of this thesis we present higher-order rewriting systems in a way slightly differing from earlier presentations. First we give the general definition, and next we present the higher-order rewriting systems with $\lambda^{\to}_{\overline{\eta}}$ as substitution calculus. For the latter class, we present results concerning confluence and normalisation in Chapter 5 and Chapter 6. In Chapter 4, we also present rewriting higher-order rewriting systems with proof nets as substitution calculus. Moreover, we show that HRSs, CRSs, ERSs and ISs fit in the framework of higher-order rewriting systems.

Terminology. The historical development of the area of higher-order rewriting has resulted in a somewhat confusing terminology with respect to the names of the various formats of higher-order (term) rewriting. To cope with this situation, we will henceforth stick to the following names:

- CRSs, HRSs, ERSs and ISs are always written as abbreviations,
- higher-order rewriting systems introduced in [Oos94, OR94b] and defined in Chapter 4 of this thesis are always written in this unabbreviated form in lower-case,
- the Higher-Order Term Rewriting Systems as defined by Wolfram in [Wol91] are written unabbreviated with capitals; they will only be considered in Section 4.4 of Chapter 4 concerned with HRSs.

Overview. This thesis is organised as follows.

In Chapter 1, we are concerned with abstract rewriting. First we focus on abstract rewriting systems. In this setting we introduce the concepts and terminology, and we recall the classical results in abstract rewriting that will be used later on in the thesis. Next we introduce functional rewriting systems. In this setting we can formalise the important notions of descendants and developments.

The classical proof of confluence using developments is given for functional rewriting systems. Moreover we give the definition of a rewrite strategy as will be used in Chapter 6. All rewriting systems that appear in the remainder of the thesis have an underlying functional rewriting system. The chapter is concluded by formalising the notion of typing in an abstract setting. We introduce abstract rewriting systems with typing, a concept that will be used in Chapter 4. We classify all rewriting systems considered in this chapter under abstract rewriting, since in all cases the structure of the expressions that are rewritten is not specified.

In Chapter 2, λ-calculus with β-reduction is considered. We recall the main definitions that will be used also in Chapter 4. Then a characterisation of the strongly normalising λ-terms is given. This characterisation, that has the form of an inductively defined set of λ-terms denoted by $\mathcal{SN}$, permits to give new and simple proofs of important results concerning normalisation in λ-calculus. Two new proofs of finiteness of developments are given. The first proof makes use of a set $\mathcal{FD}$ that contains underlined terms that admit only finite $\underline{\beta}$-rewrite sequences. The definition of $\mathcal{FD}$ is in the same spirit as the definition of $\overline{\mathcal{SN}}$. We show that this set contains all underlined λ-terms, which yields finiteness of developments. For the second proof, we define a morphism that maps a development to a rewrite sequence in the set $\mathcal{SN}$, such that a step in the original sequence corresponds to a step in its image. This also yields finiteness of developments. Next we introduce superdevelopments which form a generalisation of developments. Whereas in a development only redexes are contracted that are residuals of redexes that are present in the initial term, in a superdevelopment also some redexes that are created by rewriting may be contracted. We give two proofs of finiteness of superdevelopments, in the same way as for developments. Finally we consider simply typed λ-calculus. Using again the characterisation of strongly normalising terms, we give a proof of strong normalisation of simply typed λ-calculus that is very elementary. It cannot be extended to system F, but it can be extended to enriched λ-calculi like Gödel's T.

In Chapter 3 the syntax of multiplicative-exponential proof nets with cut-elimination is presented. We present a proof of strong normalisation of proof nets with cut-elimination. The rewriting system consisting of proof nets with cut-elimination will be used in Chapter 4 as a substitution calculus for higher-order rewriting systems.

Chapter 4 is the pivot of the thesis in the sense that we use results presented in Chapter 2 and Chapter 3, and we present the framework in which the results of Chapter 5 and Chapter 6 are proved. This chapter is concerned with higher-order rewriting. We first give the definition of a higher-order rewriting system in its most general form, without specifying the substitution calculus, but only giving the requirements a substitution calculus should satisfy. We make use of the notion of abstract rewriting with typing: a substitution calculus is supposed to be an abstract rewriting system with typing. The typing relation guarantees well-formedness if the left-hand side of a rewrite rule is replaced by its right-hand side in the definition of a rewrite step in a higher-order rewriting system, as explained above. Next we fix attention to higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus and with proof nets with cut-elimination as substitution calculus. In both cases we show that the requirements on a substitution calculus are satisfied, using among other things the results on normalisation presented in Chapter 2 and Chapter 3. The presentation of higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus is the most elaborate one, since it is in this setting that we prove the results presented in Chapter 5 and Chapter 6. Higher-order rewriting systems with proof nets as substitution calculus are much less traditional than the ones with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus. We present the main ideas, but leave many important themes unexplored. In the remainder of the chapter we consider classes of higher-order rewriting systems that were defined earlier: HRSs, CRSs, ERSs and ISs. We show that they all can be presented as higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus. In the case of HRSs this is no surprise, since higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus are basically HRSs without the restriction that rules should be of base type.

In Chapter 5 we prove that weakly orthogonal higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus are confluent. A weakly orthogonal rewriting system is left-linear and has only trivial critical pairs. The proof makes use of the notion of parallel reduction, so it is a proof à la Tait and Martin-Löf. A short discussion concerning both this proof method and the proof method using developments is included. The result presented in this chapter is also proved by van Oostrom in [Oos94], but in a different way namely using developments. Short versions of both proofs can also be found in [OR94b]. The result that all weakly orthogonal higher-order rewriting systems are confluent extends earlier results in which either a restriction to first-order rewriting or a restriction to orthogonal instead of weakly orthogonal systems was made. We also give a very short proof of confluence for the class of orthogonal higher-order rewriting systems using parallel reductions.

Chapter 6 is concerned with normalisation in higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus that are almost orthogonal and fully extended. An almost orthogonal rewriting system is left-linear and has only critical pairs that overlap at the root, and that are trivial. The condition fully extended is concerned with bound variables. We show that in an almost orthogonal and fully extended higher-order rewriting system outermost-fair rewriting is normalising. A rewrite sequence is said to be outermost-fair if every outermost-redex is eventually eliminated. The proof does not really depend on the fact that the substitution calculus is $\lambda_{\overline{\eta}}^{\rightarrow}$, but it does make use of the fact that the structures that are rewritten are terms. Already the formalisation of the notion of outermost redex is not completely straightforward if we take for instance proof nets with cut-elimination as substitution calculus. The proof of the main result makes use of a certain way to perform a development of a set of redex occurrences in phases. This construction permits to infer that the projection of an outermost-fair rewrite sequence over some rewrite step is also outermost-fair, and moreover to show that the original rewrite sequence is finite if the projection is finite. These results are used to prove that outermost-fair rewriting is normalising. As a consequence, the parallel-outermost rewrite strategy is normalising for almost orthogonal and fully extended higher-order rewriting systems.

Notation

The set of the natural numbers $0, 1, 2, \ldots$ is denoted by $\mathbb{N}$. We use $m, n, p, q, \ldots$ to range over natural numbers. The first infinite ordinal is denoted by ω. An element of $\mathbb{N} \cup \{\omega\}$ is denoted by α.

If X is a set of natural numbers then we denote by $\max\{X\}$ the maximum of X with respect to the usual ordering on $\mathbb{N}$.

Let X and Y be sets. We denote by $X \cup Y$ the union of X and Y, by $X \cap Y$ the intersection of X and Y, by $X \setminus Y$ the set X minus Y, and by $X \times Y$ the product of X and Y. We denote by $\mathcal{P}(X)$ the power-set of X and by $\#X$ the cardinality of X. The set of partial functions from X to Y is denoted by $[X \rightharpoonup Y]$. Membership is denoted by $\in$. If P is a property on X, then we denote by $\{x \in X \mid P(x)\}$ the subset of X consisting of all elements that satisfy the property P.

The set of finite sequences over a set X is denoted by X^*. The empty sequence is denoted by ϵ. If elements of X are denoted by $x, x', \ldots$, then elements of X^* are denoted by $x_1 \ldots x_m, \ldots$ or by $\vec{x}$. We denote $X^* \setminus \{\epsilon\}$ by X^+.
We use $[\_]$ to denote a multiset.

We abbreviate $x_1 \in X, \ldots, x_m \in X$ by $x_1, \ldots, x_m \in X$.

Chapter 1

Abstract Rewriting

This chapter is concerned with rewriting systems that are all abstract in the sense that the structure of the objects that are rewritten is not specified. Various such rewriting systems are considered in the literature, starting with Newman in [New42]. In this chapter we will introduce most of the terminology and concepts that will be used in the remainder of this thesis. To that end we discuss three classes of rewriting systems that we classify under abstract rewriting.

In Section 1.1 abstract rewriting systems are considered. An abstract rewriting system is just a set equipped with a binary relation, called the rewrite relation. This forms a very abstract model for computations, and is as such also used in different contexts using a different terminology. For instance, a set equipped with a binary relation is introduced by Plotkin in [Plo81] as a transition system in the study of semantics of programming languages. In the framework of abstract rewriting systems we give the definitions of important concepts in rewriting such as confluence and strong normalisation, which play a key rôle in this thesis.

In Section 1.2 functional rewriting systems are introduced. In a functional rewriting system the rewrite relation is the union of a collection of indexed relations. This means that it is not only possible to express that an object is rewritten to another object, but also how this is done. Hence functional rewriting systems form a refinement of abstract rewriting systems. Functional rewriting systems differ from other indexed rewriting systems considered in the literature in that every indexed relation is a function, that is, every indexed relation is deterministic. In this setting the important notions of redex, redex occurrence, descendant and residual can be formalised.

Finally, in Section 1.3 we introduce abstract rewriting systems with typing. Abstract rewriting systems with typing are obtained by requiring the typing relation and the rewrite relation to be compatible in a certain natural way.

Both functional rewriting systems and abstract rewriting systems with typing are introduced in collaboration with Paula Severi.

1.1 Abstract Rewriting Systems

In this section abstract rewriting systems are defined. We introduce some concepts and terminology that will be used in the remainder of this thesis. This section does not contain original material.

Definition 1.1.1. An *abstract rewriting system* is a pair $(A, \to)$ consisting of a set A, whose elements are called *objects*, and a relation $\to \subseteq A \times A$ that is called a *rewrite relation*. Objects are denoted by $a, b, c, d, \ldots$.

An abstract rewriting system is different from an abstract reduction system as defined by Klop in [Klo92], because the latter is defined as a set equipped with a collection of labelled binary relations.

Definition 1.1.2. Let A be a set. The *identity relation* on A, denoted by id_A, is $\{(a, a) \mid a \in A\}$.

Definition 1.1.3. Let A be a set and let $\to_0$ and $\to_1$ be two binary relations on A. The *sequential composition* of $\to_0$ and $\to_1$, denoted by $\to_0; \to_1$, is the binary relation on A that is defined as follows: $(a, c) \in \to_0; \to_1$ if there exists $b \in A$ such that $(a, b) \in \to_0$ and $(b, c) \in \to_1$.

Definition 1.1.4. Let $(A, \to)$ be an abstract rewriting system.

1. The *inverse of* $\to$, denoted by $\leftarrow$, is defined by $(a, b) \in \leftarrow$ if and only if $(b, a) \in \to$.
2. The *m-fold composition of* $\to$, denoted by $\to^m$, is defined by induction on m as follows:
   (a) $\to^0 = \mathsf{id}_A$,
   (b) $\to^{m+1} = \to; \to^m$.

In the following definitions some important properties of relations on a set are listed.

Definition 1.1.5. Let $(A, \to)$ be an abstract rewriting system.

1. $\to$ is *transitive* if $\to; \to \subseteq \to$.
2. $\to$ is *reflexive* if $\mathsf{id}_A \subseteq \to$.
3. $\to$ is *irreflexive* if $\mathsf{id}_A \cap \to = \emptyset$.
4. $\to$ is *symmetric* if $\to = \leftarrow$.

5. $\rightarrow$ is *anti-symmetric* if $\rightarrow \cap \leftarrow \subseteq \mathsf{id}_A$.

6. $\rightarrow$ is an *equivalence relation* if it is transitive, reflexive and symmetric.

We identify properties of a binary relation $\rightarrow$ on a set A with properties of the abstract rewriting system $(A, \rightarrow)$.

A possible way to obtain a binary relation from a given one is by taking the smallest extension of it that satisfies a certain property on relations. This is illustrated in the following definition.

Definition 1.1.6. Let $(A, \rightarrow)$ be an abstract rewriting system.

1. The *reflexive closure* of $\rightarrow$, denoted by $\rightarrow^{=}$, is $\rightarrow \cup \mathsf{id}_A$.

2. The *transitive closure* of $\rightarrow$, denoted by $\rightarrow^{+}$, is $\cup_{m>0} \rightarrow^{m}$.

3. The *transitive-reflexive closure* of $\rightarrow$, denoted by $\rightarrow^{*}$, is $\cup_{m\geq 0} \rightarrow^{m}$.

4. The *symmetric closure* of $\rightarrow$, denoted by $\leftrightarrow$, is $\rightarrow \cup \leftarrow$.

5. The *equivalence closure* of $\rightarrow$, denoted by $\leftrightarrow^{*}$, is the transitive-reflexive closure of $\leftrightarrow$.

Remark 1.1.7. For every property P for which we define the P-closure of the relation $\rightarrow$ in the previous definition, it is the case that the P-closure is the smallest relation with respect to the subset ordering that includes $\rightarrow$ and has the property P. In general, the P-closure of a relation need not to exist.

The relation $\leftrightarrow^{*}$ is also called the *convertibility relation* and two objects a and b such that $a \leftrightarrow^{*} b$ are said to be *convertible.*

Notation 1.1.8. If a relation is denoted by a (decorated) arrow, then the derived relations and their notations are as given in Definition 1.1.6. For a relation denoted by $\rightarrow$, we will write $\twoheadrightarrow$ instead of $\rightarrow^{*}$.

We can obtain a new relation by combining two existing ones as follows.

Definition 1.1.9. The *union* of two abstract rewriting systems $(A, \rightarrow_0)$ and $(B, \rightarrow_1)$ is the abstract rewriting system $(A \cup B, \rightarrow_0 \cup \rightarrow_1)$.

The most interesting unions of abstract rewriting systems are those where the two combined systems can interact, which is possible if $A \cap B \neq \emptyset$.

Another possibility to obtain an abstract rewriting system is by restricting the set of objects of a given abstract rewriting system in a suitable way, namely such that it is closed under the rewrite relation.

Definition 1.1.10. Let $(A, \rightarrow_0)$ be an abstract rewriting system. An abstract rewriting system $(B, \rightarrow_1)$ is a *substructure* of $(A, \rightarrow_0)$ if the following conditions are satisfied:

1. $B \subseteq A$,
2. for all $b, b' \in B$ we have that $b \rightarrow_1 b'$ if and only if $b \rightarrow_0 b'$,
3. if $b \in B$ and $b \rightarrow_0 b'$, then $b' \in B$.

We now introduce some basic notions and results concerning abstract rewriting systems.

Definition 1.1.11. Let $(A, \rightarrow)$ be an abstract rewriting system. A *rewrite step* is a pair (a, b) of objects with $(a, b) \in \rightarrow$. We write $a \rightarrow b$ instead of $(A, \rightarrow)$.

Definition 1.1.12. Let $(A, \rightarrow)$ be an abstract rewriting system. For $\alpha \leq \omega$, a *rewrite sequence of length α starting in a* is defined as a triple (a, α, σ) satisfying the following:

1. $a \in A$,
2. $\sigma : \alpha \rightarrow A$ is a mapping such that
   (a) $\sigma(0) = a$,
   (b) $\sigma(m-1) \rightarrow \sigma(m)$ for $m \in \alpha \setminus \{0\}$.

Notation 1.1.13. Usually a rewrite sequence as in the previous definition is denoted by $\sigma : \sigma(0) \rightarrow \sigma(1) \rightarrow \sigma(2) \rightarrow \ldots$.

A rewrite sequence is infinite if it has length ω, and is finite otherwise. In this thesis we do not consider rewrite sequences of length longer than ω. For abstract rewriting, transfinite rewrite sequences are studied by Kennaway in [Ken92].

Definition 1.1.14. Let $\sigma : a \twoheadrightarrow b$ be a finite rewrite sequence, and let $\tau : b \rightarrow b' \rightarrow b'' \rightarrow \ldots$ be a possibly infinite rewrite sequence. The *sequential composition of σ and τ*, denoted by $\sigma; \tau$, is the rewrite sequence $\sigma; \tau : a \twoheadrightarrow b \rightarrow b' \rightarrow b'' \rightarrow \ldots$.

Sequential composition of rewrite sequences, when defined, is an associative operation.

Definition 1.1.15. Let $(A, \rightarrow)$ be an abstract rewriting system. Let $a \in A$. The *set of reducts of a*, denoted by $\mathsf{red}(a)$, is $\{b \in A \mid a \twoheadrightarrow b\}$.

As discussed in the Introduction, a possible way to model the result of a computation in an abstract rewriting system is by an object that cannot be rewritten. Such an object is said to be a normal form. A natural question is whether an object can be rewritten to a normal form. If this is the case, then the object is said to be weakly normalising. These concepts, and the stronger property of strong normalisation, are formalised in the following definition.

Definition 1.1.16. Let $(A, \rightarrow)$ be an abstract rewriting system. Let $a \in A$.

1. The object a is said to be a *normal form* or to be *in normal form* if there is no $b \in A$ such that $a \rightarrow b$.
2. We say that *a has normal form b* if $a \twoheadrightarrow b$ and b is a normal form. Then a is said to be *weakly normalising.*
3. We say that a is *strongly normalising* or *terminating* if there is no infinite rewrite sequence starting with a.

Every object that is strongly normalising is weakly normalising but a weakly normalising object is not necessarily strongly normalising.

A natural question is whether the result of a computation, if it exists, is unique. If results are modelled by normal forms, this notion is formalised as follows.

Definition 1.1.17. Let $(A, \rightarrow)$ be an abstract rewriting system. Let $a \in A$. If there exists at most one $b \in A$ such that a has normal form b, then a has the property of *unique normal forms.*

Notation 1.1.18. If $(A, \rightarrow)$ is an abstract rewriting system that is weakly normalising and has the property of unique normal forms, then we denote by $a\!\downarrow$ the normal form of a.

Sometimes a variation on the property of unique normal forms is considered, by requiring that if $a \leftrightarrow^{*} b$ and a and b are normal forms, then $a = b$. This is stronger than the property of unique normal forms, as is shown by the following example.

Example 1.1.19. Consider the following abstract rewriting system, due to de Vrijer.

$$\begin{tikzcd}
 & a \arrow[dl] \arrow[dr] & & a' \arrow[dl] \arrow[dr] & \\
b & & b' \arrow[loop below] & & b''
\end{tikzcd}$$

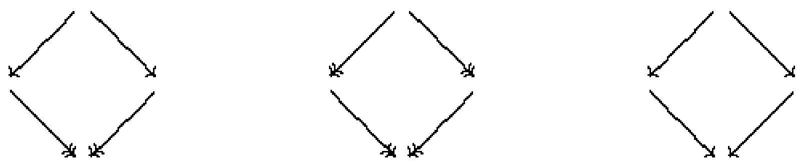
It has the property of unique normal forms as defined in Definition 1.1.17. However, we have for the normal forms b and b'' that $b \leftrightarrow^* b''$ but $b \neq b''$.

The property of unique normal forms is implied by a property called confluence. Confluence expresses that two finite rewrite sequences issuing from some object can always be pursued in such way that they end in the same object. Confluence and some variations of it are defined as follows.

Definition 1.1.20. Let $(A, \rightarrow)$ be an abstract rewriting system. Let $a \in A$.

1. The object a is said to be *locally confluent* if for all objects $b, c \in A$ such that $b \leftarrow a \rightarrow c$ there is an object $d \in A$ such that $b \twoheadrightarrow d \twoheadleftarrow c$.
2. The object a is said to be *confluent* if for all objects $b, c \in A$ such that $b \twoheadleftarrow a \twoheadrightarrow c$ there is an object $d \in A$ such that $b \twoheadrightarrow d \twoheadleftarrow c$.
3. The object a is said to have the *diamond property* if for all objects $b, c \in A$ such that $b \leftarrow a \rightarrow c$ there is an object $d \in A$ such that $b \rightarrow d \leftarrow c$.

The notions in the previous definition are depicted as follows:



Definition 1.1.21. Let P be a property defined in Definition 1.1.20. An abstract rewriting system $(A, \rightarrow)$ has property P if every object $a \in A$ has property P.

Definition 1.1.22. An abstract rewriting system $(A, \rightarrow)$ has the *Church-Rosser property* if for every $a \in A$ and $b \in A$ such that $a \leftrightarrow^* b$ there exists an object $c \in A$ such that $a \twoheadrightarrow c$ and $b \twoheadrightarrow c$.

In the following proposition classical results in rewriting are given that relate some of the concepts introduced above. For more results we refer to the survey paper [Klo92].

Proposition 1.1.23.

1. *$(A, \rightarrow)$ is Church-Rosser if and only if it is confluent.*
2. *If $(A, \rightarrow)$ has the diamond property then it is confluent.*
3. (Newman's Lemma) *If $(A, \rightarrow)$ is locally confluent and strongly normalising then it is confluent.*

4. *If $(A, \rightarrow)$ is confluent then it has the property of unique normal forms.*

Proof. A proof of 1 is given by Huet in [Hue80]. The first proof of 3 was given by Newman in [New42]; an easier proof can be found in [Hue80]. Finally, 2 and 4 follow easily from the definitions. □

We give the well-known example, due to Hindley, of an abstract rewriting system that is locally confluent but not confluent.

Example 1.1.24.

$$\begin{tikzcd} a & b \arrow[l] \arrow[r, bend left] & c \arrow[l, bend left] \arrow[r] & d \end{tikzcd}$$

The following lemmata permit to infer confluence of an abstract rewriting system from confluence of another abstract rewriting system with the same set of objects. The principle expressed in Lemma 1.1.25 is used in proofs of confluence using parallel reduction, a method which is due to Tait and Martin-Löf. In Chapter 5 we will prove confluence of weakly orthogonal higher-order rewriting systems using parallel reduction, and there we will use this lemma. Lemma 1.1.26 is mentioned just for curiosity.

Lemma 1.1.25. *Let $(A, \rightarrow_0)$ and $(A, \rightarrow_1)$ be abstract rewriting systems such that $\rightarrow_0 \subseteq \rightarrow_1 \subseteq \twoheadrightarrow_0$. Then confluence of $(A, \rightarrow_1)$ implies confluence of $(A, \rightarrow_0)$.*

Proof. If $a \twoheadrightarrow_0 b$ and $a \twoheadrightarrow_0 c$ then $a \twoheadrightarrow_1 b$ and $a \twoheadrightarrow_1 c$ since $\rightarrow_0 \subseteq \rightarrow_1$. Since $(A, \rightarrow_1)$ is confluent, there exists $d \in A$ such that $b \twoheadrightarrow_1 d$ and $c \twoheadrightarrow_1 d$. Because $\rightarrow_1 \subseteq \twoheadrightarrow_0$, this yields that $b \twoheadrightarrow_0 d$ and $c \twoheadrightarrow_0 d$, that is, $(A, \rightarrow_0)$ is confluent. □

In the following lemma we denote by $f(\rightarrow)$ the relation $\{(f(a), f(b)) \mid (a, b) \in \rightarrow\}$.

Lemma 1.1.26. *Let $(A, \rightarrow_0)$ and $(B, \rightarrow_1)$ be abstract rewriting systems. Suppose there are mappings $f : A \rightarrow B$ and $f' : B \rightarrow A$ such that*

1. $f(\rightarrow_0) \subseteq \twoheadrightarrow_1$,

2. $f'(\rightarrow_1) \subseteq \twoheadrightarrow_0$,

3. *for all $b \in B$ we have $b \twoheadrightarrow_1 f(f'(b))$.*

Then confluence of $(A, \rightarrow_0)$ implies confluence of $(B, \rightarrow_1)$

Proof. The proof is suggested by the following diagram:

$$\begin{tikzcd}
 & & f'(b_0) \arrow[dddll, two heads, "0"'] \arrow[dddrr, two heads, "0"] & & \\
 & & b_0 \arrow[u, dashed] \arrow[dl, two heads, "1"'] \arrow[dr, two heads, "1"] & & \\
 & b_1 \arrow[d, two heads, "1"] \arrow[dl, dashed] & & b_2 \arrow[d, two heads, "1"'] \arrow[dr, dashed] & \\
f'(b_1) \arrow[r, dashed] \arrow[ddrr, two heads, "0"'] & f(f'(b_1)) \arrow[dr, two heads, "1"] & & f(f'(b_2)) \arrow[dl, two heads, "1"'] & f'(b_2) \arrow[l, dashed] \arrow[ddll, two heads, "0"] \\
 & & f(b_3) & & \\
 & & b_3 \arrow[u, dashed] & &
\end{tikzcd}$$

□

Finally we mention two lemmata concerning strong normalisation. The first one is often used to infer strong normalisation of an abstract rewriting system from strong normalisation of another abstract rewriting system. This principle is used in Chapter 3 in which we give a proof of strong normalisation of proof nets with cut-elimination.

Lemma 1.1.27. *Let $(A, \rightarrow_0)$ and $(B, \rightarrow_1)$ be abstract rewriting systems. Let $f : A \rightarrow B$ be a mapping such that $f(\rightarrow_0) \subseteq \rightarrow_1^+$. Then strong normalisation of $(B, \rightarrow_1)$ implies strong normalisation of $(A, \rightarrow_0)$.*

Proof. Suppose that there is an infinite rewrite sequence $a_0 \rightarrow_0 a_1 \rightarrow_0 a_2 \rightarrow_0 \ldots$ in $(A, \rightarrow_0)$. Since $f(\rightarrow_0) \subseteq \rightarrow_1^+$, this yields that there is an infinite rewrite sequence $b_0 \rightarrow_1^+ b_1 \rightarrow_1^+ b_2 \rightarrow_1^+ \ldots$, in $(B, \rightarrow_1)$, contradicting the hypothesis. □

The second one is due to Klop. It is Corollary 5.19 in Chapter I of [Klo80]. It makes use of the notion of an increasing abstract rewriting system, which is defined as follows.

Definition 1.1.28. An abstract rewriting system $(A, \rightarrow)$ is said to be *increasing* if there is a mapping $f : A \rightarrow \mathbb{N}$ such that $f(a) < f(b)$ if $a \rightarrow b$.

Lemma 1.1.29. *Let $(A, \rightarrow)$ be an abstract rewriting system that is locally confluent, weakly normalising and increasing. Then $(A, \rightarrow)$ is strongly normalising.*

1.2 Functional Rewriting Systems

If a computation is modelled by an abstract rewriting system, we can only observe *that* some object is rewritten to some other object. We cannot observe *how* it is rewritten, and in particular it is not possible to distinguish between different ways of doing the same. Often it is desirable to have more information concerning a rewrite step. A way to express information concerning a rewrite step without specifying the structure of the expressions that are rewritten, is by considering instead of one rewrite relation a collection of rewrite relations that are distinguishable, for instance by means of indices. In the literature various kinds of indexed rewriting systems have been considered.

In this section we introduce functional rewriting systems. A functional rewriting system is a set with a collection of indexed relations that are all deterministic, in the sense that an object is related to at most one other object. So every indexed relation is a partial function. The indexed relations are obtained as follows: there is a set of indices, and a mapping $\rightarrow$ that maps an index i to a partial function $\xrightarrow{i}$ from objects to objects. All rewriting systems that we will encounter in the remainder of this thesis have an underlying functional rewriting system.

The concept of functional rewriting systems was developed in cooperation with Paula Severi. Functional rewriting systems appear in an early form under the name indexed abstract rewriting systems in [RS95].

Definition 1.2.1. A *functional rewriting system* is a triple $(A, I, \rightarrow)$ consisting of a set of *objects* A, a set of *indices* I and a mapping $\rightarrow : I \rightarrow [A \rightharpoonup A]$ that maps indices to partial functions from A to A.

Functional rewriting systems are appropriate to formalise the notions of redex and redex occurrence in an abstract way.

Definition 1.2.2. Let $(A, I, \rightarrow)$ be a functional rewriting system.

1. Let $a \in A$. A *redex occurrence in a* is an index i such that $\rightarrow(i)(a)$ is defined.

2. A *redex* is a pair (a, i) such that i is a redex occurrence in a.

3. A *rewrite step* is a triple (a, i, b) such that $\rightarrow(i)(a) = b$.

4. The redex occurrence i in a and the redex (a, i) are both said to be *contracted* in the rewrite step $i : a \rightarrow b$.

Notation 1.2.3. A rewrite step (a, i, b) in a functional rewriting system is denoted by $a \xrightarrow{i} b$ or by $i : a \rightarrow b$.

In the more concrete case of λ-calculus with β-reduction, positions play the rôle of indices, and in first-order rewriting systems this is done by pairs consisting of a position and a rewrite rule.

Two typical properties of functional rewriting systems are the following. First, in a functional rewriting system one can have rewrite steps $i : a \to b$ and $j : a \to b$ with $i \neq j$. So it is possible to distinguish between two rewrite steps from an object a to an object b. In this respect functional rewriting systems differ from abstract rewriting systems, because in the latter it is impossible to have more than one rewrite step between two objects.

Second, if $i : a \to b$ and $i : a \to c$ are rewrite steps in a functional rewriting system, then $b = c$. So an index uniquely determines the result of a transformation of an object. This forms a difference with the abstract reduction systems as defined in [Klo92], where it is possible to have $a \to_i b$ and $a \to_i c$ with $b \neq c$. Van Oostrom considers in [Oos94] labelled abstract rewriting systems that differ from functional rewriting systems since, like in abstract reduction systems as in [Klo80], a labelled relation is not deterministic.

Abstract rewriting systems can be mapped to functional rewriting systems and vice versa. The underlying abstract rewriting system of a functional rewriting system $(A, I, \to)$ is the pair $(A, \to')$ with $(a, b) \in \to'$ if and only if there exists $i \in I$ such that $i : a \to b$. Vice versa, we can associate to an abstract rewriting system $(A, \to)$ a functional rewriting system of the form $(A, A, \to)$, with $\to : A \to [A \rightharpoonup A]$ defined by $\to(b)(a) = b$ if $a \to b$ in $(A, \to)$, and $\to(b)(a)$ being undefined otherwise.

The definition of a rewrite sequence in a functional rewriting system is a refinement of the one given for an abstract rewriting system in Definition 1.1.12. It contains also the information about the way the rewrite steps in the sequence are obtained.

Definition 1.2.4. Let $(A, I, \to)$ be a functional rewriting system. For $\alpha \leq \omega$, a *rewrite sequence* of length α issuing from a is defined as a triple (a, α, σ) such that

1. $a \in A$,
2. $\sigma : \alpha \to I$ is a mapping that defines a sequence $\{a_m\}_{m \in \alpha}$ as follows:
   (a) $a_0 = a$,
   (b) $a_m = \to(\sigma(m))(a_{m-1})$, for all m with $m \in \alpha \setminus \{0\}$.

Notation 1.2.5. A rewrite sequence as defined in Definition 1.2.4 is denoted by $\sigma : a_0 \overset{\sigma(1)}{\to} a_1 \overset{\sigma(2)}{\to} a_2 \overset{\sigma(3)}{\to} \ldots$. The length α of a rewrite sequence (a, α, σ) is also denoted by $|\sigma|$.

Morphisms of Functional Rewriting Systems. It is possible to express that there is a correspondence between two rewrite sequences in two functional rewriting systems in a precise way, in the sense that not only the objects but also the rewrite steps are related. One often aims at establishing such a correspondence in translations from one rewriting system to another. It is formalised by the notion of morphism of functional rewriting systems, which is defined as follows.

Definition 1.2.6. Let $(A, I, \rightarrow_0)$ and $(B, J, \rightarrow_1)$ be functional rewriting systems. A *morphism of functional rewriting systems* is a pair of mappings $f = (f_0, f_1)$ with

$$\begin{aligned} f_0 &: A \rightarrow B \\ f_1 &: I \rightarrow J \end{aligned}$$

such that for every rewrite step $i : a \rightarrow_0 b$ in $(A, I, \rightarrow_0)$ we have $f_1(i) : f_0(a) \rightarrow_1 f_0(b)$ in $(B, J, \rightarrow_1)$. In a diagram:

$$\begin{tikzcd} a \arrow[r, "i"', "0"'] \arrow[d, dashed, "f_0"'] & b \arrow[d, dashed, "f_0"] \\ f_0(a) \arrow[r, dashed, "f_1(i)", "1"'] & f_0(b) \end{tikzcd}$$

If f is a morphism of functional rewriting systems from $(A, I, \rightarrow_0)$ to $(B, J, \rightarrow_1)$, and

$$\sigma : a_0 \xrightarrow{i_0}_0 a_1 \xrightarrow{i_1}_0 a_2 \xrightarrow{i_2}_0 \ldots$$

is a rewrite sequence in $(A, I, \rightarrow_0)$, then

$$f_0(a_0) \xrightarrow{f_1(i_0)}_1 f_0(a_1) \xrightarrow{f_1(i_1)}_1 f_0(a_2) \xrightarrow{f_1(i_2)}_1 \ldots$$

is a rewrite sequence in $(B, J, \rightarrow_1)$. This rewrite sequence is denoted by $f(\sigma)$.

Definition 1.2.7. Let f be a morphism between two functional rewriting systems $(A, I, \rightarrow_0)$ and $(B, J, \rightarrow_1)$. Let σ be a rewrite sequence in $(A, I, \rightarrow_0)$. The rewrite sequence $f(\sigma)$ in $(B, J, \rightarrow_1)$ is said to be an *image* of σ and σ is said to be a *lifting* of $f(\sigma)$.

An example of a naturally arising morphism is a mapping that erases underlinings. We will encounter this kind of morphisms in Chapter 2, in Section 2.3 and in Section 2.4. A rewrite sequence in which some symbols are underlined is usually called a lifting of the rewrite sequence without underlinings, which motivated the terminology in Definition 1.2.7.

Descendants and Residuals In rewriting it is often important to be able to trace a property of a term along a rewrite sequence. An important example is the following. Suppose there are two rewrite steps $i : a \to b$ and $i' : a \to b'$ issuing from an object a. It can be the case that these rewrite steps are, intuitively speaking, independent in the sense that after performing the first, one can still perform the second. This is very imprecise because it is in general not the case that $b = a$, and hence one cannot speak of 'the same rewrite step'. In fact, we need a mapping that given a rewrite step $i : a \to b$ assigns to a redex occurrence i' in the object a a set of redex occurrences in the object b. Those redex occurrences in the object b are called the residuals of the redex occurrence i'. Contraction of i can erase i', in that case i' has no residuals in b. It can also be the case that contraction of i multiplies i' with a factor greater than 1, in that case i' has more than one residual in b. And it can be the case that i' has exactly one residual in b.

We will define a residual relation that traces redex occurrences along a rewrite sequence. Before doing so we will define a more general relation, called a descendant relation, that traces a predicate defined on objects and elements of some set which we will call labels. A residual relation is then obtained as a particular case of a descendant relation for the predicate P on $A \times I$ that is defined by $P(a, i)$ if and only if (a, i) is a redex, taking indices for labels. A residual relation should moreover satisfy an additional requirement. The reason for considering the more general notion of a descendant relation is that it appears naturally in various examples. For instance in term rewriting or λ-calculus, it is natural to trace not only redex occurrences but also for instance positions or subterms. As a matter of fact, in Chapter 4 we will need a descendant relation that traces positions in λ-calculus.

Various definitions of descendant and residual relations appear in the literature. Church and Rosser introduce in [CR36] a notion of residual in order to trace redex occurrences in the λ-calculus. For term rewriting, notions of residuals are introduced by Rosen in [Ros73] and by O'Donnell in [O'D77]. In [Bar71], [Klo80], [Klo92] and [Bar84] residuals are defined by using a way to mark subterms, by underlining or colouring them. De Vrijer introduces in [Vri87b] licencing systems for λ-calculus, which can for instance be used to trace residuals. A descendant relation that differs from the traditional one is defined by Khasidashvili, see for instance [Kha92]. In this definition, every symbol in the pattern of the contracted redex occurrence descends to the position of the head-symbol of the contractum, which means that the second requirement in Definition 1.2.8 below doesn't hold. A more abstract approach is taken in [GK96b]. Residual relations are moreover defined by Gonthier, Lévy and Melliès in [GLM92] and by van Oostrom, who introduces residual rewriting systems in [Oos94]. The definitions of descendant and residual relation given below are similar to the ones given in [Oos94] and [GLM92].

Now the definition of a descendant relation is given as follows. We suppose to have a set $\mathcal{L}$ consisting of *labels* at our disposal.

Definition 1.2.8. Let $(A, I, \rightarrow)$ be a functional rewriting system. Let $\mathcal{L}$ be a set of labels and let P be a predicate on $A \times \mathcal{L}$. A *descendant relation for P* is a mapping $\mathsf{Des} : A \times I \rightarrow \mathcal{L} \rightarrow \mathcal{P}(\mathcal{L})$ such that for a rewrite step $i : a \rightarrow a'$ the following holds:

1. if $P(a, l)$ then for all $l' \in \mathsf{Des}(a, i)(l)$ we have $P(a', l')$,
2. if $P(a, l_0)$ and $P(a, l_1)$, and $l' \in \mathsf{Des}(a, i)(l_0)$ and $l' \in \mathsf{Des}(a, i)(l_1)$ then $l_0 = l_1$.

If $l' \in \mathsf{Des}(a, i)(l)$, then l' is said to be a *descendant* of l. The first clause states that if the predicate holds for (a, l) and a is rewritten to a', then for every descendant l' of l the predicate holds for (a', l'). The second clause states that a label can be the descendant of at most one label.

The definition of a descendant relation clearly needs to be extended into two directions. First, one would like to trace a set of labels instead of a single one. This can be done by a straightforward set-theoretical extension of the mapping $\mathsf{Des} : A \times I \rightarrow \mathcal{L} \rightarrow \mathcal{P}(\mathcal{L})$ to a mapping $\mathsf{Des} : A \times I \rightarrow \mathcal{P}(\mathcal{L}) \rightarrow \mathcal{P}(\mathcal{L})$. Second, one would like to trace a set of labels along a rewrite sequence possibly consisting of more than one step. This is realised by the following extension Des^* of Des.

Definition 1.2.9. Let $(A, I, \rightarrow)$ be a functional rewriting system. Let $\mathcal{L}$ be a set and let P be a predicate on $A \times \mathcal{L}$. Let $\mathsf{Des} : A \times I \rightarrow \mathcal{L} \rightarrow \mathcal{P}(\mathcal{L})$ be a descendant relation for P. Let $a_1 \xrightarrow{i_1} a_2 \xrightarrow{i_2} \ldots \xrightarrow{i_m} a_{m+1}$ be a rewrite sequence.

A descendant relation $\mathsf{Des}^* : A \times I^* \rightarrow \mathcal{P}(\mathcal{L}) \rightarrow \mathcal{P}(\mathcal{L})$ is obtained by composition of Des:

1. $\mathsf{Des}^*(a_1, \epsilon)(L) = L$,
2. $\mathsf{Des}^*(a_1, i_1 \ldots i_m)(L) = \mathsf{Des}^*(a_2, i_2 \ldots i_m)(\mathsf{Des}^*(a_1, i_1)(L))$.

If $l' \in \mathsf{Des}^*(a_1, i_1 \ldots i_m)(\{l\})$, then l' is again said to be a *descendant* of l.

Notation 1.2.10. We denote Des^* by Des. We write l for $\{l\}$ and i for (i).

Now a residual relation is defined as a particular case of a descendant relation, satisfying moreover an additional property, namely that a redex occurrence that is contracted does not leave a descendant. Redex occurrences are taken for labels, and we trace the property of being a redex occurrence.

Definition 1.2.11. Let $(A, I, \rightarrow)$ be a functional rewriting system. Let P be the predicate on $A \times I$ defined by $P(a, i)$ holds if and only if (a, i) is a redex. A *residual relation* for $(A, I, \rightarrow)$ is a descendant relation $\mathsf{Des} : A \times I \rightarrow I \rightarrow \mathcal{P}(I)$ for P with the additional property that for a rewrite step $i : a \rightarrow b$ we have

$$\mathsf{Des}(a, i)(i) = \emptyset.$$

Notation 1.2.12. A residual relation is denoted by Res instead of by Des. The extended version of a residual relation, obtained as defined in Definition 1.2.9, is also denoted by Res.

A descendant with respect to the residual relation is called a *residual*. If (a, i) is a redex, $j : a \to a'$ is a rewrite step and $i' \in \mathsf{Res}(a, j)(i)$, then the redex (a', i') is also said to be a residual of (a, i). The index i doesn't have a residual after a rewrite step $i : a \to b$.

Developments. A development is a rewrite sequence in which only residuals of redex occurrences that are present in the initial object are contracted. Hence a development can be seen as a computation of a set of redex occurrences in some object. In this thesis we will encounter developments in λ-calculus with β-reduction in Chapter 2 and developments in higher-order rewriting systems in the Chapters 4, 5 and 6. A development in a functional rewriting system is defined as follows.

Definition 1.2.13. Let $(A, I, \to)$ be a functional rewriting system. Let $a \in A$ be an object and let $\mathcal{I}$ be a set of redex occurrences in a. A *development* of $\mathcal{I}$ is a rewrite sequence $a = a_0 \xrightarrow{i_0} a_1 \xrightarrow{i_1} a_2 \xrightarrow{i_2} \dots$ such that for every m with $m \geq 0$ we have $i_m \in \mathsf{Res}(a, i_0 \dots i_{m-1})(\mathcal{I})$.

Developments are not necessarily finite. However, for several important classes of rewriting systems the result that all developments are finite has been established. This is for instance a classical result for λ-calculus with β-reduction. In Section 2.3 a new proof of finiteness of developments for λ-calculus with β-reduction is given, and we discuss earlier proofs. Melliès gives in [Mel96] axioms that imply finiteness of developments.

In the remainder of this section we will suppose that all developments of a set of redex occurrences are finite.

Definition 1.2.14. Let $(A, I, \to)$ be a functional rewriting system. Let $a \in A$ be an object and let $\mathcal{I}$ be a set of redex occurrences in a. A development $a_0 \xrightarrow{i_0} \dots \xrightarrow{i_{m-1}} a_m$ of $\mathcal{I}$ is said to be *complete* if $\mathsf{Res}(a_0, i_0 \dots i_{m-1})(\mathcal{I}) = \emptyset$.

Confluence. We present the classical result that confluence is guaranteed if for every set $\mathcal{I}$ of redex occurrences in an object all complete developments of $\mathcal{I}$ are finite and end in the same term. In the presentation we follow mainly [Oos94] and [Mel96].

Definition 1.2.15. Let $(A, I, \rightarrow)$ be a functional rewriting system.

1. Let $a \in A$. A set $\mathcal{I}$ of redex occurrences in a is said to be *consistent* if every complete development of $\mathcal{I}$ ends in the same term and induces the same residual relation.

2. If for every object a every set of redex occurrences in a is consistent, then the functional rewriting system $(A, I, \rightarrow)$ is said to be consistent.

Notation 1.2.16. Let $\mathcal{I}$ be a consistent set of redex occurrences in an object a. Suppose performing a complete development of $\mathcal{I}$ yields an object b. Then a rewrite sequence that is a complete development of $\mathcal{I}$ is denoted by $\mathcal{I} : a \mathrel{-\!\!\circ\!\!\!\rightarrow} b$ or by $a \overset{\mathcal{I}}{\mathrel{-\!\!\circ\!\!\!\rightarrow}} b$.

If $(A, I, \rightarrow)$ is a consistent functional rewriting system, then $(A, \mathcal{P}(I), \mathrel{-\!\!\circ\!\!\!\rightarrow})$ with $\mathcal{I} : a \mathrel{-\!\!\circ\!\!\!\rightarrow} b$ if a rewrites to b by a complete development in $(A, I, \rightarrow)$, is also a functional rewrite system. A rewrite sequence in $(A, \mathcal{P}(I), \mathrel{-\!\!\circ\!\!\!\rightarrow})$ is said to be a *development rewrite sequence* in $(A, I, \rightarrow)$. Note that we have $\rightarrow \subseteq \mathrel{-\!\!\circ\!\!\!\rightarrow} \subseteq \twoheadrightarrow$.

We present a construction, called the orthogonal projection, that is used in the proof that consistent functional rewriting systems are confluent. This construction, and a variation of it, will be used in Chapter 6.

Definition 1.2.17. Let $(A, I, \rightarrow)$ be a consistent functional rewriting system. Let $\sigma : a_0 \xrightarrow{i_0} a_1 \xrightarrow{i_1} \ldots$ be a rewrite sequence and let $j : a_0 \rightarrow b_0$ be a rewrite step. Define

$$\begin{aligned}
\mathcal{J}_0 &= \{j\}, \\
\mathcal{J}_{m+1} &= \mathsf{Res}(a_m, i_m)(\mathcal{J}_m) \quad \text{for } m \geq 0, \\
\mathcal{I}_m &= \mathsf{Res}(a_m, \mathcal{J}_m)(i_m) \quad \text{for } m \geq 0.
\end{aligned}$$

The *orthogonal projection* of σ over $j : a_0 \rightarrow b_0$ is the development rewrite sequence $\tau : b_0 \overset{\mathcal{I}_0}{\mathrel{-\!\!\circ\!\!\!\rightarrow}} b_1 \overset{\mathcal{I}_1}{\mathrel{-\!\!\circ\!\!\!\rightarrow}} b_2 \overset{\mathcal{I}_2}{\mathrel{-\!\!\circ\!\!\!\rightarrow}} \ldots$. In a picture:

$$\begin{tikzcd}
\sigma : & a_0 \arrow[r, "i_0"] \arrow[d, "\mathcal{J}_0"', "\circ" marking] & a_1 \arrow[r, "i_1"] \arrow[d, "\mathcal{J}_1"', "\circ" marking] & a_2 \arrow[r, dotted, no head] \arrow[d, "\mathcal{J}_2"', "\circ" marking] & {} \\
\tau : & b_0 \arrow[r, "\mathcal{I}_0"', "\circ" marking] & b_1 \arrow[r, "\mathcal{I}_1"', "\circ" marking] & b_2 \arrow[r, dotted, no head] & {}
\end{tikzcd}$$

By constructing the orthogonal projection of a rewrite sequence $\sigma : a \twoheadrightarrow b$ over a rewrite step $i : a \rightarrow c$, a common reduct of the objects b and c is found. Using the orthogonal projection, the proof that consistency implies confluence follows by a straightforward induction.

Theorem 1.2.18. *All consistent functional rewriting systems are confluent.*

Proof. Let $(A, I, \rightarrow)$ be a consistent functional rewriting system. Suppose $a \twoheadrightarrow b$ and $a \twoheadrightarrow c$. The proof proceeds by induction on the length of $a \twoheadrightarrow c$ and is suggested by the following diagram:

$$\begin{tikzcd}
a \arrow[r, two heads] \arrow[d] & b \arrow[d, two heads] \\
c' \arrow[r, two heads] \arrow[d, two heads] & d' \arrow[d, two heads] \\
c \arrow[r, two heads] & d
\end{tikzcd}$$

The rewrite sequence $c' \twoheadrightarrow d'$ is the orthogonal projection of $a \twoheadrightarrow b$ over $a \rightarrow c'$. $\square$

We conclude this section by considering elementary diagrams as introduced in [Klo80, p.59].

Definition 1.2.19. Let $(A, I, \rightarrow)$ be a functional rewriting system.

1. Let $a \in A$ and let i and j be redex occurrences in a. An *elementary diagram of i and j* is a pair of two complete developments of $\{i, j\}$ of the form $i; \sigma'$ and $j; \tau'$.

2. The functional rewriting system $(A, I, \rightarrow)$ is said to *have elementary diagrams* if for every object a and every two redex occurrences i and j in a every complete development of $\{i, j\}$ ends in the same term and induces the same residual relation.

The requirement of having elementary diagrams is a local version of the requirement of being consistent. If all developments are finite, then consistency is implied by the property of having elementary diagrams. This is shown in the following proposition, which is proved in [CFC58], see also [Klo80], [Oos94] and [Mel96].

Lemma 1.2.20. *Let $(A, I, \rightarrow)$ be a functional rewriting system. Suppose that all developments are finite and that $(A, I, \rightarrow)$ has elementary diagrams. Then we have that $(A, I, \rightarrow)$ is consistent.*

Strategies. In rewriting, one considers rewrite strategies that indicate which path or which paths through the graph of reducts of an object we should follow. Usually, a rewrite strategy is a mapping that takes as input an object and gives as output a recipe that indicates how to rewrite this expression, for instance a set of rewrite steps issuing from this object from which we may choose one. In general, rewrite strategies are used to find a rewrite path with a certain property. For instance, we might want to find a rewrite sequence that ends in a normal form, or maybe one that is infinite. In this section we define rewrite strategies for functional rewriting systems.

Definition 1.2.21. Let $(A, I, \rightarrow)$ be a functional rewriting system. A *one-step rewrite strategy* is a mapping $\mathsf{F} : A \rightarrow \mathcal{P}(I)$ such that for every $a \in A$ every index $i \in \mathsf{F}(a)$ is a redex occurrence in a.

A many-step strategy assigns to an object rewrite sequences starting in that object.

Definition 1.2.22. Let $(A, I, \rightarrow)$ be a functional rewriting system. A *many-step rewrite strategy* is a mapping $\mathsf{F} : A \rightarrow \mathcal{P}(I^+)$ such that for all $a \in A$, if $i_0 \dots i_m \in \mathsf{F}(a)$, then there exist $b_1, \dots, b_m, c \in A$ such that $a \xrightarrow{i_0} b_1 \xrightarrow{i_1} \dots \xrightarrow{i_{m-1}} b_m \xrightarrow{i_m} c$ is a rewrite sequence.

Alternatively, a many-step rewrite strategy can be defined as a one-step rewrite strategy for the functional rewriting system $(A, I^+, \rightarrow^+)$.

Definition 1.2.23. Let F be a one-step or a many-step rewrite strategy for a functional rewriting system $(A, I, \rightarrow)$.

1. F is *deterministic* if for every $a \in A$ the set $\mathsf{F}(a)$ contains at most one element.

2. Otherwise F is said to be *non-deterministic.*

In the sequel we only consider one-step rewrite strategies, therefore we say simply 'strategy' instead of 'one-step rewrite strategy'.

Definition 1.2.24. Let F be a strategy in a functional rewriting system $(A, I, \rightarrow)$.

1. A rewrite step $i : a \rightarrow b$ is said to be an F*-rewrite step* if $i \in \mathsf{F}(a)$.

2. A rewrite sequence $\sigma : a_0 \xrightarrow{i_0} a_1 \xrightarrow{i_1} a_2 \xrightarrow{i_2} \dots$ is said to be an F*-rewrite sequence* if every rewrite step $i_m : a_m \rightarrow a_{m+1}$ in σ is an F-rewrite step.

3. An F-rewrite sequence is said to be *complete* if it is either infinite or it ends an object from which no F-rewrite steps are possible.

Definition 1.2.25.

1. A strategy F is *normalising* if for every $a \in A$ that is weakly normalising, every complete F-rewrite sequence starting in a ends in a normal form.
2. A strategy F is *perpetual* if for every $a \in A$ that admits an infinite rewrite sequence every complete F-rewrite sequence is infinite.

In Chapter 6 we will study outermost-fair rewrite sequences, which are rewrite sequences that either end in a normal form or do not contain an infinite chain of outermost redexes. In this chapter we present a general definition of P-fair rewrite sequences. Therefore we first formalise the notion of a chain of descendants.

Definition 1.2.26. Let $(A, I, \rightarrow)$ be a functional rewriting system. Let $\mathcal{L}$ be a set and let $\mathsf{Des} : A \times I \rightarrow \mathcal{L} \rightarrow \mathcal{P}(\mathcal{L})$ be a descendant relation for a property P on $A \times \mathcal{L}$. Let $\sigma : a_0 \xrightarrow{i_0} a_1 \xrightarrow{i_1} a_2 \xrightarrow{i_2} \ldots$ be an infinite rewrite sequence.

1. An *infinite P-chain* in σ with respect to the descendant relation Des is an infinite sequence $l_m, l_{m+1}, l_{m+2}, \ldots$ for some $m \geq 0$, such that
   (a) $P(a_m, l_m)$,
   (b) $l_{n+1} \in \mathsf{Des}(a_n, i_n)(l_n)$ for all $n \geq m$.
2. An *infinite residual chain* in σ is an infinite P-chain in σ with respect to the residual relation, with P on $A \times I$ defined by $P(a, i)$ if and only if (a, i) is a redex.

Notation 1.2.27. An infinite descendant chain and an infinite residual chain as in the previous definition are denoted by $\vec{l}$ and $\vec{\imath}$.

Now the notions P-fair and redex-fair are defined as follows.

Definition 1.2.28. Let $(A, I, \rightarrow)$ be a functional rewriting system.

1. A rewrite sequence is said to be *P-fair* either if ends in a normal form, or if it is infinite and does not contain an infinite P-chain.
2. A rewrite sequence is said to be *redex-fair* either if ends in a normal form, or if it is infinite and does not contain an infinite residual chain.

Intuitively, in a P-fair rewrite sequence a label l satisfying P is eventually eliminated. In a redex-fair rewrite sequence, every redex occurrence is eventually eliminated.

Finally, we consider cofinal rewrite sequences.

Definition 1.2.29. A rewrite sequence $\sigma : a_0 \xrightarrow{i_0} a_1 \xrightarrow{i_1} \dots$ is *cofinal* if for every object b with $a_0 \twoheadrightarrow b$ there exists for some $m \leq |\sigma|$ an object a_m in σ such that $b \twoheadrightarrow a_m$.

Proposition 1.2.30. *Let $(A, I, \rightarrow)$ be a consistent functional rewriting system and let a be a weakly normalising object in A. Every cofinal rewrite sequence starting in a is finite.*

Proof. Let σ be a cofinal rewrite sequence starting in a. Let $a \twoheadrightarrow b$ with b a normal form of a. Since σ is cofinal, b can be rewritten to an object in σ, hence σ ends in b. $\square$

The following result is due to O'Donnell [O'D77].

Proposition 1.2.31. *Let $(A, I, \rightarrow)$ be a consistent functional rewriting system. Every redex-fair rewrite sequence is cofinal.*

Proof. Suppose that $\sigma : a_0 \xrightarrow{i_0} a_1 \xrightarrow{i_1} a_2 \xrightarrow{i_2} \dots$ is a redex-fair rewrite sequence. If σ ends in a normal form, then the statement clearly holds since $(A, I, \rightarrow)$ is confluent. Suppose that σ is infinite and let $\tau : a_0 \twoheadrightarrow b_0$. We prove by induction on the length of τ that an object a_m in σ exists such that $b_0 \twoheadrightarrow a_m$.

If $|\tau| = 0$, then $b_0 \twoheadrightarrow a_0$.

Suppose $|\tau| > 0$, so $\tau : a_0 \xrightarrow{j} a'_0 \twoheadrightarrow b_0$. Let σ' be the orthogonal projection of σ over $j : a_0 \rightarrow a'_0$.

The situation is illustrated in the following diagram.

$$\begin{tikzcd}
\sigma : & a_0 \arrow[r, "i_0"] \arrow[d, "j"'] & a_1 \arrow[r, "i_1"] \arrow[d, "\mathcal{J}_1"'] & a_2 \arrow[r, "i_2"] \arrow[d, "\mathcal{J}_2"'] & \cdots\cdots & a_n \arrow[r, "i_n"] \arrow[d, "\emptyset"'] & a_{n+1} \arrow[d, "\emptyset"'] & \cdots\cdots \\
\sigma' : & a'_0 \arrow[r, "\mathcal{I}_0"] & a'_1 \arrow[r, "\mathcal{I}_1"] & a'_2 \arrow[r, "\mathcal{I}_2"] & \cdots\cdots & a'_n \arrow[r, "\mathcal{I}'_n"] & a'_{n+1} & \cdots\cdots
\end{tikzcd}$$

Since σ is redex-fair, there exists an a_n in σ such that $\mathsf{Res}(a_0, i_0 \dots i_{n-1})(j) = \emptyset$. Hence the final part of σ' coincides with σ, so σ' is redex-fair. By induction hypothesis, b_0 can be rewritten to an object in σ'. We conclude that σ is cofinal.

$\square$

1.3 Abstract Rewriting Systems with Typing

In computing science the notion of typing appears frequently. A typing relation can be thought of as expressing that an object has a particular property. In this section we introduce abstract typing systems, that are meant to model the typing relation in a general way. An abstract typing system is a binary relation on a set. So a priori there is no difference between a rewrite relation and a typing relation, but clearly rewrite relations and typing relations are studied with different intentions in mind. One can consider sets that are equipped with both a typing and a rewrite relation. We define an abstract rewriting system with typing as a rewriting system in which the rewrite relation and the typing relation are compatible in a natural way. Relations on this definition can be obtained by requiring another kind of compatibility. Abstract typing systems are defined in collaboration with Paula Severi.

Abstract Typing Systems.

Definition 1.3.1. An *abstract typing system* is a pair $(A, :)$ consisting of a set A of *objects* and a relation $: \subseteq A \times A$, called a *typing relation*.

Notation 1.3.2. We write $a : b$ instead of $(a, b) \in :$.

Definition 1.3.3. Let $(A, :)$ be a typing system.

1. If $a : b$ then we say that *a has type b* or *a is of type b*.
2. We say that $a \in A$ is *typable* if there exists $b \in A$ such that $a : b$.
3. We say that $b \in A$ is *inhabited* if there exists $a \in A$ such that $a : b$.

Abstract Rewriting Systems with Typing. We define various natural ways in which a typing relation and a rewrite relation on some set may interact. Subject reduction expresses that the type of an object does not change if the object is rewritten. Furthermore, instead of rewriting the object that is typed, one can also rewrite the object that is inhabited. It is then possible to consider various weaker variants of the property of subject reduction, for instance by requiring that a reduct of an object a with type b is typable by a reduct of b. We will only introduce the notions that will be used in the remainder of this thesis, more precisely, in Section 4.1 of Chapter 4.

Definition 1.3.4. Let A be a set. Let $(A, \rightarrow)$ be an abstract rewriting system and let $(A, :)$ be a typing system.

1. $(A, \rightarrow)$ satisfies the property of *subject reduction* with respect to $(A, :)$ if for all $a \in A$ the following holds: if $a : b$ and $a \rightarrow a'$ then $a' : b$.
2. $(A, \rightarrow)$ satisfies the property of *subject conversion* with respect to $(A, :)$ if for all $a \in A$ the following holds: if $a : b$ and $a \leftrightarrow a'$ then $a' : b$.

3. $(A, \rightarrow)$ satisfies the property of *type reduction* with respect to $(A, :)$ if for all $a \in A$ the following holds: if $a : b$ and $b \rightarrow b'$ then $a : b'$.

4. $(A, \rightarrow)$ satisfies the property of *type conversion* with respect to $(A, :)$ if for all $a \in A$ the following holds: if $a : b$ and $b \leftrightarrow b'$ then $a : b'$.

We define an abstract rewriting system with typing as an abstract rewriting system satisfying subject reduction and type reduction. Again, one can easily imagine variations on this theme.

Definition 1.3.5. Let $(A, \rightarrow)$ be an abstract rewriting system and let $(A, :)$ be a typing system. If $(A, \rightarrow)$ satisfies the properties of subject reduction and type reduction with respect to $(A, :)$, then the triple $(A, \rightarrow, :)$ is said to be an *abstract rewriting system with typing.*

Often one is only interested in objects that are typable or inhabited.

Definition 1.3.6. Let P be a property on A. The abstract rewriting system with typing $(A, \rightarrow, :)$ has property P if every $a \in A$ that is typable or inhabited satisfies property P.

We will encounter an example of a rewriting system with typing in Section 2.5 of Chapter 2 in which simply typed λ-calculus is considered. Moreover, the substitution calculus of a higher-order rewriting system as defined in Chapter 4 will be an abstract rewriting system with typing. Types can be used to express properties as illustrated in the following example.

Example 1.3.7. We consider a typed version of the abstract rewriting system given in Example 1.1.24. We consider the following set of objects:

$$\{a, b, c, d, \omega, \mathsf{SN}\}.$$

The object ω expresses the property of admitting an infinite rewrite sequence and the object SN expresses the property strong normalisation. The rewrite relation is as follows:

$$\begin{tikzcd} a & b \arrow[l] \arrow[r, bend left] & c \arrow[l, bend left] \arrow[r] & d & & \omega \arrow[r] & \mathsf{SN} \end{tikzcd}$$

The typing relation is defined by $a : \mathsf{SN}, b : \omega, c : \omega, d : \mathsf{SN}$. The rewrite relation does not satisfy the property of subject reduction with respect to the typing relation, since for instance $b \rightarrow a$ and $b : \omega$ but not $a : \omega$. It satisfies a weaker property: we have that the type of b can be rewritten to the type of a.

A special case of an abstract rewriting system with typing arises when the objects that have a type and the objects that are inhabited form two disjoint sets, and only elements of the first set can be rewritten. Then we have an abstract rewriting system with typing of the form $(A \cup B, \rightarrow, :)$ such that

1. $A \cap B = \emptyset$,
2. $\rightarrow \subseteq A \times A$,
3. $: \subseteq A \times B$.

In Chapter 2 we will see that simply typed λ-calculus is an abstract rewriting system with typing of this form. There are rewriting systems studied in the literature that are not of this form, for instance the Calculus of Constructions defined in [CH88].

The framework of abstract rewriting systems with typing permits to define the notion uniqueness of types as follows.

Definition 1.3.8. Let $(A, \rightarrow :)$ be an abstract rewriting system with typing. An object a has the *property of uniqueness of types* if for all $b, b' \in A$ such that $a : b$ and $a : b'$. we have that $b \leftrightarrow^* b'$.

Typing Systems with Environments. More structure can be given to a typing system by specifying how $a : b$ is obtained. We define abstract typing systems with environments as a triple consisting of a set of objects, a set of environments and a mapping :. Typing relations are obtained by applying : to an environment. This section is added to show that it is possible to express typing systems in which types depend on environments in an abstract way. We won't use the concept of typing systems with environments in the remainder of this thesis.

Definition 1.3.9. A *abstract typing system with environments* is a triple $(A, C, :)$ consisting of a set A, a set C and a mapping : from C to $\mathcal{P}(A \times A)$.

Notation 1.3.10. Elements of C are denoted by $\Gamma, \Delta, \ldots$. We write $\Gamma \vdash a : b$ instead of $(a, b) \in :(\Gamma)$.

Definition 1.3.11. Let $(A, C, :)$ be a typing system with environments.

1. If $\Gamma \vdash a : b$, then we say that *a has type b in Γ* or *a is of type b in Γ*. If a $\Gamma \in C$ exists such that $\Gamma \vdash a : b$, then we say that *a has type b* or *a is of type b*.
2. We say that $a \in A$ is *typable in* Γ if there exists $b \in A$ such that $\Gamma \vdash a : b$. We say that *$a \in A$ is typable* if there exist $\Gamma \in C$ and $b \in A$ such that $\Gamma \vdash a : b$.

3. We say that $b \in A$ is *inhabitable in* Γ if there exists $a \in A$ such that $\Gamma \vdash a : b$. We say that $b \in A$ is *inhabitable* if there exist $\Gamma \in C$ and $a \in A$ such that $\Gamma \vdash a : b$.

A special case of a typing system with environment arises when $:$ is a mapping from C to partial functions from A to A. Then all typable objects have a unique type.

Definition 1.3.4 can easily be adapted to the case of typing systems with environments. We won't need the definition in the sequel so we just give the definition of subject reduction and of type reduction.

Definition 1.3.12. Let $(A, \rightarrow)$ be an abstract rewriting system and let $(A, C, :)$ be a typing system with environments.

1. $(A, \rightarrow)$ satisfies the property of *subject reduction* with respect to $(A, C, :)$ if $a \rightarrow a'$ and $\Gamma \vdash a : b$ implies that $\Gamma \vdash a' : b$.

2. $(A, \rightarrow)$ satisfies the property of *type reduction* with respect to $(A, C, :)$ if $b \rightarrow b'$ and $\Gamma \vdash a : b$ implies that $\Gamma \vdash a : b'$.

Definition 1.3.13. If $(A, \rightarrow)$ is an abstract rewriting system that satisfies subject reduction and type reduction with respect to the typing system with environments $(A, C, :)$, then the tuple $(A, C, \rightarrow, :)$ is said to be an *abstract rewriting system with typing with environments.*

Again, it is often the case that abstract rewriting systems with typing with environments occur in a special form, namely such that the objects that are typable and the objects that are inhabited form two disjoint sets, and only objects of the first set are rewritten.

Chapter 2

Normalisation in Lambda Calculus

In this chapter normalisation in λ-calculus with β-reduction is studied. We give a characterisation of strongly normalising λ-terms in the form of an inductively defined set $\mathcal{SN}$. This characterisation is used to give new and simple proofs of various classical results concerning normalisation in λ-calculus with β-reduction.

In Section 2.3 we give two proofs of finiteness of developments. Developments are defined as usual, using underlined λ-terms and underlined β-reduction. The first proof of finiteness of developments makes use of an inductively defined set $\mathcal{FD}$. We show that the set of all underlined terms coincides with the set $\mathcal{FD}$, and, by an easy induction on the definition of $\mathcal{FD}$, that every term in $\mathcal{FD}$ admits only finite developments. The second proof of finiteness of developments makes direct use of the characterisation of strongly normalising terms: every development is mapped to a β-rewrite sequence in $\mathcal{SN}$ preserving rewrite steps. This yields that every development is finite.

In Section 2.4 we consider superdevelopments. A superdevelopment is a rewrite sequence in which besides residuals of redex occurrences that are present in the initial term also redex occurrences that are created 'upwards' during rewriting may be contracted. So superdevelopments form a generalisation of developments. We give two proofs of finiteness of superdevelopments, as in Section 2.3 for the case of developments: one using an inductively defined set $\mathcal{FSD}$ and one using the characterisation of strongly normalising terms directly.

In Section 2.5 the characterisation of strongly normalising terms is used to give a short proof of strong normalisation of simply typed λ-calculus.

To start with, the syntax of untyped λ-calculus is given in Section 2.1, in order to fix the notation and to collect some definitions that will be used in Chapter 4.

The work presented in this chapter was performed in collaboration with Paula Severi and is reported in [RS95].

2.1 Lambda Calculus

In this section we recall the definition of untyped λ-calculus with β-reduction in order to fix the notation. We consider λ-calculus with constants. The standard references for λ-calculus are [Bar84] and [HS86].

λ-terms. We suppose that there is a set denoted by Var consisting of infinitely many *variables*. Variables are denoted by $x, y, z, \ldots$. We suppose further that there is a set denoted by C consisting of possibly infinitely many *constants*. Constants are denoted by $f, g, h, \ldots$. Lambda terms are built from variables, constants, λ-abstraction and λ-application as specified in the following definition.

Definition 2.1.1. The set of *λ-terms*, denoted by Λ, is the smallest set satisfying the following:

1. if $x \in \mathsf{Var}$ then $x \in \Lambda$,
2. if $f \in \mathsf{C}$ then $f \in \Lambda$,
3. if $x \in \mathsf{Var}$ and $M \in \Lambda$ then $\lambda x.M \in \Lambda$,
4. if $M \in \Lambda$ and $N \in \Lambda$ then $MN \in \Lambda$.

Lambda terms are denoted by $M, N, P, Q, \ldots$.

Definition 2.1.2. The set of *free variables* of a λ-term M, denoted by $\mathsf{FV}(M)$, is defined by induction on the structure of M.

1. $\mathsf{FV}(x) = \{x\}$ for a variable x,
2. $\mathsf{FV}(f) = \emptyset$ for a constant f,
3. $\mathsf{FV}(\lambda x.M_0) = \mathsf{FV}(M_0) \setminus \{x\}$,
4. $\mathsf{FV}(M_0M_1) = \mathsf{FV}(M_0) \cup \mathsf{FV}(M_1)$.

A variable x is said to *occur free* or to *be free* in M if $x \in \mathsf{FV}(M)$. A variable in M that is not free in M is said to be *bound in M* or to *occur bound in M*. A λ-term that doesn't contain free variables is said to be *closed*.

As usual we identify terms that are equal up to a renaming of bound variables or α-conversion. So we identify for instance $\lambda x.x$ and $\lambda y.y$. We do not make a syntactic distinction between free and bound variables, but we assume that bound variables are renamed whenever necessary. This is called the 'variable convention' in [Bar84]. Assuming the variable convention we can give the definition of substitution of a term for a variable as follows.

Definition 2.1.3. Let $M \in \Lambda$ and $N \in \Lambda$. The *substitution of N for x in M*, denoted by $M[x := N]$, is defined by induction on the structure of M as follows.

1. $x[x := N] = N$,
2. $y[x := N] = y$ if $y \neq x$,
3. $f[x := N] = f$,
4. $(\lambda y.M_0)[x := N] = \lambda y.M_0[x := N]$,
5. $(M_0 M_1)[x := N] = (M_0[x := N])(M_1[x := N])$.

By the variable convention, in the fourth clause of the previous definition the variable y does not occur free in the term N, and $x \neq y$.

Definition 2.1.4. The set of *positions in λ-terms*, denoted by Pos, is $\{0, 1\}^*$. Positions in λ-terms are denoted by $\phi, \chi, \psi, \ldots$. There is an operator for *concatenation* of positions that is denoted by juxtaposition and that is supposed to be associative. The neutral element for concatenation of positions is the empty sequence denoted by ϵ.

Definition 2.1.5.

1. A partial order denoted by $\preceq$ on the set Pos is defined as follows: $\phi \preceq \chi$ if there exists a $\phi' \in \mathsf{Pos}$ such that $\phi\,\phi' = \chi$. The strict partial order, denoted by $\prec$, is obtained by requiring $\phi' \neq \epsilon$.
2. Two positions ϕ and χ are said to be *disjoint*, denoted by $\phi \mid \chi$, if they are incomparable with respect to $\preceq$.

We give the definitions of the set of positions of a term M and of the subterm of a term M at a position ϕ.

Definition 2.1.6. Let $M \in \Lambda$. The set of *positions of M*, denoted by $\mathsf{Pos}(M)$, is defined by induction on the structure of M as follows:

1. $\mathsf{Pos}(x) = \{\epsilon\}$,
2. $\mathsf{Pos}(f) = \{\epsilon\}$,
3. $\mathsf{Pos}(\lambda x.M_0) = \{\epsilon\} \cup \{0\,\phi_0 \mid \phi_0 \in \mathsf{Pos}(M_0)\}$,
4. $\mathsf{Pos}(M_0 M_1) = \{\epsilon\} \cup \{0\,\phi_0 \mid \phi_0 \in \mathsf{Pos}(M_0)\} \cup \{1\,\phi_1 \mid \phi_1 \in \mathsf{Pos}(M_1)\}$.

Definition 2.1.7. Let $M \in \Lambda$ and let $\phi \in \mathsf{Pos}(M)$. The *subterm of M at position ϕ*, denoted by $M|_\phi$, is defined as follows:

1. $M|_\epsilon = M$,
2. $(\lambda x.M_0)|_{0\,\phi_0} = M_0|_{\phi_0}$,
3. $(M_0M_1)|_{0\,\phi_0} = M_0|_{\phi_0}$,
4. $(M_0M_1)|_{1\,\phi_1} = M_1|_{\phi_1}$.

We give the definition of the replacement of the subterm of M at position ϕ by a term N.

Definition 2.1.8. Let $M \in \Lambda$ and let $\phi \in \mathsf{Pos}(M)$. Let $N \in \Lambda$. The *replacement of the subterm of* M *at position* ϕ *by* N, denoted by $M[\phi \leftarrow N]$, is defined as follows:

1. $M[\epsilon \leftarrow N] = N$,
2. $(\lambda x.M_0)[0\,\phi_0 \leftarrow N] = \lambda x.M_0[\phi \leftarrow N]$,
3. $(M_0M_1)[0\,\phi_0 \leftarrow N] = (M_0[\phi_0 \leftarrow N])M_1$,
4. $(M_0M_1)[1\,\phi_1 \leftarrow N] = M_0(M_1[\phi_1 \leftarrow N])$.

The difference between substitution and replacement is that free variables cannot become bound by a substitution, but they can become bound by a replacement. This difference is illustrated by the following example. We have $(\lambda x.y)[y := x] = \lambda x'.x$. Note that the bound variable is renamed. In case of a replacement, we have $(\lambda x.y)[0 \leftarrow x] = \lambda x.x$. In the second case the variable x is 'captured', in the first case it isn't. Moreover, with a substitution one can change a term only at a position where a free variable occurs, in the case of a replacement there is no such restriction.

β-reduction. The set of λ-terms is equipped with a rewrite relation called β-reduction. We define the notions of β-redex occurrence, β-redex and β-rewrite step.

Definition 2.1.9.

1. Let M be a λ-term. A *β-redex occurrence in* M is a pair (ϕ, β) such that $M|_\phi = (\lambda x.P)Q$, for some $P, Q \in \Lambda$.
2. A *β-redex* is a pair $(M, (\phi, \beta))$ such that (ϕ, β) is a β-redex occurrence in M.
3. A *β-rewrite step* is a triple $(M, (\phi, \beta), M')$ such that
   (a) $M|_\phi = (\lambda x.P)Q$,

(b) $M' = M[\phi \leftarrow P[x := Q]]$,

for some $P, Q \in \Lambda$.

4. We say that the β-redex $(M, (\phi, \beta))$ *is contracted* in the β-rewrite step $(M, (\phi, \beta), M')$, and also that the β-redex occurrence (ϕ, β) *is contracted* in the β-rewrite step $(M, (\phi, \beta), M')$.

In this chapter we consider λ-calculus with only β-reduction and hence it is not necessary to specify the β-rewrite rule in the definition of redex occurrence, redex and rewrite step. We nevertheless do so in order to be able to use this definition also in Chapter 4 where the rewrite relation is not only induced by the β-rule but also by the $\overline{\eta}$-rule.

Notation 2.1.10. A β-rewrite step $(M, (\phi, \beta), M')$ as in Definition 2.1.9 is usually denoted by $(\phi, \beta) : M \rightarrow M'$ or by $M \xrightarrow{\phi}_\beta M'$.

Definition 2.1.11. We denote by $\mathfrak{U}_\beta$ the set consisting of all pairs of the form (ϕ, β) with $\phi \in \mathsf{Pos}$. The set of all β-redex occurrences in a term M is denoted by $\mathfrak{U}_\beta(M)$.

The rewriting system λ-calculus has an underlying abstract rewriting system of the form $(\Lambda, \rightarrow_\beta)$, with $M \rightarrow_\beta M'$ if there is a position ϕ in M such that $M \xrightarrow{\phi}_\beta M'$. The underlying functional rewriting system of λ-calculus with β-reduction is $(\Lambda, \mathfrak{U}_\beta, \rightarrow)$ with $\rightarrow((\phi, \beta))(M) = M'$ if $M \xrightarrow{\phi}_\beta M'$, and $\rightarrow((\phi, \beta))(M)$ being undefined otherwise.

Example 2.1.12. We give an example of a rewrite sequence in λ-calculus. We use the standard abbreviations $K = \lambda x.\lambda x'.x$ and $\Omega = (\lambda y.yy)(\lambda y.yy)$. Then

$$Kx\Omega \rightarrow_\beta Kx\Omega \rightarrow_\beta Kx\Omega \rightarrow_\beta \ldots$$

is an infinite β-rewrite sequence in which Ω is rewritten to itself in every rewrite step. An example of a finite rewrite sequence starting in the same term is

$$Kx\Omega \rightarrow_\beta x.$$

Descendants. We define a descendant relation that traces positions along a β-rewrite sequence.

Definition 2.1.13. A descendant relation

$$\mathsf{Des}_\beta : \Lambda \times \mathfrak{U}_\beta \rightarrow \mathsf{Pos} \rightarrow \mathcal{P}(\mathsf{Pos})$$

for the predicate $P(M, \phi)$ on $\Lambda \times \mathsf{Pos}$ with $P(M, \phi)$ if and only if $\phi \in \mathsf{Pos}(M)$, is defined as follows. Let $(\phi, \beta) : M \rightarrow M'$ be a β-rewrite step with $M|_\phi = (\lambda x.P)Q$. Let $\chi \in \mathsf{Pos}(M)$.

1. If it is not the case that $\phi \preceq \chi$, then
$$\mathsf{Des}_\beta(M,(\phi,\beta))(\chi) = \{\chi\}.$$
2. If $\chi = \phi\,0\,0\,\chi'$ and $M|_\chi \neq x$, then
$$\mathsf{Des}_\beta(M,(\phi,\beta))(\chi) = \{\phi\,\chi'\}.$$
3. If $\chi = \phi\,1\,\chi'$, then
$$\mathsf{Des}_\beta(M,(\phi,\beta))(\chi) = \{\phi\,\phi'\,\chi' \mid \phi' \in X\}$$
where $X = \{\phi' \in \mathsf{Pos}(M) \mid M|_{\phi'} = x\}$.
4. In all other cases,
$$\mathsf{Des}_\beta(M,(\phi,\beta))(\chi) = \emptyset.$$

Using the descendant relation Des_β every position in a λ-term can be traced along a β-rewrite sequence. It is clear from the definition that the position ϕ doesn't leave a descendant after the rewrite step $(\phi,\beta) : M \to M'$. Moreover, if $(\phi,\beta) : M \to M'$ and $\chi \in \mathfrak{U}_\beta(M)$, then for every $\chi' \in \mathsf{Des}_\beta(M,(\phi,\beta))(\chi)$ we have $\chi' \in \mathfrak{U}_\beta(M')$. Hence if we trace only positions that define a redex occurrence, then Des_β can serve as a residual relation. Formally, we obtain a residual relation $\mathsf{Res}_\beta : \Lambda \times \mathfrak{U}_\beta \times \mathfrak{U}_\beta \to \mathcal{P}(\mathfrak{U}_\beta)$.

Standardisation. In this chapter we will make use of an important result concerning λ-calculus with β-reduction, called the standardisation theorem. It states that if a λ-term M can be rewritten to a λ-term M', then M' can be obtained from M by rewriting from left to right in the string representation of the terms, possibly jumping over some redex occurrences. Such a rewrite sequence is said to be standard. The definition of a standard rewrite makes use of the notion of a leftmost β-redex.

Definition 2.1.14.

1. Let M be a λ-term and let $\phi, \phi' \in \mathsf{Pos}(M)$. We say that ϕ *occurs to the left of* ϕ' if $\phi \neq \phi'$ and moreover either $\phi \preceq \phi'$ or $\phi = \chi\,0\,\phi_0$ and $\phi' = \chi\,1\,\phi'_0$.
2. Let (ϕ,β) and (ϕ',β) be two β-redex occurrences in a λ-term M. We say that (ϕ,β) *occurs to the left of* (ϕ',β) if ϕ occurs to the left of ϕ'.

Definition 2.1.15. A rewrite sequence $\sigma : M_0 \xrightarrow{\phi_0}_\beta M_1 \xrightarrow{\phi_1}_\beta \ldots \xrightarrow{\phi_{m-1}}_\beta M_m$ is said to be a *standard β-rewrite sequence* if for every $n \in \{0,\ldots,m-1\}$ and for every $p \in \{0,\ldots,n-1\}$ it is not the case that the β-redex occurrence (ϕ_n,β) in M_n is a residual of a β-redex occurrence (ϕ'_n,β) in M_p that occurs to the left of the β-redex occurrence (ϕ_p,β).

Example 2.1.16. The rewrite sequence

$$(\lambda x.(\lambda y.y)x)z \xrightarrow{\epsilon}_\beta (\lambda y.y)z \xrightarrow{\epsilon}_\beta z$$

is a standard β-rewrite sequence. The rewrite sequence

$$(\lambda x.(\lambda y.y)x)z \xrightarrow{00}_\beta (\lambda x.x)z \xrightarrow{\epsilon}_\beta z$$

is not a standard β-rewrite sequence.

Now we state the standardisation theorem. It is proved at several places in the literature: by Curry and Feys in [CFC58], by Mitschke as reported in [Bar84], in two different ways by Klop in [Klo80] and finally by Takahashi in [Tak95].

Theorem 2.1.17. *For every β-rewrite sequence $\sigma : M \twoheadrightarrow_\beta M'$ there is a standard β-rewrite sequence $\tau : M \twoheadrightarrow_\beta M'$.*

In fact, there is a unique standard rewrite sequence τ such that σ and τ are permutation equivalent as defined by Lévy in [Lév78], see [Klo80] and [Bar84]. We won't need this in the sequel.

A corollary of the standardisation theorem is that the result of a computation of a λ-term, if it exists, can be found by contracting repeatedly the leftmost β-redex.

Definition 2.1.18.

1. Let M be a λ-term. A *leftmost β-redex occurrence in M* is a β-redex occurrence (ϕ, β) that occurs to the left of every other β-redex occurrence (ϕ', β) in M.

2. The *leftmost β-rewrite strategy*, denoted by F_{lm}, is defined as the mapping that assigns to each term not in β-normal form its leftmost β-redex occurrence.

3. A *leftmost β-rewrite sequence* is a F_{lm}-rewrite sequence.

We will use the following corollary of the standardisation theorem.

Corollary 2.1.19. *A λ-term has a β-normal form if and only if its leftmost β-rewrite sequence eventually ends in a β-normal form.*

Since not all λ-terms have a normal form, it is of interest to formulate a notion of partial result. Often one takes for a partial result a head normal form.

Definition 2.1.20.

1. A λ-term of the form $\lambda x_1 \ldots. \lambda x_m.yM_1 \ldots M_n$ or $\lambda x_1 \ldots. \lambda x_m.fM_1 \ldots M_n$, both with $m, n \geq 0$, is said to be *in head normal form.*
2. A λ-term is said to be *head normalising* if it can be rewritten to a head normal form.

A λ-term is either of the form $\lambda x_1 \ldots. \lambda x_m.yM_1 \ldots M_n$ or $\lambda x_1 \ldots. \lambda x_m.fM_1 \ldots M_n$, or of the form $\lambda x_1 \ldots. \lambda x_m.(\lambda y.P)M_1 \ldots M_n$, with $m, n \geq 0$. In the last case, the term is said to have a head redex occurrence .

Definition 2.1.21.

1. Let M be a λ-term. If $M = \lambda x_1 \ldots. \lambda x_m.(\lambda y.P)M_1 \ldots M_n$ for some $m, n \geq 0$, then $(0^{m+n-1}, \beta)$ is the *head redex occurrence* of M. Otherwise M does not have a head-redex.
2. The *head β-rewrite strategy*, denoted by $\mathsf{F_h}$, is defined as the mapping that assigns to each term that is not in head normal form its head redex occurrence.
3. A *head β-rewrite sequence* is a $\mathsf{F_h}$-rewrite sequence.

Note that a head redex is a leftmost redex but not vice versa. We will make use of yet another corollary of the standardisation theorem.

Corollary 2.1.22. *A λ-term has a head normal form if and only if its head β-rewrite sequence eventually ends in a head normal form.*

2.2 Three Characterisations

In this section we characterise strongly normalising λ-terms, weakly normalising λ-terms and head normalising λ-terms all by inductively defined sets. We start by giving a characterisation of the set of normal forms, also as an inductively defined set.

Definition 2.2.1. The set $\mathcal{NF}$ is the smallest set of λ-terms that satisfies the following:

1. if x is a variable and $M_1, \ldots, M_m \in \mathcal{NF}$ for some $m \geq 0$, then $xM_1 \ldots M_m \in \mathcal{NF}$,
2. if f is a constant and $M_1, \ldots, M_m \in \mathcal{NF}$ for some $m \geq 0$, then $fM_1 \ldots M_m \in \mathcal{NF}$,
3. if $M \in \mathcal{NF}$, then $\lambda x.M \in \mathcal{NF}$.

The following proposition is easy to prove.

Proposition 2.2.2. *A term M is in normal form if and only if $M \in \mathcal{NF}$.*

Strongly Normalising Lambda Terms. We characterise the strongly normalising λ-terms.

Definition 2.2.3. The set $\mathcal{SN}$ is the smallest set of λ-terms that satisfies the following:

1. if x is a variable and $M_1, \ldots, M_m \in \mathcal{SN}$ for some $m \geq 0$ then $xM_1 \ldots M_m \in \mathcal{SN}$,

2. if f is a constant and $M_1, \ldots, M_m \in \mathcal{SN}$ for some $m \geq 0$ then $fM_1 \ldots M_m \in \mathcal{SN}$,

3. if $M \in \mathcal{SN}$, then $\lambda x.M \in \mathcal{SN}$,

4. if $M[x := N]P_1 \ldots P_m \in \mathcal{SN}$ for some $m \geq 0$ and $N \in \mathcal{SN}$, then $(\lambda x.M)NP_1 \ldots P_m \in \mathcal{SN}$.

Note that every subterm of a term in $\mathcal{SN}$ is in $\mathcal{SN}$. Further, we have that $\mathcal{NF} \subseteq \mathcal{SN}$. We prove that the set $\mathcal{SN}$ characterises the strongly normalising λ-terms.

Theorem 2.2.4. *A λ-term M is strongly normalising if and only if $M \in \mathcal{SN}$.*

Proof. Suppose that M is a strongly β-normalising λ-term. Let $\mathsf{maxred}(M)$ denote the maximum length of a rewrite sequence starting in M and ending in the normal form of M. Note that $\mathsf{maxred}(M)$ is well-defined by König's Lemma. We prove by induction on $(\mathsf{maxred}(M), M)$, ordered by the lexicographic product of the usual ordering on $\mathbb{N}$ and the subterm ordering, that $M \in \mathcal{SN}$.

If $\mathsf{maxred}(M) = 0$, then M is a normal form. Then $M \in \mathcal{SN}$ since $\mathcal{NF} \subseteq \mathcal{SN}$.

Suppose $\mathsf{maxred}(M) > 0$. Let $M = \lambda x_1. \ldots \lambda x_m.PQ_1 \ldots Q_n$ with $m \geq 0$ and $n \geq 0$. Two cases are distinguished.

1. $P = y$ or $P = f$. Every reduct of M is of the form $\lambda x_1. \ldots \lambda x_m.yQ'_1 \ldots Q'_n$ or of the form $\lambda x_1. \ldots \lambda x_m.fQ'_1 \ldots Q'_n$, in both cases with Q'_p a reduct of Q_p for every $p \in \{1, \ldots, n\}$. By the induction hypothesis, $Q_1, \ldots, Q_n \in \mathcal{SN}$. If $M = \lambda x_1. \ldots \lambda x_m.yQ_1 \ldots Q_n$, then we have $M \in \mathcal{SN}$ by the first and the third clause of the definition of $\mathcal{SN}$. If $M = \lambda x_1. \ldots \lambda x_m.fQ_1 \ldots Q_n$ then we have $M \in \mathcal{SN}$ by the second and the third clause of the definition of $\mathcal{SN}$.

2. $P = \lambda y.P_0$. In that case, we have $M = \lambda x_1. \ldots \lambda x_m.(\lambda y.P_0)Q_1 \ldots Q_n \rightarrow_\beta \lambda x_1. \ldots \lambda x_m.P_0[y := Q_1]Q_2 \ldots Q_n$. By the induction hypothesis, we have that $\lambda x_1. \ldots \lambda x_m.P_0[y := Q_1]Q_2 \ldots Q_n \in \mathcal{SN}$. Also by the induction hypothesis, we have that $Q_1 \in \mathcal{SN}$. By the last clause of the definition of $\mathcal{SN}$, we have $M = \lambda x_1. \ldots \lambda x_m.(\lambda y.P_0)Q_1 \ldots Q_n \in \mathcal{SN}$.

For the converse, suppose that $M \in \mathcal{SN}$. We prove by induction on the derivation of $M \in \mathcal{SN}$ that M is strongly β-normalising.

1. If $M = xP_1 \dots P_m$ or $M = fP_1 \dots P_m$ with $P_1, \dots, P_m \in \mathcal{SN}$ for some $m \geq 0$, then the statement follows easily by the induction hypothesis.

2. If $M = \lambda x.P$ with $P \in \mathcal{SN}$, then the statement follows easily by induction hypothesis.

3. Finally, suppose $M = (\lambda x.P)Q_1Q_2 \dots Q_m$ with $P[x := Q_1]Q_2 \dots Q_m \in \mathcal{SN}$ and $Q_1 \in \mathcal{SN}$ for some $m \geq 1$. Consider an arbitrary rewrite sequence $\sigma : M = M_0 \rightarrow_\beta M_1 \rightarrow_\beta M_2 \rightarrow_\beta \dots$ starting in the λ-term M. There are two possibilities: the head redex of M is contracted in σ or not.

   In the first case, there is a rewrite step $M_n \rightarrow_\beta M_{n+1}$ in the rewrite sequence σ with $M_n = (\lambda x.P')Q'_1Q'_2 \dots Q'_m$ and $M_{n+1} = P'[x := Q'_1]Q'_2 \dots Q'_m$ such that $P \twoheadrightarrow_\beta P'$ and $Q_1 \twoheadrightarrow_\beta Q'_1, \dots, Q_m \twoheadrightarrow_\beta Q'_m$. Since by induction hypothesis $P[x := Q_1]Q_2 \dots Q_m$ is strongly β-normalising, we have that its reduct M_{n+1} is strongly β-normalising and hence σ is finite.

   In the second case, we have that all λ-terms in the rewrite sequence σ are of the form $(\lambda x.P')Q'_1Q'_2 \dots Q'_m$ with $P \twoheadrightarrow_\beta P'$ and $Q_1 \twoheadrightarrow_\beta Q'_1, \dots, Q_m \twoheadrightarrow_\beta Q'_m$. Since we have by the induction hypothesis $P[x := Q_1]Q_2 \dots Q_m$ is strongly β-normalising and Q_1 is strongly β-normalising, all terms in σ are strongly β-normalising. Hence σ is finite. □

Weakly Normalising Lambda Terms. We characterise the weakly normalising λ-terms.

Definition 2.2.5. The set $\mathcal{WN}$ is the smallest set of λ-terms satisfying the following:

1. if x is a variable and $M_1, \dots, M_m \in \mathcal{WN}$ for some $m \geq 0$ then $xM_1 \dots M_m \in \mathcal{WN}$,

2. if f is a constant and $M_1, \dots, M_m \in \mathcal{WN}$ for some $m \geq 0$ then $fM_1 \dots M_m \in \mathcal{WN}$,

3. if $M \in \mathcal{WN}$ then $\lambda x.M \in \mathcal{WN}$,

4. if $M[x := N]P_1 \dots P_m \in \mathcal{WN}$ for some $m \geq 0$, then $(\lambda x.M)NP_1 \dots P_m \in \mathcal{WN}$.

The difference with the definition of the set $\mathcal{SN}$ is in the last clause. We show that the set $\mathcal{WN}$ characterises the weakly normalising terms.

Theorem 2.2.6. *A λ-term M is weakly normalising if and only if $M \in \mathcal{WN}$.*

Proof. Suppose that M is weakly normalising. Let $\mathbf{l}(M)$ denote the length of the leftmost β-rewrite sequence from M to its normal form. We prove by induction on $(\mathbf{l}(M), M)$, ordered by the lexicographic product of the ordering on $\mathbb{N}$ and the subterm ordering, that $M \in \mathcal{WN}$. We have $M = \lambda x_1. \ldots \lambda x_m.PQ_1 \ldots Q_n$ for some $m, n \geq 0$. Two cases are distinguished.

1. $P = y$ or $P = f$. By induction hypothesis, $Q_1, \ldots, Q_n \in \mathcal{WN}$. Hence $M \in \mathcal{WN}$.
2. $P = \lambda y.P_0$. We have $M \rightarrow_\beta \lambda x_1. \ldots \lambda x_m.P_0[y := Q_1]Q_2 \ldots Q_n$. We have that $M_0' = P_0[y := Q_1]Q_2 \ldots Q_n$ is weakly normalising and $\mathbf{l}(M_0') < \mathbf{l}(M)$. Hence by the induction hypothesis $M_0' \in \mathcal{WN}$ and therefore $(\lambda y.P_0)Q_1 \ldots Q_n \in \mathcal{WN}$. We conclude that $M \in \mathcal{WN}$.

Suppose for the converse that $M \in \mathcal{WN}$. We prove by induction on the definition of $\mathcal{WN}$ that M is weakly normalising.

1. Suppose $M = xP_1 \ldots P_m$ or $M = fP_1 \ldots P_m$ with $P_1, \ldots, P_m \in \mathcal{WN}$ for some $m \geq 0$. The statement follows by the induction hypothesis.
2. Suppose $M = \lambda x.P$ with $P \in \mathcal{WN}$. By induction hypothesis P is weakly normalising, hence M is weakly normalising.
3. Suppose $M = (\lambda x.P)Q_1Q_2 \ldots Q_m$ with $P[x := Q_1]Q_2 \ldots Q_m \in \mathcal{WN}$ for some $m \geq 1$. Since $M \rightarrow_\beta P[x := Q_1]Q_2 \ldots Q_m$ and the latter term is by induction hypothesis weakly normalising, we have that M is weakly normalising. □

Head Normalising Lambda Terms. We characterise the head normalising λ-terms.

Definition 2.2.7. The set $\mathcal{HN}$ is the smallest set of λ-terms satisfying the following:

1. if x is a variable and if $M_1, \ldots, M_m \in \Lambda$ for some $m \geq 0$, then $xM_1 \ldots M_m \in \mathcal{HN}$,
2. if f is a constant and if $M_1, \ldots, M_m \in \Lambda$ for some $m \geq 0$, then $fM_1 \ldots M_m \in \mathcal{HN}$,
3. if $M \in \mathcal{HN}$ then $\lambda x.M \in \mathcal{HN}$,
4. if $M[x := N]P_1 \ldots P_m \in \mathcal{HN}$ for some $m \geq 0$, then $(\lambda x.M)NP_1 \ldots P_m \in \mathcal{HN}$.

The difference with the definition of the set $\mathcal{WN}$ is in the first two clauses.

Theorem 2.2.8. *A λ-term M is head normalising if and only if $M \in \mathcal{HN}$.*

Proof. Suppose M is head normalising. Let $\mathbf{h}(M)$ denote the length of the head β-rewrite sequence of M to its head normal form. We prove by induction on $(\mathbf{h}(M), M)$, lexicographically ordered by the usual ordering on $\mathbb{N}$ and the subterm ordering, that $M \in \mathcal{HN}$. We have $M = \lambda x_1 \ldots . \lambda x_m.PQ_1 \ldots Q_n$ for some $m, n \geq 0$. Two cases are distinguished.

1. $P = y$ or $P = f$. Then clearly $M \in \mathcal{HN}$.

2. $P = \lambda y.P_0$. Then $M \rightarrow_\beta \lambda x_1 \ldots . \lambda x_m.P_0[y := Q_1]Q_2 \ldots Q_n$. We have that $M_0' = P_0[y := Q_1]Q_2 \ldots Q_n$ is head normalising, and $\mathbf{h}(M_0') < \mathbf{h}(M)$. Therefore we have by the induction hypothesis that $M_0' \in \mathcal{HN}$, and hence $(\lambda y.P_0)Q_1 \ldots Q_n \in \mathcal{HN}$. We conclude that $M \in \mathcal{HN}$.

Suppose $M \in \mathcal{HN}$. We prove by induction on the definition of $\mathcal{HN}$ that M is head normalising.

1. Suppose $M = xP_1 \ldots P_m$ or $M = fP_1 \ldots P_m$. Then M is a head normal form.

2. Suppose $M = \lambda x.P$ with $P \in \mathcal{HN}$. By the induction hypothesis the term P has a head normal form, hence M has a head normal form.

3. Suppose $M = (\lambda x.P)Q_1Q_2 \ldots Q_m$ with $P[x := Q_1]Q_2 \ldots Q_m \in \mathcal{HN}$ for some $m \geq 1$. We have $M \rightarrow_\beta P[x := Q_1]Q_2 \ldots Q_m$ and by the induction hypothesis the latter term has a head normal from. Hence M has a head normal form. □

λI-calculus. We illustrate the use of the characterisations by presenting a simple proof of the theorem stating that every weakly normalising λI-term is strongly normalising. This result is obtained by Church in [Chu41] and is known as Church's Theorem. The set of λI-terms consists of λ-terms without abstractions $\lambda x.M$ such that $x \notin \mathsf{FV}(M)$. So the set of λI-terms is obtained by defining the set of λI terms as the set of λ-terms, with the difference that the third clause in Definition 2.1.1 is replaced by the following: if M is a λI-term and $x \in \mathsf{FV}(M)$, then $\lambda x.M$ is a λI-term. Moreover, the set of free variables of a λI-term must be defined simultaneously with the set of λI-terms. The following result is in fact an immediate consequence of the definition of $\mathcal{SN}$.

Theorem 2.2.9. *Let M be a λI-term. If M is weakly normalising, then M is strongly normalising.*

Proof. Let M be a λI-term with normal form M'. Let $M \twoheadrightarrow_\beta M'$ be the leftmost β-rewrite sequence of M to its normal form and let $\mathsf{l}(M)$ denote the length of this sequence. We prove by induction on $(\mathsf{l}(M), M)$, ordered by the lexicographic product of the usual ordering on $\mathbb{N}$ and the subterm ordering, that $M \in \mathcal{SN}$.

If $\mathsf{l}(M) = 0$, then M is a normal form and hence $M \in \mathcal{SN}$.

Suppose $\mathsf{l}(M) > 0$. We have $M = \lambda x_1. \ldots \lambda x_m.PQ_1 \ldots Q_n$ for some $m, n \geq 0$. Two cases are distinguished.

1. $P = y$ or $P = f$. The λ-terms $Q_1, \ldots, Q_n$ are weakly normalising. By induction hypothesis $Q_1, \ldots, Q_n \in \mathcal{SN}$. Hence $M \in \mathcal{SN}$.

2. $P = \lambda y.P_0$. Then

$$\begin{array}{lcl} M & = & \lambda x_1. \ldots \lambda x_m.(\lambda y.P_0)Q_1 \ldots Q_n \\ & \rightarrow_\beta & \lambda x_1. \ldots \lambda x_m.P_0[y := Q_1]Q_2 \ldots Q_n. \end{array}$$

   Call the latter term M'. We have that M' is weakly normalising. Since $\mathsf{l}(M) > \mathsf{l}(M')$, we have by the induction hypothesis that $M' \in \mathcal{SN}$. By definition of the set of λI-terms, we have that $y \in \mathsf{FV}(P_0)$. Hence Q_1 is a subterm of M', which yields that we also have that $Q_1 \in \mathcal{SN}$. We conclude that $(\lambda y.P_0)Q_1Q_2 \ldots Q_n \in \mathcal{SN}$ and hence $M \in \mathcal{SN}$. □

2.3 Finite Developments

A development is a rewrite sequence in which only residuals of redex occurrences that are present in the initial term are contracted. Developments are defined in an abstract setting in Section 1.2 of Chapter 1. In this section we consider developments in λ-calculus with β-reduction, called β-developments. A classical result in λ-calculus is that all β-developments are finite. In this section we present two new proofs of this result. The definition of a development is given as usual, making use of a set of underlined λ-terms with underlined β-reduction. The first proof makes use of a set $\mathcal{FD}$ that contains underlined terms that admit only finite $\underline{\beta}$-rewrite sequences. The definition of $\mathcal{FD}$ is in the same spirit as the definition of $\mathcal{SN}$. The second proof makes direct use of the set $\mathcal{SN}$. A β-development is translated into a β-rewrite sequence in the set $\mathcal{SN}$. This yields that all β-developments are finite. Before embarking on the proof, we recall the traditional definition of β-developments by means of underlined λ-terms. At the end of the section we discuss earlier proofs of finiteness of developments.

Developments. Traditionally, developments in λ-calculus with β-reduction are defined using underlined terms. The set of underlined terms is defined as follows.

Definition 2.3.1. The set of *underlined λ-terms*, denoted by $\underline{\Lambda}$, is the smallest set satisfying the following:

1. $x \in \underline{\Lambda}$ for every variable x,
2. $f \in \underline{\Lambda}$ for every constant f,
3. if $M \in \underline{\Lambda}$, then $\lambda x.M \in \underline{\Lambda}$,
4. if $M \in \underline{\Lambda}$ and $N \in \underline{\Lambda}$, then $MN \in \underline{\Lambda}$,
5. if $M \in \underline{\Lambda}$ and $N \in \underline{\Lambda}$, then $(\underline{\lambda} x.M)N \in \underline{\Lambda}$.

For underlined λ-terms M and N, the definitions of the set of positions of M, denoted by $\mathsf{Pos}(M)$, the subterm of M at position ϕ, denoted by $M|_\phi$, the substitution of N for x in M, denoted by $M[x := N]$ and the replacement of the subterm in M at position ϕ by N, denoted by $M[\phi \leftarrow N]$, can be given by minor variations on the definitions given in Section 2.1. We do not give these definitions explicitly. Note that a subterm of $\underline{\Lambda}$ is not necessarily in $\underline{\Lambda}$.

Definition 2.3.2.

1. Let $M \in \underline{\Lambda}$. A *$\underline{\beta}$-redex occurrence in M* is a pair $(\phi, \underline{\beta})$ such that $M|_\phi = (\underline{\lambda} x.P)Q$.
2. A *$\underline{\beta}$-redex* is a pair $(M, (\phi, \underline{\beta}))$ such that $(\phi, \underline{\beta})$ is a $\underline{\beta}$-redex occurrence in M.
3. A *$\underline{\beta}$-rewrite step* is a triple $(M, (\phi, \underline{\beta}), M')$ such that
   (a) $M|_\phi = (\underline{\lambda} x.P)Q$,
   (b) $M' = M[\phi \leftarrow P[x := Q]]$.

Notation 2.3.3. A $\underline{\beta}$-rewrite step $(M, (\phi, \underline{\beta}), M')$ is usually denoted by $(\phi, \underline{\beta}) : M \to M'$ or by $M \xrightarrow{\phi}_{\underline{\beta}} M'$.

Remark 2.3.4.

1. By induction on M, it follows that $M[x := N] \in \underline{\Lambda}$ if $M, N \in \underline{\Lambda}$.
2. If $M \in \underline{\Lambda}$ and $M \to_{\underline{\beta}} M'$, then $M' \in \underline{\Lambda}$. So $\underline{\Lambda}$ is closed under $\underline{\beta}$-rewriting.

Definition 2.3.5. The set denoted by $\mathfrak{U}_{\underline{\beta}}$ consists of all pairs of the form $(\phi, \underline{\beta})$ with ϕ a position. The set of all $\underline{\beta}$-redex occurrences in an underlined term M is denoted by $\mathfrak{U}_{\underline{\beta}}(M)$.

The rewriting system consisting of underlined β-terms with $\underline{\beta}$-reduction has an underlying abstract rewriting system of the form $(\underline{\Lambda}, \rightarrow_{\underline{\beta}})$ with $M \rightarrow_{\underline{\beta}} M'$ if there is a position $\phi \in \mathsf{Pos}(M)$ such that $(\phi, \underline{\beta}) : M \rightarrow M'$. The underlying functional rewriting system is $(\underline{\Lambda}, \mathfrak{U}_{\underline{\beta}}, \rightarrow)$ with $\rightarrow((\phi, \underline{\beta}))(M) = M'$ if $(\phi, \underline{\beta}) : M \rightarrow M'$ and $\rightarrow((\phi, \underline{\beta}))(M)$ being undefined otherwise.

In order to switch between λ-terms with β-reduction and underlined λ-terms with $\underline{\beta}$-reduction, we define a mapping $\mathcal{E} = (\mathcal{E}_0, \mathcal{E}_1)$ that erases underlinings.

Definition 2.3.6. The mapping

$$\mathcal{E} = (\mathcal{E}_0, \mathcal{E}_1) : (\underline{\Lambda}, \mathfrak{U}_{\underline{\beta}}, \rightarrow) \rightarrow (\Lambda, \mathfrak{U}_{\beta}, \rightarrow)$$

is defined as follows.

1. The mapping $\mathcal{E}_0 : \underline{\Lambda} \rightarrow \Lambda$ is defined by induction on the definition of $\underline{\Lambda}$ as follows:
   (a) $\mathcal{E}_0(x) = x$,
   (b) $\mathcal{E}_0(f) = f$,
   (c) $\mathcal{E}_0(\lambda x.M) = \lambda x.\mathcal{E}_0(M)$,
   (d) $\mathcal{E}_0(MN) = \mathcal{E}_0(M)\mathcal{E}_0(N)$,
   (e) $\mathcal{E}_0((\underline{\lambda} x.M)N) = (\lambda x.\mathcal{E}_0(M))\mathcal{E}_0(N)$.
2. The mapping $\mathcal{E}_1 : \mathfrak{U}_{\underline{\beta}} \rightarrow \mathfrak{U}_{\beta}$ is defined by $\mathcal{E}_1((\phi, \underline{\beta})) = (\phi, \beta)$.

Recall from Definition 1.2.6 in Section 1.2 of Chapter 1 that a morphism of functional rewriting systems is a mapping that preserves the rewrite relation in a precise way.

Proposition 2.3.7. *The mapping $\mathcal{E} = (\mathcal{E}_0, \mathcal{E}_1)$ is a morphism of functional rewriting systems.*

Proof. Suppose that we have a rewrite step $M \xrightarrow{\phi}_{\underline{\beta}} M'$. Observe that $\mathcal{E}_0(M)|_\phi = \mathcal{E}_0(M|_\phi)$. Hence we have $\mathcal{E}_0(M) \xrightarrow{\phi}_{\beta} \mathcal{E}_0(M')$. □

A rewrite sequence σ is a lifting of a rewrite sequence τ if we can obtain τ by applying a morphism to the rewrite sequence σ. The definition of a lifting has been given in Definition 1.2.7 of Chapter 1.

Definition 2.3.8. A β-rewrite sequence $\sigma : M \twoheadrightarrow_{\beta} N$ is a *β-development* if there is a $\underline{\beta}$-rewrite sequence τ in $(\underline{\Lambda}, \mathfrak{U}_{\underline{\beta}}, \rightarrow)$ that is an $\mathcal{E}$-lifting of σ.

Remark 2.3.9. This definition of β-development is equivalent to the definition of a development in a functional rewriting system as given in Definition 1.2.13 of Chapter 1.

First Proof of Finite Developments. In this section we show that all β-developments are finite using a set $\mathcal{FD}$ consisting of underlined λ-terms.

Definition 2.3.10. The set $\mathcal{FD}$ is the smallest set of underlined λ-terms that satisfies the following.

1. $x \in \mathcal{FD}$ for every variable x,
2. $f \in \mathcal{FD}$ for every constant f,
3. if $M \in \mathcal{FD}$, then $\lambda x.M \in \mathcal{FD}$,
4. if $M \in \mathcal{FD}$ and $N \in \mathcal{FD}$, then $MN \in \mathcal{FD}$,
5. if $M[x := N] \in \mathcal{FD}$ and $N \in \mathcal{FD}$, then $(\underline{\lambda} x.M)N \in \mathcal{FD}$.

Lemma 2.3.11. *Let $P \in \mathcal{FD}$ and $Q \in \mathcal{FD}$. Then $P[x := Q] \in \mathcal{FD}$.*

Proof. Induction on the derivation of $P \in \mathcal{FD}$. □

Proposition 2.3.12. $\underline{\Lambda} = \mathcal{FD}$.

Proof. Suppose $M \in \underline{\Lambda}$. We prove by induction on the definition of $\underline{\Lambda}$ that $M \in \mathcal{FD}$. The first four cases are immediate. If $M = (\underline{\lambda} x.P)Q$ with $P \in \underline{\Lambda}$ and $Q \in \underline{\Lambda}$, then by the induction hypothesis $P \in \mathcal{FD}$ and $Q \in \mathcal{FD}$. By Lemma 2.3.11 we have that $P[x := Q] \in \mathcal{FD}$. By the definition of $\mathcal{FD}$ we have that $M = (\underline{\lambda} x.P)Q \in \mathcal{FD}$.

Suppose $M \in \mathcal{FD}$. We prove by induction on the definition of $\mathcal{FD}$ that $M \in \underline{\Lambda}$. Again the first four cases are immediate. If $M = (\underline{\lambda} x.P)Q$ with $P[x := Q] \in \mathcal{FD}$ and $Q \in \mathcal{FD}$, then by the induction hypothesis, $P[x := Q] \in \underline{\Lambda}$ and $Q \in \underline{\Lambda}$. This yields that $P \in \underline{\Lambda}$, hence $M = (\underline{\lambda} x.P)Q \in \underline{\Lambda}$. □

This yields that β-developments are finite if and only if all $\underline{\beta}$-rewrite sequences in $\mathcal{FD}$ are finite. In Theorem 2.3.14 it is shown that all $\underline{\beta}$-rewrite sequences in $\mathcal{FD}$ are finite. The proof makes use of the following observation.

Remark 2.3.13. Let $M = PQ$ with $P, Q \in \mathcal{FD}$. If $M \twoheadrightarrow_{\underline{\beta}} M'$, then $M' = P'Q'$ with $P \twoheadrightarrow_{\underline{\beta}} P'$ and $Q \twoheadrightarrow_{\underline{\beta}} Q'$.

Theorem 2.3.14. *Let $M \in \mathcal{FD}$. Every $\underline{\beta}$-rewrite sequence starting in M is finite.*

Proof. The proof proceeds by induction on the derivation of $M \in \mathcal{FD}$.

1. If $M = x$ or $M = f$, then the statement clearly holds.

2. Let $M = \lambda x.P$ with $P \in \mathcal{FD}$, then by induction hypothesis P is strongly $\underline{\beta}$-normalising. Hence $M = \lambda x.P$ is strongly $\underline{\beta}$-normalising.

3. Let $M = PQ$ with $P \in \mathcal{FD}$ and $Q \in \mathcal{FD}$. Then by induction hypothesis both P and Q are strongly $\underline{\beta}$-normalising. By Remark 2.3.13 we have that if $M \twoheadrightarrow_{\underline{\beta}} M'$, then $M' = P'Q'$ with $P \twoheadrightarrow_{\underline{\beta}} P'$ and $Q \twoheadrightarrow_{\underline{\beta}} Q'$. Hence $M = PQ$ is strongly $\underline{\beta}$-normalising.

4. Let $M = (\underline{\lambda} x.P)Q$ with $P[x := Q] \in \mathcal{FD}$ and $Q \in \mathcal{FD}$. Consider an arbitrary rewrite sequence $\sigma : M = M_0 \to_{\underline{\beta}} M_1 \to_{\underline{\beta}} M_2 \to_{\underline{\beta}} \ldots$ starting in M. There are two possibilities: the head redex of M is contracted in σ or not.

   In the first case there is a rewrite step $M_m \to_{\underline{\beta}} M_{m+1}$ with $M_m = (\underline{\lambda} x.P')Q'$ and $M_{m+1} = P'[x := Q']$. Since by induction hypothesis $P[x := Q]$ is strongly $\underline{\beta}$-normalising, we have that its reduct $P'[x := Q']$ is strongly $\underline{\beta}$-normalising. Hence σ is finite.

   In the second case every underlined λ-term M_m in σ is of the form $(\underline{\lambda} x.P')Q'$ with $P \twoheadrightarrow_{\underline{\beta}} P'$ and $Q \twoheadrightarrow_{\underline{\beta}} Q'$. By induction hypothesis, Q is strongly $\underline{\beta}$-normalising and $P[x := Q]$ is strongly $\underline{\beta}$-normalising, hence P is strongly $\underline{\beta}$-normalising. We conclude that σ is finite. □

Corollary 2.3.15. *All β-developments are finite.*

Proof. By Proposition 2.3.12 and Theorem 2.3.14. □

Second Proof of Finite Developments. The second proof of finiteness of β-developments makes direct use of the set $\mathcal{SN}$. We define a morphism $\mathcal{F}$ that maps a $\underline{\beta}$-rewrite sequence in $\underline{\Lambda}$ to a β-rewrite sequence in $\mathcal{SN}$. This yields that all developments are finite.

We suppose that there is a distinguished constant denoted by Abs. The idea of the morphism is that underlinings are erased and that β-redexes that do not correspond to a $\underline{\beta}$-redex, are blocked by adding Abs in front of the λ.

To make $\mathcal{F}$ into a morphism we need that the positions where Abs occurs 'do not count'. To that end we need to modify the definition of the set of positions of a λ-term (Definition 2.1.6) and the definition of a subterm of a λ-term at a certain position (Definition 2.1.7). We will be concerned with λ-terms in which Abs occurs only in subterms of the form Abs$M$. Hence it is sufficient to add the following clause to the definition of the set of positions of a λ-term:

$$\mathsf{Pos}(\mathsf{Abs}M) = \mathsf{Pos}(M)$$

and the following clause to the definition of a λ-term at a certain position

$$(\mathsf{Abs}M)|_{\phi} = M|_{\phi}.$$

Now we define the mapping $\mathcal{F}$.

Definition 2.3.16. The mapping

$$\mathcal{F} = (\mathcal{F}_0, \mathcal{F}_1) : (\underline{\Lambda}, \mathfrak{U}_{\underline{\beta}}, \rightarrow) \rightarrow (\Lambda, \mathfrak{U}_{\beta}, \rightarrow)$$

is defined as follows.

1. The mapping $\mathcal{F}_0 : \underline{\Lambda} \rightarrow \mathcal{SN}$ is defined by induction on the definition of $\underline{\Lambda}$.
   (a) $\mathcal{F}_0(x) = x$,
   (b) $\mathcal{F}_0(f) = f$,
   (c) $\mathcal{F}_0(\lambda x.M) = \mathsf{Abs}\lambda x.\mathcal{F}_0(M)$,
   (d) $\mathcal{F}_0(MN) = \mathcal{F}_0(M)\mathcal{F}_0(N)$,
   (e) $\mathcal{F}_0((\underline{\lambda} x.M)N) = (\lambda x.\mathcal{F}_0(M))\mathcal{F}_0(N)$.
2. The mapping $\mathcal{F}_1 : \mathfrak{U}_{\underline{\beta}} \rightarrow \mathfrak{U}_{\beta}$ is defined by $\mathcal{F}_1((\phi, \underline{\beta})) = (\phi, \beta)$.

We first show that $\mathcal{F}$ is a morphism of functional rewriting systems and then that $\mathcal{F}$ maps an underlined λ-term to a λ-term in $\mathcal{SN}$.

Proposition 2.3.17. *The mapping $\mathcal{F} = (\mathcal{F}_0, \mathcal{F}_1)$ is a morphism of functional rewriting systems.*

Proof. We show two things:

1. if $M|_{\phi} = (\underline{\lambda} x.P)Q$ then $\mathcal{F}_0(M)|_{\phi} = (\lambda x.\mathcal{F}_0(P))\mathcal{F}_0(Q)$,
2. $\mathcal{F}_0(P[x := Q]) = \mathcal{F}_0(P)[x := \mathcal{F}_0(Q)]$.

1. We proceed by induction on ϕ. We treat only the most complicated case, which is when $M = \lambda y.M_0$ and $\phi = 0\,\phi_0$. Suppose $M|_{0\phi_0} = (\underline{\lambda} x.P)Q$, that is, $M_0|_{\phi_0} = (\underline{\lambda} x.P)Q$. By the induction hypothesis, we have that $\mathcal{F}_0(M)|_{\phi_0} = (\lambda x.\mathcal{F}_0(P))\mathcal{F}_0(Q)$. Using this, it is easy to see that $\mathcal{F}_0(M)|_{\phi} = (\lambda x.\mathcal{F}_0(P))\mathcal{F}_0(Q)$:

$$\begin{aligned} \mathcal{F}_0(M)|_{\phi} &= \\ \mathcal{F}_0(\lambda y.M_0)|_{0\phi_0} &= \\ (\mathsf{Abs}\lambda y.\mathcal{F}_0(M_0))|_{0\phi_0} &= \\ (\lambda y.\mathcal{F}_0(M_0))|_{0\phi_0} &= \\ \mathcal{F}_0(M_0)|_{\phi_0} &= \\ (\lambda x.\mathcal{F}_0(P))\mathcal{F}_0(Q). & \end{aligned}$$

2. By induction on the structure of P. $\square$

Lemma 2.3.18. *If $M \in \underline{\Lambda}$ then $\mathcal{F}_0(M) \in \mathcal{SN}$.*

Proof. By induction on the definition of $\underline{\Lambda}$ we prove the following simultaneously:

1. $\mathcal{F}_0(M) \in \mathcal{SN}$,
2. $\mathcal{F}_0(M)$ is not of the form $\lambda y.P$.

We proceed by induction on the definition of $\underline{\Lambda}$.

1. Suppose $M = x$ or $M = f$. Then $\mathcal{F}_0(M) = M$, so clearly $\mathcal{F}_0(M) \in \mathcal{SN}$ and $\mathcal{F}_0(M)$ is not of the form $\lambda y.P$.
2. Suppose $M = \lambda x.M_0$ with $M_0 \in \underline{\Lambda}$. Then $\mathcal{F}_0(M) = \mathsf{Abs}\lambda x.\mathcal{F}_0(M_0)$. By induction hypothesis, $\mathcal{F}_0(M_0) \in \mathcal{SN}$. Hence $\mathcal{F}_0(M) \in \mathcal{SN}$ and $\mathcal{F}_0(M)$ is not of the form $\lambda y.P$.
3. Suppose $M = M_0M_1$ with $M_0 \in \underline{\Lambda}$ and $M_1 \in \underline{\Lambda}$. Then we have $\mathcal{F}_0(M) = \mathcal{F}_0(M_0)\mathcal{F}_0(M_1)$. By the induction hypothesis, $\mathcal{F}_0(M_0), \mathcal{F}_0(M_1) \in \mathcal{SN}$ and $\mathcal{F}_0(M_0)$ is not of the form $\lambda y.P$. By induction on the derivation of $\mathcal{F}_0(M_0) \in \mathcal{SN}$, it then follows that $\mathcal{F}_0(M) \in \mathcal{SN}$: all cases are immediate since the case $\mathcal{F}_0(M_0) = \lambda y.P \in \mathcal{SN}$ doesn't occur. Clearly $\mathcal{F}_0(M)$ is not of the form $\lambda y.P$. $\square$

Theorem 2.3.19. *All β-developments are finite.*

Proof. By Proposition 2.3.17 and Lemma 2.3.18. $\square$

Related Work. Finiteness of β-developments is a classical result in λ-calculus and various proofs of it are known. Church and Rosser prove finiteness of developments for λI-calculus with β-reduction in [CR36]. The first proof for the full λ-calculus is given by Schroer in [Sch65]. Other proofs have been given by Hyland in [Hyl73] and by Hindley in [Hin78]. In [Bar84] a proof of finiteness of developments using a decreasing labelling is given. This proof is due to Barendregt, Bergstra, Klop and Volken and is also reported in [BBKV76]. A short and elegant proof using a perpetual strategy is given by de Vrijer in [Vri85]. This proof gives as extra information an exact bound on the length of a development. An encoding of developments in rewrite sequences in λ-calculus with intersection types is given by Parigot in [Par90]. Since λ-calculus with intersection types is known to be strongly normalising, this yields finiteness of developments. This proof is also reported in [Kri93]. It is also possible to obtain a proof of finiteness of developments by encoding a development as a rewrite sequence in simply typed λ-calculus. This is done in [OR94a] (see also [OR93]) and in [Ghi94]. The proofs using an encoding of developments in a strongly normalising calculus are similar in spirit to the second proof presented in this section. An encoding of developments in a λ-calculus with memory as in the work by Nederpelt [Ned73] and Klop [Klo80] is given by van Oostrom. Melliès gives an axiomatic proof of finiteness of developments that applies also to the case of λ-calculus with β-reduction. This proof is reported in [Mel96].

2.4 Finite Superdevelopments

In this section we study β-rewrite sequences called superdevelopments. Superdevelopments form a generalisation of developments. Whereas in a development only redex occurrences are contracted that are residuals of redex occurrences in the initial term, in a superdevelopment one may moreover contract redex occurrences that are created by rewriting, provided that they are created in a particular way. Lévy has analysed in [Lév78] the ways in which β-redex occurrences can be created. It appears that there are the following three ways of creating β-redexes:

1. $((\lambda x.\lambda y.M)N)P \rightarrow_\beta (\lambda y.M[x := N])P$,
2. $(\lambda x.x)(\lambda y.M)N \rightarrow_\beta (\lambda y.M)N$,
3. $(\lambda x.M)(\lambda y.N) \rightarrow_\beta M[x := \lambda y.N]$ if there is a position $\phi \in \mathsf{Pos}(M)$ such that $M|_\phi = xM_0$, for some $M_0 \in \Lambda$.

Note that in the last case we have for every position ϕ such that $M|_\phi = xM_0$, that $(M[x := \lambda y.N])|_\phi = (\lambda y.N)M_0'$, with $M_0' = M_0[x := \lambda y.N]$. So at every such position ϕ a β-redex occurrence is created. In the first two ways of creating a β-redex occurrence, on can say that the creation is 'upwards', whereas in the last case it can said to be 'downwards'. A superdevelopment will be defined as a β-rewrite sequence in which besides redex occurrences that are residuals of redex occurrences in the initial term, also redex occurrences may be contracted that are created in either the first or the second way as explained above. We will show that all superdevelopments are finite. Note that in the rewrite sequence $\Omega \rightarrow_\beta \Omega \rightarrow_\beta \Omega \rightarrow_\beta \ldots$ with $\Omega = (\lambda x.xx)(\lambda x.xx)$, only the third kind of redex creation occurs.

Confluence of λ-calculus with β-reduction can be shown using a notion of parallel reduction. This proof is due to Tait and Martin-Löf. A parallel reduction step corresponds to a complete development of a set of β-redex occurrences performed in an inside-out way. The essential clause in the definition of a parallel β-reduction step, denoted by $\multimap\!\!\rightarrow$, is: $(\lambda x.M)N \multimap\!\!\rightarrow M'[x := N']$ if $M \multimap\!\!\rightarrow M'$ and $N \multimap\!\!\rightarrow N'$. There is a complete β-development of M to N if and only if there is a parallel reduction step from M to N. Note that this does not yield that all developments are finite. For a more elaborate discussion of parallel reduction we refer to Section 5.1 of Chapter 5. Superdevelopments correspond to a stronger notion of parallel reduction, due to Aczel [Acz78]. The essential clause in the definition of a parallel β-reduction step à la Aczel is the following: $MN \multimap M'[x := N']$ if $M \multimap \lambda x.M'$ and $N \multimap N'$. A superdevelopment is defined such that there is a complete β-superdevelopment of M to N if and only if there is a parallel reduction step à la Aczel from M to N.

This section is organised as follows. First we define superdevelopments as in [Raa93] by means of a labelled λ-calculus. It is shown that superdevelopments are finite, making use of a set denoted by $\mathcal{FSD}$ that plays the same rôle as the set $\mathcal{FD}$ in the first proof of finiteness of developments given in the previous section. Then we show in another way that superdevelopments are finite, namely by mapping a superdevelopment to a rewrite sequence in the set $\mathcal{SN}$. The mapping is a morphism of functional rewriting systems. This yields that superdevelopments are finite, since all λ-terms in $\mathcal{SN}$ are strongly normalising.

Superdevelopments. In the following we define a labelled λ-calculus such that a labelled rewrite sequence corresponds, after erasing the labels, to a rewrite sequence in which besides redex occurrences that are residuals of redex occurrences in the initial term, only redex occurrences are contracted that are created in the first or the second way as explained above. Consider the first way of creating a β-redex. Already before performing the rewrite step $((\lambda x.\lambda y.M)N)P \rightarrow_\beta (\lambda y.M[x := N])P$ creating the β-redex occurrence $(\lambda y.M[x := N])P$, there was a relationship between the application and the λ of the created β-redex: the λ was already in the scope of the application, that is, in the subterm with the application at the root. The same holds for the second way of creating a β-redex, although in this case the creation is less 'upwards' in nature, since the λ of the created β-redex occurrence switches from one branch to another in the tree representation of the term.

We now formalise this as follows. Initially, λ's are labelled by different natural numbers, and applications are either labelled by a natural number or are not labelled. The initial labelling must be such that a λ can only have the same label as an application it is in the scope of that application. The labels control the rewrite relation in the following way: $(\lambda_p x.PQ)^p \rightarrow_{\beta_l} P[x := Q]$. In that case, the λ with label p was initially already in the scope of the application with label p.

Labelled λ-terms are built from variables, constants, labelled λ-abstraction and labelled application as specified in the following definition.

Definition 2.4.1. The set of *labelled λ-terms*, denoted by Λ_l, is defined as the smallest set that satisfies the following:

1. $x \in \Lambda_l$ for every variable x,

2. $f \in \Lambda_l$ for every constant f,
3. if $M \in \Lambda_l$ and $p \in \mathbb{N}$, then $\lambda_p x.M \in \Lambda_l$,
4. if $M, N \in \Lambda_l$ and $p \in \mathbb{N}$, then $(MN)^p \in \Lambda_l$.

We define labelled β-reduction on the set of labelled λ-terms as follows.

Definition 2.4.2.

1. Let $M \in \Lambda_l$. A *β_l-redex occurrence in M* is a pair (ϕ, β_l) such that $M|_\phi = ((\lambda_p x.P)Q)^p$.
2. A *β_l-redex* is a pair $(M, (\phi, \beta_l))$ such that (ϕ, β_l) is a β_l-redex occurrence in M.
3. A *β_l-rewrite step* is a triple $(M, (\phi, \beta_l), M')$ such that
   (a) $M|_\phi = ((\lambda_p x.P)Q)^p$,
   (b) $M' = M[\phi \leftarrow P[x := Q]]$.

Notation 2.4.3. A β_l-rewrite step $(M, (\phi, \beta_l), M')$ is usually denoted by $(\phi, \beta_l) : M \rightarrow M'$ or by $M \xrightarrow{\phi}_{\beta_l} M'$.

Definition 2.4.4. The set consisting of all pairs of the form (ϕ, β_l) is denoted by $\mathfrak{U}_{\beta_l}$. The set of all β_l-redex occurrences in a labelled term M is denoted by $\mathfrak{U}_{\beta_l}(M)$.

In order to define superdevelopments we will restrict attention to terms that are labelled such that the label of an application cannot be equal to the label of a λ that is not in its scope.

Definition 2.4.5. Let $M \in \Lambda_l$.

1. A term $M \in \Lambda_l$ is said to be *well-labelled* if the following holds. If we have
   (a) $M|_\phi = (P_0 P_1)^p$,
   (b) $M|_\chi = \lambda_p x.Q$,

   then $\phi \preceq \chi$. The set of well-labelled λ-terms is denoted by Λ_l^w.
2. A term M is said to be *initially labelled* if
   (a) M is well-labelled,
   (b) if $M|_\phi = \lambda_p x.P$ and $M|_\chi = \lambda_p x'.P'$ then $\phi = \chi$.

Remark 2.4.6. It is not difficult to see that the set Λ_l^w is closed under β_l-reduction.

In the remainder of this section we will suppose all labelled terms to be well-labelled.

The main property of β_l-reduction is that no redexes are created 'by substitution'.

Remark 2.4.7. Let $M \to M'$ be a β_l-rewrite step of the form $((\lambda_p x.P)Q)^p \to P[x := Q]$. Then every β_l-rewrite step in any reduct of M' takes place either in a descendant of P or in a descendant of Q. This is the case since any other β_l-redex occurrence would consist of an application in a descendant of P and a λ in a descendant of Q. But in that case the label of the λ is not equal to the label of the application, since M is supposed to be well-labelled.

We define a mapping from labelled λ-terms with labelled β-reduction to λ-terms with β-reduction. We will show that this mapping is a morphism of functional rewriting systems as in Definition 1.2.6 of Chapter 1.

Definition 2.4.8. The mapping

$$\mathcal{E} = (\mathcal{E}_0, \mathcal{E}_1) : (\Lambda_l^w, \mathfrak{U}_{\beta_l}, \to) \to (\Lambda, \mathfrak{U}_\beta, \to)$$

is defined as follows.

1. The mapping $\mathcal{E}_0 : \Lambda_l \to \Lambda$ is defined by induction on the definition of Λ.
   (a) $\mathcal{E}_0(x) = x$,
   (b) $\mathcal{E}_0(f) = f$,
   (c) $\mathcal{E}_0(\lambda_p x.M) = \lambda x.\mathcal{E}_0(M)$,
   (d) $\mathcal{E}_0((MN)^p) = \mathcal{E}_0(M)\mathcal{E}_0(N)$.
2. The mapping $\mathcal{E}_1 : \mathfrak{U}_{\beta_l} \to \mathfrak{U}_\beta$ is defined by $\mathcal{E}_1((\phi, \beta_l)) = (\phi, \beta)$.

Proposition 2.4.9. *The mapping $\mathcal{E} = (\mathcal{E}_0, \mathcal{E}_1)$ is a morphism of functional rewriting systems.*

Proof. Suppose that $M|_\phi = ((\lambda_p x.P)Q)^p$. Then we have $(\mathcal{E}_0(M))|_\phi = \mathcal{E}_0(M|_\phi) = (\lambda x.\mathcal{E}_0(P))\mathcal{E}_0(Q)$. This yields that $\mathcal{E}_0(M) \xrightarrow{\mathcal{E}_1(\phi)}_\beta \mathcal{E}_0(M')$ if $M \xrightarrow{\phi}_{\beta_l} M'$. □

Recall the definition of a lifting from Definition 1.2.7 in Chapter 1. We define a superdevelopment as a β-rewrite sequence in Λ that can be lifted to a β_l-rewrite sequence in Λ_l.

Definition 2.4.10. A β-rewrite sequence $\sigma : M \twoheadrightarrow_\beta N$ is a *β-superdevelopment* if there is a rewrite sequence τ that starts in an initially labelled term and that is an $\mathcal{E}$-lifting of σ.

Example 2.4.11.

1. The β-rewrite sequence
$$\sigma : (\lambda x.\lambda y.xy)zz' \rightarrow_\beta (\lambda y.zy)z' \rightarrow_\beta zz'$$
is a superdevelopment, since the following β_l-rewrite sequence is an $\mathcal{E}$-lifting of σ:
$$(((\lambda_1 x.\lambda_2 y.xy)z)^1 z')^2 \rightarrow_{\beta_l} ((\lambda_2 y.zy)z')^2 \rightarrow_{\beta_l} zz'.$$

2. The rewrite sequence
$$\sigma' : (\lambda x.x)(\lambda y.y)z \rightarrow_\beta (\lambda y.y)z \rightarrow_\beta z$$
is a superdevelopment, since the following β_l-rewrite sequence is an $\mathcal{E}$-lifting of σ':
$$(((\lambda_1 x.x)\lambda_2 y.y)^1 z)^2 \rightarrow_{\beta_l} ((\lambda_2 y.y)z)^2 \rightarrow_{\beta_l} z.$$

3. The rewrite sequence
$$(\lambda x.xx)(\lambda x.xx) \rightarrow_\beta (\lambda x.xx)(\lambda x.xx) \rightarrow_\beta (\lambda x.xx)(\lambda x.xx)$$
is not a superdevelopment.

Note that σ and σ' are not developments.

First Proof of Finite Superdevelopments. In this section we show that all β-superdevelopments terminate. We define a rewriting system consisting of a set $\mathcal{FSD}$ of underlined terms and a rewrite relation $\rightarrow_{\underline{\beta}}$. A β_l-rewrite sequence in Λ_l is mapped to a $\underline{\beta}$-rewrite sequence in $\mathcal{FSD}$, preserving rewrite steps. We show that the rewrite relation $\rightarrow_{\underline{\beta}}$ on $\mathcal{FSD}$ is strongly normalising by an easy induction. This yields strong normalisation of $\rightarrow_{\beta_l}$ on Λ_l, and hence finiteness of superdevelopments. The set $\mathcal{FSD}$ is similar to the set $\mathcal{FD}$ used in the first proof of finiteness of developments presented in the previous section.

Definition 2.4.12. The set $\mathcal{FSD}$ is defined as the smallest set of underlined λ-terms that satisfies the following.

1. $x \in \mathcal{FSD}$ for every variable x,

2. $f \in \mathcal{FSD}$ for every constant f,

3. if $M \in \mathcal{FSD}$ then $\lambda x.M \in \mathcal{FSD}$,
4. if $M \in \mathcal{FSD}$ and $N \in \mathcal{FSD}$ then $MN \in \mathcal{FSD}$,
5. if $M[x := N]P_1 \dots P_m \in \mathcal{FSD}$ for some $m \geq 0$ and $N \in \mathcal{FSD}$, then $(\underline{\lambda} x.M)NP_1 \dots P_m \in \mathcal{FSD}$,
6. if $(\underline{\lambda} x.M)N_0 \dots N_m \in \mathcal{FSD}$ for some $m \geq 0$, then $(\underline{\lambda} y.y)(\underline{\lambda} x.M)N_0 \dots N_m \in \mathcal{FSD}$.

The definition of a $\underline{\beta}$-redex occurrence, a $\underline{\beta}$-redex and a $\underline{\beta}$-rewrite step are as given in Definition 2.3.2 of the previous section. The difference, of course, is that this time we consider $\underline{\beta}$-reduction on the set $\mathcal{FSD}$. Further, we denote by $\mathfrak{U}_{\underline{\beta}}$ the set of pairs of the form $(\phi, \underline{\beta})$. An easy observation concerning $\underline{\beta}$-reduction on $\mathcal{FSD}$ is the following. It is a consequence of the fact that $\mathcal{FSD}$ does not contain underlined λ-terms of the form $\underline{\lambda} x.M$.

Remark 2.4.13. Let $M = PQ$ with $P \in \mathcal{FSD}$ and $Q \in \mathcal{FSD}$. If $M \twoheadrightarrow_{\underline{\beta}} M'$, then $M' = P'Q'$ with $P \twoheadrightarrow_{\underline{\beta}} P'$ and $Q \twoheadrightarrow_{\underline{\beta}} Q'$.

The following lemma shows that the set $\mathcal{FSD}$ is closed under substitution.

Lemma 2.4.14. *Let $M \in \mathcal{FSD}$ and $N \in \mathcal{FSD}$. Then $M[x := N] \in \mathcal{FSD}$.*

Proof. The proof proceeds by induction on the derivation of $M \in \mathcal{FSD}$.

1. If $M = y$ or $M = f$, then the statement follows immediately.
2. Suppose $M = \lambda y.M_0$ with $M_0 \in \mathcal{FSD}$. By the induction hypothesis, we have that $M_0[x := N] \in \mathcal{FSD}$. Hence $M[x := N] = \lambda y.(M_0[x := N]) \in \mathcal{FSD}$.
3. Suppose $M = M_0 M_1$ with $M_0 \in \mathcal{FSD}$ and $M_1 \in \mathcal{FSD}$. By the induction hypothesis, we have that $M_0[x := N] \in \mathcal{FSD}$ and $M_1[x := N] \in \mathcal{FSD}$. Hence $M[x := N] \in \mathcal{FSD}$.
4. Suppose $M = (\underline{\lambda} y.M_0)M_1 M_2 \dots M_m$ with $(M_0[y := M_1])M_2 \dots M_m \in \mathcal{FSD}$, $M_1 \in \mathcal{FSD}$ and $m \geq 1$. By the induction hypothesis, we have $(M_0[y := M_1]M_2 \dots M_m)[x := N] \in \mathcal{FSD}$ and $M_1[x := N] \in \mathcal{FSD}$. This yields that $M[x := N] \in \mathcal{FSD}$.
5. Suppose $M = (\underline{\lambda} z.z)(\underline{\lambda} y.M_0)M_1 \dots M_m$ with $(\underline{\lambda} y.M_0)M_1 \dots M_m \in \mathcal{FSD}$ and $m \geq 1$. By the induction hypothesis, we have that $((\underline{\lambda} y.M_0)M_1 \dots M_m)[x := N] \in \mathcal{FSD}$ and this yields that $M[x := N] \in \mathcal{FSD}$. $\square$

We now define a mapping $\mathcal{F}$ from Λ_l^w with β_l-reduction to $\mathcal{FSD}$ with $\underline{\beta}$-reduction. It is shown that $\mathcal{F}$ is a morphism of functional rewrite sequences. This means that in order to prove finiteness of superdevelopments it is sufficient to prove that $\underline{\beta}$-rewriting on $\mathcal{FSD}$ is strongly normalising, which follows by an easy induction. The definition of $\mathcal{F}$ makes use of the following definition, that makes use of a descendant relation for the set $\mathcal{FSD}$ with $\underline{\beta}$-reduction. It has not been defined explicitly but is just a small variation on the definition of the descendant relation for λ-calculus that has been given in Definition 2.1.13.

Definition 2.4.15. Let $P \in \mathcal{FSD}$. We define a underlined λ-term P^* as follows. If $P \twoheadrightarrow_{\underline{\beta}} \lambda x.P'$, then the term P^* is defined as P where the unique symbol λ that descends to the head-lambda in $\lambda x.P'$ is underlined. Otherwise $P^* = P$.

Note that not necessarily $P^* \in \mathcal{FSD}$. Using the previous definition we now define the mapping $\mathcal{F}$ and show that it is a morphism of functional rewriting systems from $(\Lambda_l^w, \mathfrak{U}_{\beta_l}, \rightarrow)$ to $(\mathcal{FSD}, \mathfrak{U}_{\underline{\beta}}, \rightarrow)$.

Definition 2.4.16. The mapping

$$\mathcal{F} = (\mathcal{F}_0, \mathcal{F}_1) : (\Lambda_l^w, \mathfrak{U}_{\beta_l}, \rightarrow) \rightarrow (\mathcal{FSD}, \mathfrak{U}_{\underline{\beta}}, \rightarrow)$$

is defined as follows.

1. The mapping $\mathcal{F}_0 : \Lambda_l^w \rightarrow \mathcal{FSD}$ is defined by induction on the definition of Λ_l.

   (a) $\mathcal{F}_0(x) = x$,

   (b) $\mathcal{F}_0(f) = f$,

   (c) $\mathcal{F}_0(\lambda_p x.P) = \lambda x.\mathcal{F}_0(P)$,

   (d) $\mathcal{F}_0((PQ)^p) = \begin{cases} \mathcal{F}_0(P)^*\mathcal{F}_0(Q) & \text{if } P \twoheadrightarrow_{\beta_l} \lambda_p x.P_0, \\ \mathcal{F}_0(P)\mathcal{F}_0(Q) & \text{otherwise.} \end{cases}$

2. The mapping $\mathcal{F}_1 : \mathfrak{U}_{\beta_l} \rightarrow \mathfrak{U}_{\underline{\beta}}$ is defined by $\mathcal{F}_1(\phi) = \phi$.

We show that the mapping $\mathcal{F}$ is a morphism. For the proof we make use of Lemma 2.4.14 and of the following lemma.

Lemma 2.4.17. *Let $P \in \mathcal{FSD}$ with $P \twoheadrightarrow_{\underline{\beta}} \lambda x.P'$. Let $Q \in \mathcal{FSD}$. Then $P^*Q \in \mathcal{FSD}$.*

Proof. The proof proceeds by induction on the derivation of $P \in \mathcal{FSD}$.

1. Suppose $P = \lambda x.P_0$ with $P_0 \in \mathcal{FSD}$. By Lemma 2.4.14 we have $P_0[x := Q] \in \mathcal{FSD}$. Hence $P^*Q = (\underline{\lambda} x.P_0)Q \in \mathcal{FSD}$.

2. Suppose $P = (\underline{\lambda} y.P_0)P_1 \dots P_m$ with $P_0[y := P_1]P_2 \dots P_m \in \mathcal{FSD}$ and $P_1 \in \mathcal{FSD}$. We have $P_0[y := P_1]P_2 \dots P_m \twoheadrightarrow_{\underline{\beta}} \lambda x.P'$. By induction hypothesis, we have $(P_0[y := P_1]P_2 \dots P_m)^* Q \in \mathcal{FSD}$. Hence $P^* Q \in \mathcal{FSD}$.

3. Suppose $P = (\underline{\lambda} y.y)(\underline{\lambda} z.P_0)P_1 \dots P_m$ with $(\underline{\lambda} z.P_0)P_1 \dots P_m \in \mathcal{FSD}$. We have that $(\underline{\lambda} z.P_0)P_1 \dots P_m \twoheadrightarrow_{\underline{\beta}} \lambda x.P'$. By induction hypothesis, we have $((\underline{\lambda} z.P_0)P_1 \dots P_m)^* Q \in \mathcal{FSD}$. Hence $P^* Q \in \mathcal{FSD}$. □

Theorem 2.4.18. *The mapping $\mathcal{F}$ is a morphism of functional rewriting systems.*

Proof. We prove two things: for all $M \in \Lambda_l^w$ we have $\mathcal{F}_0(M) \in \mathcal{FSD}$, and if $M \xrightarrow{\phi_0}_{\beta_l} \dots \xrightarrow{\phi_m}_{\beta_l} M'$, then $\mathcal{F}_0(M) \xrightarrow{\phi_0}_{\underline{\beta}} \dots \xrightarrow{\phi_m}_{\underline{\beta}} \mathcal{F}_0(M')$. Both statements are proved simultaneously by induction on the definition of Λ_l.

1. Suppose $M = x$ or $M = f$. We have $\mathcal{F}_0(M) \in \mathcal{FSD}$. Since there are no β_l-rewrite sequences issuing from M the second statement holds.

2. Suppose $M = \lambda_p x.M$. Then $\mathcal{F}_0(M) = \lambda x.\mathcal{F}_0(P)$. By the induction hypothesis $\mathcal{F}_0(P) \in \mathcal{FSD}$, hence $\mathcal{F}_0(M) \in \mathcal{FSD}$. Since a β_l-rewrite sequence starting in M is of the form $\lambda_p x.P \xrightarrow{\phi}_{\beta_l} \lambda_p x.P' \xrightarrow{\phi'}_{\beta_l} \lambda_p x.P'' \xrightarrow{\phi''}_{\beta_l} \dots$, the second statement holds by the induction hypothesis.

3. Suppose $M = (PQ)^p$. Two cases are distinguished.

   (a) Suppose $M \twoheadrightarrow_{\beta_l} \lambda_p x.P_0$. By the induction hypothesis, $\mathcal{F}_0(P) \in \mathcal{FSD}$ and $\mathcal{F}_0(Q) \in \mathcal{FSD}$. By the induction hypothesis, we have moreover $\mathcal{F}_0(P) \twoheadrightarrow_{\underline{\beta}} \lambda x.\mathcal{F}_0(P_0)$. By Lemma 2.4.17, $\mathcal{F}_0(M) = \mathcal{F}_0(P)^* \mathcal{F}_0(Q) \in \mathcal{FSD}$. Using the observation of Remark 2.4.7 we also have that the second statement holds.

   (b) Otherwise, we have that $\mathcal{F}_0(M) = \mathcal{F}_0(P)\mathcal{F}_0(Q)$. By the induction hypothesis, $\mathcal{F}_0(P) \in \mathcal{FSD}$ and $\mathcal{F}_0(Q) \in \mathcal{FSD}$. Hence $\mathcal{F}_0(M) \in \mathcal{FSD}$. Further, every β_l-rewrite sequence starting in M is of the form $(PQ)^p \twoheadrightarrow_{\beta_l} (P'Q')^p$ with $P \twoheadrightarrow_{\beta_l} P'$ and $Q \twoheadrightarrow_{\beta_l} Q'$. Hence the second statement holds. □

So every β_l-rewrite sequence in Λ_l corresponds in a precise way to a $\underline{\beta}$-rewrite sequence in $\mathcal{FSD}$. It is also the case that every $\underline{\beta}$-rewrite sequence corresponds to a β_l-rewrite sequence in a precise way, but since we don't need this for the proof of finiteness of superdevelopments we do not give the proof here. Now we prove that all $\underline{\beta}$-rewrite sequence in $\mathcal{FSD}$ are finite.

Theorem 2.4.19. *Let $M \in \mathcal{FSD}$. Every $\underline{\beta}$-rewrite sequence starting in M is finite.*

Proof. The proof proceeds by induction on the derivation of $M \in \mathcal{FSD}$.

1. If M is a variable or a constant then clearly every $\underline{\beta}$-rewrite sequence starting in M is finite.

2. If $M = \lambda x.P$ with $P \in \mathcal{FSD}$, then by the induction hypothesis P is strongly $\underline{\beta}$-normalising. Hence $M = \lambda x.P$ is strongly $\underline{\beta}$-normalising.

3. If $M = PQ$ with $P \in \mathcal{FSD}$ and $Q \in \mathcal{FSD}$, then by the induction hypothesis both P and Q are strongly $\underline{\beta}$-normalising. Let σ be an arbitrary $\underline{\beta}$-rewrite sequence starting in M. By Remark 2.4.13 every term in σ is of the form $P'Q'$ with $P \twoheadrightarrow_{\underline{\beta}} P'$ and $Q \twoheadrightarrow_{\underline{\beta}} Q'$. Hence σ is finite.

4. Let $M = (\underline{\lambda} x.P)Q_1Q_2 \ldots Q_m$ with $P[x := Q_1]Q_2 \ldots Q_m \in \mathcal{FSD}$ and $Q_1 \in \mathcal{FSD}$. Consider an arbitrary $\underline{\beta}$-rewrite sequence $\sigma : M = M_0 \rightarrow_{\underline{\beta}} M_1 \rightarrow_{\underline{\beta}} M_2 \rightarrow_{\underline{\beta}} \ldots$ starting in M. There are two possibilities: the head redex of M is contracted in σ or it is not contracted.

   In the first case there is a rewrite step $M_n \rightarrow_{\underline{\beta}} M_{n+1}$ in σ with $M_n = (\underline{\lambda} x.P')Q'_1Q'_2 \ldots Q'_m$ and $M_{n+1} = P'[x := Q'_1]Q'_2 \ldots Q'_m$ such that $P \twoheadrightarrow_{\underline{\beta}} P', Q_1 \twoheadrightarrow_{\underline{\beta}} Q'_1, \ldots, Q_m \twoheadrightarrow_{\underline{\beta}} Q'_m$. By the induction hypothesis, we have that $P[x := Q_1]Q_2 \ldots Q_m$ is strongly $\underline{\beta}$-normalising. Hence its $\underline{\beta}$-reduct $P'[x := Q'_1]Q'_2 \ldots Q'_m$ is strongly $\underline{\beta}$-normalising and σ is finite.

   In the second case all terms in σ are of the form $(\underline{\lambda} x.P')Q'_1Q'_2 \ldots Q'_m$ with $P \twoheadrightarrow_{\underline{\beta}} P', Q_1 \twoheadrightarrow_{\underline{\beta}} Q'_1, \ldots Q_m \twoheadrightarrow_{\underline{\beta}} Q'_m$. By induction hypothesis, $P[x := Q_1]Q_2 \ldots Q_m$ is strongly $\underline{\beta}$-normalising, so $P, Q_2, \ldots, Q_m$ are strongly $\underline{\beta}$-normalising. Moreover, we have by the induction hypothesis that Q_1 is strongly $\underline{\beta}$-normalising. Hence every term in σ is strongly $\underline{\beta}$-normalising and σ is finite.

5. Let $M = (\underline{\lambda} y.y)(\underline{\lambda} x.P)Q_1Q_2 \ldots Q_m$ with $(\underline{\lambda} x.P)Q_1Q_2 \ldots Q_m \in \mathcal{FSD}$. Consider an arbitrary $\underline{\beta}$-rewrite sequence $\sigma : M = M_0 \rightarrow_{\underline{\beta}} M_1 \rightarrow_{\underline{\beta}} M_2 \rightarrow_{\underline{\beta}} \ldots$ starting in M. There are two possibilities: the head redex of M is contracted in σ or it is not contracted.

   In the first case, there is a rewrite step $M_n \rightarrow_{\underline{\beta}} M_{n+1}$ in σ with $M_n = (\underline{\lambda} y.y)(\underline{\lambda} x.P')Q'_1Q'_2 \ldots Q'_m$ and $M_{n+1} = (\underline{\lambda} x.P')Q'_1Q'_2 \ldots Q'_m$ and such that $P \twoheadrightarrow_{\underline{\beta}} P', Q_1 \twoheadrightarrow_{\underline{\beta}} Q'_1, \ldots, Q_m \twoheadrightarrow_{\underline{\beta}} Q'_m$. By the induction hypothesis, we have that $(\underline{\lambda} x.P)Q_1Q_2 \ldots Q_m$ is strongly $\underline{\beta}$-normalising, and hence we have that its reduct $(\underline{\lambda} x.P')Q'_1Q'_2 \ldots Q'_m$ is also strongly $\underline{\beta}$-normalising. This yields that $(\underline{\lambda} y.y)(\underline{\lambda} x.P')Q'_1Q'_2 \ldots Q'_m$ is strongly $\underline{\beta}$-normalising and σ is finite.

   In the second case, all terms in σ are of the form $(\underline{\lambda} y.y)(\underline{\lambda} x.P')Q'_1Q'_2 \ldots Q'_m$. By the induction hypothesis, $(\underline{\lambda} x.P')Q'_1Q'_2 \ldots Q'_m$ is strongly $\underline{\beta}$-normalising. This yields that σ is finite. □

Corollary 2.4.20. *All β-superdevelopments are finite.*

Proof. By Theorem 2.4.18 and Theorem 2.4.19. □

Second Proof of Finite Superdevelopments. The second proof of finiteness of superdevelopments is similar to the second proof of finiteness of developments. We define a morphism $\mathcal{F}$ that maps a β_l-rewrite sequence in Λ_l^w to a β-rewrite sequence in $\mathcal{SN}$. This yields that all superdevelopments are finite.

In this case we suppose that there is a distinguished constant denoted by App. The idea of the morphism is exactly the same as in the case of finiteness of developments: labels are erased and β-redexes that do not correspond to a β_l-redex are blocked by adding App in front.

As in the case of the proof of finiteness of developments, to make $\mathcal{F}$ into a morphism we need that the positions of App 'do not count'. We will be concerned with λ-terms in which App occurs only in subterms of the form App$M$. So it suffices to add the following clause to the definition of the set of positions of a λ-term:

$$\mathsf{Pos}(\mathsf{App}M) = \mathsf{Pos}(M)$$

and the following clause to the definition of a subterm of a λ-term at a certain position:

$$(\mathsf{App}M)|_\phi = M|_\phi.$$

Definition 2.4.21. The mapping

$$\mathcal{F} = (\mathcal{F}_0, \mathcal{F}_1) : (\Lambda_l^w, \mathfrak{U}_{\beta_l}, \to) \to (\Lambda, \mathfrak{U}_\beta, \to)$$

is defined as follows.

1. The mapping $\mathcal{F}_0 : \Lambda_l^w \to \Lambda$ is defined by induction on the definition of Λ_l.
   (a) $\mathcal{F}_0(x) = x$,
   (b) $\mathcal{F}_0(f) = f$,
   (c) $\mathcal{F}_0(\lambda_p x.M) = \lambda x.\mathcal{F}_0(M)$,
   (d) $\mathcal{F}_0((MN)^p) = \begin{cases} \mathcal{F}_0(M)\mathcal{F}_0(N) & \text{if } M \twoheadrightarrow_{\beta_l} \lambda_p x.M_0, \\ \mathsf{App}\,\mathcal{F}_0(M)\mathcal{F}_0(N) & \text{otherwise.} \end{cases}$
2. The mapping $\mathcal{F}_1 : \mathfrak{U}_{\beta_l} \to \mathfrak{U}_\beta$ is defined by $\mathcal{F}_1((\phi, \beta_l)) = (\phi, \beta)$.

Note that $\mathcal{F}_0(M)$ is never of the form $xM_1 \ldots M_m$ with $m > 0$. So we won't treat this possibility in the following proofs by induction.

Proposition 2.4.22. *The mapping $\mathcal{F}$ is a morphism of functional rewriting systems.*

Proof. We show two things:

1. if $M|_\phi = ((\lambda_p x.P)Q)^p$ then $\mathcal{F}_0(M)|_\phi = (\lambda x.\mathcal{F}_0(P))\mathcal{F}_0(Q)$,
2. $\mathcal{F}_0(P[x := Q]) = \mathcal{F}_0(P)[x := \mathcal{F}_0(Q)]$.

The first point is proved by induction on ϕ. We only treat the difficult case, which is when $\phi = 0\,\phi_0$ and $M = (M_0M_1)^p$ and it is not the case that $M_0 \twoheadrightarrow_{\beta_l} \lambda_p.P_0$. In that case $\mathcal{F}_0(M) = \mathsf{App}\,\mathcal{F}_0(M_0)\mathcal{F}_0(M_1)$. By the induction hypothesis, we have that if $M_0|_{\phi_0} = ((\lambda_p x.P)Q)^p$, then $\mathcal{F}_0(M_0)|_{\phi_0} = (\lambda x.\mathcal{F}_0(P))\mathcal{F}_0(Q)$. Now it follows easily that $\mathcal{F}_0(M)|_\phi = (\lambda x.\mathcal{F}_0(P))Q$:

$$\begin{array}{rl}
\mathcal{F}_0(M)|_{0\,\phi_0} & = \\
\mathcal{F}_0((M_0M_1)^p)|_{0\,\phi_0} & = \\
\mathsf{App}\mathcal{F}_0(M_0)\mathcal{F}_0(M_1)|_{0\,\phi_0} & = \\
\mathsf{App}\mathcal{F}_0(M_0)|_{\phi_0} & = \\
\mathcal{F}_0(M_0)|_{\phi_0} & = \\
(\lambda x.\mathcal{F}_0(P)\mathcal{F}_0(Q). &
\end{array}$$

The second point follows by induction on the structure of P, supposing the labels in P and in Q to be distinct. □

We now show that $\mathcal{F}$ maps labelled terms to terms in $\mathcal{SN}$. We need two auxiliary lemmata.

Lemma 2.4.23. *Let $M \in \Lambda_l$ and suppose $\mathcal{F}_0(M) \in \mathcal{SN}$. Let $N \in \mathcal{SN}$. Then $\mathcal{F}_0(M)[x := N] \in \mathcal{SN}$.*

Proof. The proof proceeds by induction on the derivation of $\mathcal{F}_0(M) \in \mathcal{SN}$.

1. Suppose $\mathcal{F}_0(M) = fP_1 \ldots P_m$ with $P_1 \in \mathcal{SN}, \ldots, P_m \in \mathcal{SN}$ for some $m \geq 0$. If $m = 0$, then the statement clearly holds. If $m > 0$, then $f = \mathsf{App}$. By the induction hypothesis we have $P_1[x := N] \in \mathcal{SN}, \ldots, P_m[x := N] \in \mathcal{SN}$. Hence $\mathcal{F}_0(M)[x := N] = fP_1[x := N] \ldots P_m[x := N] \in \mathcal{SN}$.

2. Suppose $\mathcal{F}_0(M) = \lambda y.P$ with $P \in \mathcal{SN}$. By the induction hypothesis $P[x := N] \in \mathcal{SN}$. Hence $\mathcal{F}_0(M)[x := N] = \lambda y.(P[x := N]) \in \mathcal{SN}$.

3. Suppose $\mathcal{F}_0(M) = (\lambda y.P)Q_0Q_1 \ldots Q_m$ with $P[y := Q_0]Q_1 \ldots Q_m \in \mathcal{SN}$ and $Q_0 \in \mathcal{SN}$. By the induction hypothesis we have $(P[y := Q_0]Q_1 \ldots Q_m)[x := N] \in \mathcal{SN}$ and $Q_0[x := N] \in \mathcal{SN}$. Hence we have $\mathcal{F}_0(M)[x := N] = ((\lambda y.P)Q_0Q_1 \ldots Q_m)[x := N] \in \mathcal{SN}$. □

Lemma 2.4.24. *Let $M \in \Lambda_I$ and suppose $\mathcal{F}_0(M) \in \mathcal{SN}$. Let $N \in \mathcal{SN}$. Then $\mathcal{F}_0(M)N \in \mathcal{SN}$.*

Proof. The proof proceeds by induction on the derivation of $\mathcal{F}_0(M) \in \mathcal{SN}$.

1. Suppose $\mathcal{F}_0(M) = fP_1 \dots P_m$ with $P_1 \in \mathcal{SN}, \dots, P_m \in \mathcal{SN}$. Then we have $\mathcal{F}_0(M)N = fP_1 \dots P_m N \in \mathcal{SN}$.

2. Suppose $\mathcal{F}_0(M) = \lambda x.P$ with $P \in \mathcal{SN}$. Then $M = \lambda_p x.M_0$ with $\mathcal{F}_0(M_0) = P$. By Lemma 2.4.23 we have $P[x := N] \in \mathcal{SN}$. So $\mathcal{F}_0(M)N = (\lambda x.P)N \in \mathcal{SN}$.

3. Suppose $\mathcal{F}_0(M) = (\lambda x.P)Q_0Q_1 \dots Q_m$ with $P[x := Q_0]Q_1 \dots Q_m \in \mathcal{SN}$ and $Q_0 \in \mathcal{SN}$. By the induction hypothesis $P[x := Q_0]Q_1 \dots Q_m N \in \mathcal{SN}$. Hence $\mathcal{F}_0(M)N = (\lambda x.P)Q_0Q_1 \dots Q_m N \in \mathcal{SN}$. □

Theorem 2.4.25. *If $M \in \Lambda_I^w$, then $\mathcal{F}_0(M) \in \mathcal{SN}$.*

Proof. The proof proceeds by induction on the definition of Λ_I^w.

1. Suppose $M = x$ or $M = f$ then $\mathcal{F}_0(M) \in \mathcal{SN}$.

2. Suppose $M = \lambda_p x.M_0$. By induction hypothesis, $\mathcal{F}_0(M_0) \in \mathcal{SN}$. Hence $\mathcal{F}_0(M) = \lambda x.\mathcal{F}_0(M_0) \in \mathcal{SN}$.

3. Suppose $M = (M_0M_1)^p$. By the induction hypothesis, $\mathcal{F}_0(M_0) \in \mathcal{SN}$ and $\mathcal{F}_0(M_1) \in \mathcal{SN}$. There are two possibilities.

   If $M_0 \twoheadrightarrow_{\beta_I} \lambda_p x.P$, then $\mathcal{F}_0(M) = \mathcal{F}_0(M_0)\mathcal{F}_0(M_1)$. By Lemma 2.4.24, we have that $\mathcal{F}_0(M) \in \mathcal{SN}$.

   In the other case, $\mathcal{F}_0(M) = \mathsf{App}\mathcal{F}_0(M_0)\mathcal{F}_0(M_1)$. Hence $\mathcal{F}_0(M) \in \mathcal{SN}$. □

Corollary 2.4.26. *All β-superdevelopments are finite.*

Proof. By Proposition 2.4.22 and Theorem 2.4.25. □

In [Raa93] a different proof of finiteness of superdevelopments is given. It is a proof by contradiction that makes use of a minimal counterexample.

2.5 Simply Typed Lambda Calculus

In this section we give a new proof of strong normalisation of simply typed λ-calculus with β-reduction. The proof makes use of the set $\mathcal{SN}$ given in Definition 2.2.3.

Simply Typed Lambda Calculus. We consider simply typed λ-calculus à la Church. First we recall the definition of the *simple types*. We suppose there is a set of *base types*. Base types are denoted by $0, 0', \ldots$. The alphabet of the simple types consists of the set of base types and a binary operator denoted by $\rightarrow$.

Definition 2.5.1. The set Types of *simple types* is defined as the smallest set satisfying the following:

1. $0 \in \mathsf{Types}$ for every base type 0,
2. if $A \in \mathsf{Types}$ and $B \in \mathsf{Types}$ then $A \rightarrow B \in \mathsf{Types}$.

Simple types are denoted by $A, B, C, \ldots$.

We use association to the right, so for instance $A \rightarrow B \rightarrow C$ stands for $A \rightarrow (B \rightarrow C)$. In Chapter 4 we will make use of the notion of arity of a simple type, which is defined as follows.

Definition 2.5.2. The *arity* of a simple type A, denoted by $\mathsf{ar}(A)$, is defined by induction on the structure of A:

1. $\mathsf{ar}(0) = 0$,
2. $\mathsf{ar}(A_0 \rightarrow A_1) = \mathsf{ar}(A_1) + 1$.

We suppose that for every simple type A there is a set Var_A consisting of infinitely many *variables of type A*. The set of all variables is denoted by Var and is defined as

$$\mathsf{Var} = \cup_{A \in \mathsf{Types}} \mathsf{Var}_A.$$

We suppose that for every simple type A there is a set C_A consisting of infinitely many *constants of type A*. The set of all constants is denoted by C and is defined as

$$\mathsf{C} = \cup_{A \in \mathsf{Types}} \mathsf{C}_A.$$

The alphabet of the terms of simply typed λ-calculus consists of the set Var of simply typed variables, the set C of simply typed constants, and two binary operators: one for λ-abstraction and one for application.

Definition 2.5.3. For every simple type A, the set of *simply typed λ-terms of type A*, denoted by $\Lambda^{\rightarrow}_A$, is defined as the smallest set that satisfies the following:

1. if x is a variable of type A, then $x \in \Lambda^{\rightarrow}_A$,
2. if f is a constant of type A, then $f \in \Lambda^{\rightarrow}_A$,

3. if $A = A_0 \to A_1$, $M \in \Lambda^{\to}_{A_1}$ and x is a variable of type A_0, then $\lambda x.M \in \Lambda^{\to}_{A}$,
4. if $M \in \Lambda^{\to}_{B\to A}$ and $N \in \Lambda^{\to}_{B}$, then $MN \in \Lambda^{\to}_{A}$.

Definition 2.5.4. The set of simply typed λ-terms, denoted by $\Lambda^{\to}$, is defined as

$$\Lambda^{\to} = \cup_{A\in\mathsf{Types}}\Lambda^{\to}_{A}.$$

Notation 2.5.5. We will also write $M : A$ to denote that $M \in \Lambda^{\to}_{A}$.

Definition 2.5.6. Let $X \subseteq \Lambda$ and $Y \subseteq \Lambda$. The set denoted by $X \to Y$ is defined by $X \to Y = \{M \in \Lambda \mid \forall N \in X : MN \in Y\}$.

We define the set of terms of type A that are strongly normalising. Note that a simply typed λ-term is also an untyped λ-term.

Definition 2.5.7. $\mathcal{SN}(A) = \{M \in \Lambda^{\to}_{A} \mid M \in \mathcal{SN}\}$.

Remark 2.5.8. Note that for every type A we have $\Lambda^{\to}_{A} \neq \emptyset$ and $\mathcal{SN}(A) \neq \emptyset$.

We will make use of the following classical result, called the Substitution Lemma. For a proof, see for instance [Bar92] for the case of pure type systems.

Lemma 2.5.9. *If $P : B$, $x : A$ and $N : A$ then $P[x := N] : B$.*

Strong Normalisation.

Proposition 2.5.10. $\Lambda^{\to}_{A\to B} = \Lambda^{\to}_{A} \to \Lambda^{\to}_{B}$.

Proof. Let $M \in \Lambda^{\to}_{A\to B}$. We clearly have for every $N \in \Lambda^{\to}_{A}$ that $MN \in \Lambda^{\to}_{B}$. Hence $\Lambda^{\to}_{A\to B} \subseteq \Lambda^{\to}_{A} \to \Lambda^{\to}_{B}$. For the converse, let $M \in \Lambda^{\to}_{A} \to \Lambda^{\to}_{B}$. Let $N \in \Lambda^{\to}_{A}$. Then we have $MN \in \Lambda^{\to}_{B}$. This yields that $M \in \Lambda^{\to}_{A\to B}$. Hence $\Lambda^{\to}_{A} \to \Lambda^{\to}_{B} \subseteq \Lambda^{\to}_{A\to B}$. □

Lemma 2.5.11. $\mathcal{SN}(A \to B) \supseteq \mathcal{SN}(A) \to \mathcal{SN}(B)$.

Proof. Let $M \in \mathcal{SN}(A) \to \mathcal{SN}(B)$. For $N \in \mathcal{SN}(A)$, we have $MN \in \mathcal{SN}(B)$ so $MN \in \mathcal{SN}$ and hence $M \in \mathcal{SN}$. Moreover $M \in \Lambda^{\to}_{A\to B}$, since $N \in \mathcal{SN}(A)$ and $MN \in \Lambda^{\to}_{B}$. We conclude that $M \in \mathcal{SN}(A \to B)$. □

The converse of the statement of the previous lemma is more difficult to prove. We need the following lemma.

Lemma 2.5.12. *Let $N \in \mathcal{SN}(A_1) \to \ldots \to \mathcal{SN}(A_m)$ with A_m a base type. Let $P \in \mathcal{SN}(B)$ and $x : A_1 \to \ldots \to A_m$. Then $P[x := N] \in \mathcal{SN}(B)$.*

Proof. The proof proceeds by induction on the derivation of $P \in \mathcal{SN}$.

1. Suppose $P = yP_1 \dots P_n$ with $P_1, \dots, P_n \in \mathcal{SN}$. We have $y : B_1 \to \dots \to B_n \to B$ and $P_1 : B_1, \dots, P_n : B_n$. By Lemma 2.5.9, we have that $P_p[x := N] : B_p$ for every $p \in \{1, \dots, n\}$. Together with the induction hypothesis, this yields that $P_1[x := N] \in \mathcal{SN}(B_1), \dots, P_n[x := N] \in \mathcal{SN}(B_n)$. Now two cases are distinguished.

   If $y \neq x$, then $P[x := N] = yP_1[x := N] \dots P_n[x := N]$. We have $P[x := N] \in \mathcal{SN}$, since $P_1[x := N] \in \mathcal{SN}, \dots, P_n[x := N] \in \mathcal{SN}$. Hence $P[x := N] \in \mathcal{SN}(B)$.

   If $y = x$, then $B_1 = A_1, \dots, B_n = A_n$ and $B = A_{n+1} \to \dots \to A_m$. By applying Lemma 2.5.11, we find that $\mathcal{SN}(A_{n+1}) \to \dots \to \mathcal{SN}(A_m) \subseteq \mathcal{SN}(B)$. This yields that $\mathcal{SN}(A_1) \to \dots \to \mathcal{SN}(A_m) \subseteq \mathcal{SN}(A_1) \to \dots \to \mathcal{SN}(A_n) \to \mathcal{SN}(B)$. So $N \in \mathcal{SN}(A_1) \to \dots \to \mathcal{SN}(A_n) \to \mathcal{SN}(B)$. Since $P_1[x := N] \in \mathcal{SN}(A_1), \dots, P_n[x := N] \in \mathcal{SN}(A_n)$, we conclude that $P[x := N] = N(P_1[x := N]) \dots (P_n[x := N]) \in \mathcal{SN}(B)$.

2. Suppose $P = fP_1 \dots P_n$ with $P_1 \in \mathcal{SN}, \dots, P_n \in \mathcal{SN}$. By reasoning in the same way as in the case $y \neq x$ above, we come to the conclusion that $P[x := N] \in \mathcal{SN}(B)$.

3. Suppose $P = \lambda y.P_0$ with $P_0 \in \mathcal{SN}$. By induction hypothesis we have $P_0[x := N] \in \mathcal{SN}$. Using moreover Lemma 2.5.9 we have $P[x := N] = \lambda y.(P_0[x := N]) \in \mathcal{SN}(B)$.

4. Suppose $P = (\lambda y.P_0)P_1P_2 \dots P_n$ with $P_0[y := P_1]P_2 \dots P_n \in \mathcal{SN}$ and $P_1 \in \mathcal{SN}$. By the induction hypothesis we have $(P_0[y := P_1]P_2 \dots P_n)[x := N] \in \mathcal{SN}$ and $P_1[x := N] \in \mathcal{SN}$. Together with Lemma 2.5.9 this yields $P[x := N] = ((\lambda y.P_0)P_1P_2 \dots P_n)[x := N] \in \mathcal{SN}(B)$. □

Now we can prove the following crucial lemma.

Lemma 2.5.13. $\mathcal{SN}(A \to B) \subseteq \mathcal{SN}(A) \to \mathcal{SN}(B)$.

Proof. Let $M \in \mathcal{SN}(A \to B)$. Let $N \in \mathcal{SN}(A)$. We need to prove that $MN \in \mathcal{SN}(B)$. We have $MN : B$. It remains to show that $MN \in \mathcal{SN}$. This is proved by induction on A, and for every A by induction on the derivation of $M \in \mathcal{SN}$.

1. Suppose $M = yP_1 \dots P_n$ with $P_1, \dots, P_n \in \mathcal{SN}$. Since $N \in \mathcal{SN}$, we have $MN = yP_1 \dots P_nN \in \mathcal{SN}$.

2. Suppose $M = fP_1 \dots P_n$ with $P_1, \dots, P_n \in \mathcal{SN}$. Since $N \in \mathcal{SN}$, we have $MN = fP_1 \dots P_nN \in \mathcal{SN}$.

3. Suppose $M = \lambda x.P$ with $P \in \mathcal{SN}$. We have $P \in \mathcal{SN}(B)$. We need to prove that $P[x := N] \in \mathcal{SN}$. We have $A = A_1 \to \ldots \to A_m$ with A_m a base type. By the induction hypothesis of the induction on the structure of A we have $N \in \mathcal{SN}(A_1) \to \ldots \to \mathcal{SN}(A_m)$. Lemma 2.5.12 yields that $P[x := N] \in \mathcal{SN}$.

4. Suppose $M = (\lambda y.P_0)P_1P_2 \ldots P_n$ with $P_0[y := P_1]P_2 \ldots P_n \in \mathcal{SN}$ and $P_1 \in \mathcal{SN}$. By the induction hypothesis of the induction on the derivation of $M \in \mathcal{SN}$, we have $P_0[y := P_1]P_2 \ldots P_n N \in \mathcal{SN}$. Since moreover $P_1 \in \mathcal{SN}$, we conclude that $MN = (\lambda y.P_0)P_1P_2 \ldots P_n N \in \mathcal{SN}$. □

Theorem 2.5.14. $\mathcal{SN}(A \to B) = \mathcal{SN}(A) \to \mathcal{SN}(B)$.

Proof. By Lemma 2.5.11 and Lemma 2.5.13. □

The proof of strong normalisation of simply typed λ-calculus with β-reduction is now a matter of simple induction.

Theorem 2.5.15. *Let A be a simple type. If $M \in \Lambda^{\to}_{A}$ then $M \in \mathcal{SN}(A)$.*

Proof. The proof proceeds by induction on $M \in \Lambda^{\to}_{A}$.

1. Suppose $M = x$ or $M = f$. Clearly $M \in \mathcal{SN}(A)$.

2. Suppose $M = \lambda x.P$ and $A = A_0 \to A_1$. By the induction hypothesis we have $P \in \mathcal{SN}(A_1)$ so $P \in \mathcal{SN}$. This yields that $M = \lambda x.P \in \mathcal{SN}$ and hence $M \in \mathcal{SN}(A)$.

3. Suppose $M = PQ$ with $P : B \to A$ and $Q : B$. By the induction hypothesis we have $P \in \mathcal{SN}(B \to A)$ and $Q \in \mathcal{SN}(B)$. By Theorem 2.5.14, $P \in \mathcal{SN}(B) \to \mathcal{SN}(A)$. This yields that $PQ \in \mathcal{SN}(A)$. □

Related Work. A proof of strong normalisation of the system Gödel's T, which is an extension of simply typed λ-calculus with primitive recursion, is given by Tait in [Tai67]. The proof makes use of the notion of computability and is quite short but complex. Girard introduces in [Gir72] the notion of candidate of reducibility. He extends Tait's method in order to prove strong normalisation of system F (second order λ-calculus) and $\lambda\omega$. In [Gan80], Gandy proves strong normalisation using a notion of strict monotonic functional. Van de Pol discusses in [Pol96] the relationship between the proof by Gandy and the proof by Tait. De Vrijer gives in [Vri87b], which is also published as [Vri87a], a proof of strong normalisation that makes use of the notion of strict monotonic functional and of a perpetual strategy. In this proof an exact bound on the length of a rewrite sequence issuing from a term is computed. A proof of strong normalisation of simply typed λ-calculus using an induction on a triple, consisting of the size of the type, the maximal length of a rewrite sequence and the size of the term, is due to van Daalen, see [NGV94, p.507]. Lévy applied this technique in [Lév78] to prove strong normalisation of a labelled λ-calculus with a bounded predicate. This proof yields also that all developments are finite, and standardisation, as reported in [Cur95]. Klop shows strong normalisation of labelled λ-calculus by an interpretation in λI-calculus where there is no erasure of subterms. Recently, Loader announced a proof of strong normalisation of system F on the 'types mailing list' [Loa95], which makes also use of the set of strongly normalising λ-terms. The proof presented in this section is mostly related to the proof by van Daalen and Lévy, but is more tailor-made for simply typed λ-calculus.

Chapter 3

Normalisation in Proof Nets

Intuitionistic linear logic was introduced by Girard in [Gir87] as a refinement of intuitionistic logic, obtained by deleting the structural rules for weakening and contraction. Weakening and contraction are then reintroduced, but this time as logical rules. There are many embeddings of intuitionistic logic in linear logic that are faithful with respect to provability. One such embedding has as characteristic property that an intuitionistic implication $A \to B$ is translated into a linear implication $!A^* \multimap B^*$, with A^* and B^* the translations of A and B.

In [Gir87], classical linear logic is obtained from intuitionistic linear logic by adding an operator for linear negation, and enforcing a complete symmetry: a formula $A^{\perp\perp}$ is identified with A and every other operator has its dual.

Proofs nets are graphical representations of derivations in sequent calculus of classical linear logic. Derivations in sequent calculus that are the same up to an irrelevant permutation, are represented by the same proof net. Thus the advantage of proof nets above derivations in sequent calculus is that we abstract over these irrelevant permutations.

In this chapter we are concerned with proof nets defined for the multiplicative-exponential fragment of classical linear logic. Rewriting of proof nets consists of cut-elimination. We present a proof of strong normalisation of proof nets with cut-elimination. It does not make use of a notion of candidates of reducibility as the one presented in [Gir87], but is more similar to the one presented by Joinet in [Joi93]. Simply typed λ-terms can be represented as proof nets such that one β-rewrite step corresponds to one or more steps of cut-elimination. The first such representations are studied by Danos [Dan90] and Régnier [Rég92]. We briefly discuss a translation of simply typed λ-calculus into proof nets.

This chapter is organised as follows. In Section 3.1 we recall the multiplicative-exponential fragment of classical linear logic. In Section 3.2 the syntax of the rewriting system consisting of proof nets with cut-elimination is presented. In Section 3.3 we present a proof of strong normalisation of this rewriting system.

3.1 Multiplicative-Exponential Linear Logic

In this section the sequent calculus of the multiplicative-exponential fragment of classical linear logic is presented.

Formulae. We suppose there is a set denoted by Atoms consisting of infinitely many *atomic formulae*. Atomic formulae are denoted by $p, q, \ldots$. For every atom $p \in$ Atoms there is a negative form $p' \in$ Atoms, called the *linear negation* of p. So the set Atoms is partitioned in positive and negative atomic formulae.

The alphabet of linear logic consists of the atomic formulae in Atoms, a binary operator called *times* denoted by $\otimes$, a binary operator called *par* denoted by $\wp$, a unary operator called *of course* denoted by !, and a unary operator called *why not* denoted by ?. We do not consider units for $\otimes$ and $\wp$.

The set of formulae is defined as follows.

Definition 3.1.1. The set Form of *formulae* is the smallest set satisfying the following:

1. if $p \in$ Atoms, then $p \in$ Form,
2. if $A \in$ Form and $B \in$ Form, then $A \otimes B \in$ Form,
3. if $A \in$ Form and $B \in$ Form, then $A \wp B \in$ Form,
4. if $A \in$ Form, then $!A \in$ Form,
5. if $A \in$ Form, then $?A \in$ Form.

The *linear negation* of a formula A, denoted by $A^\perp$, is defined by De Morgan equations:

$$\begin{array}{rclcrclcrcl} (p)^\perp & = & p', & & (A \otimes B)^\perp & = & A^\perp \wp B^\perp, & & (!A)^\perp & = & ?A^\perp, \\ p'^\perp & = & p, & & (A \wp B)^\perp & = & A^\perp \otimes B^\perp, & & (?A)^\perp & = & !A^\perp. \end{array}$$

In the sequel we write $p^\perp$ instead of p'. *Linear implication*, denoted by $\multimap$, is defined by

$$A \multimap B = A^\perp \wp B.$$

An *environment* is a multiset of formulae. We will denote an environment by $A_1, \ldots, A_m$, where it is to be understood that a formula may occur more than once and that the order is irrelevant.

Notation 3.1.2. We abbreviate $A_1, \ldots, A_m$ by $\vec{A}$. Further, if $\vec{A}$ abbreviates $A_1, \ldots, A_m$, then $?\vec{A}$ abbreviates $?A_1, \ldots, ?A_m$, $!\vec{A}$ abbreviates $!A_1, \ldots, !A_m$ and $\vec{A}^\perp$ abbreviates $A_1^\perp, \ldots, A_m^\perp$. If we use $\vec{A}$ and $\vec{B}$, then $\vec{A}$ abbreviates $A_1, \ldots, A_m$ and $\vec{B}$ abbreviates $B_1, \ldots, B_n$ with possibly $m \neq n$.

Sequent Calculus. The sequent calculus for classical multiplicative-exponential linear logic is given in the following table. We consider sequents of the form $\vdash \vec{A}$. A sequent of the form $\vec{A} \vdash \vec{B}$ can be represented as $\vdash \vec{A}^{\perp}, \vec{B}$.

$\vdash A, A^{\perp}$	(axiom)	$\dfrac{\vdash \vec{C}, A \quad \vdash A^{\perp}, \vec{D}}{\vdash \vec{C}, \vec{D}}$	(cut)
$\dfrac{\vdash \vec{C}, A \quad \vdash B, \vec{D}}{\vdash \vec{C}, A \otimes B, \vec{D}}$	(times)	$\dfrac{\vdash \vec{C}, A, B}{\vdash \vec{C}, A \parr B}$	(par)
$\dfrac{\vdash \vec{C}}{\vdash \vec{C}, ?A}$	(weakening)	$\dfrac{\vdash \vec{C}, ?A, ?A}{\vdash \vec{C}, ?A}$	(contraction)
$\dfrac{\vdash \vec{C}, A}{\vdash \vec{C}, ?A}$	(dereliction)	$\dfrac{\vdash ?\vec{C}, A}{\vdash ?\vec{C}, !A}$	(box)

The box rule is also called the promotion rule.

Example 3.1.3. We give an easy example of a derivation in sequent calculus.

$$\dfrac{\vdash A \otimes B^{\perp}, A^{\perp} \parr B \qquad \dfrac{\vdash A^{\perp}, A \quad \vdash B^{\perp}, B}{\vdash A^{\perp}, A \otimes B^{\perp}, B}}{\vdash A \otimes B^{\perp}, A^{\perp}, B}$$

3.2 Proof Nets

In this section the rewriting system $(\mathsf{PN}, \triangleright)$, consisting of the set of proof nets PN and a rewrite relation $\triangleright$ on PN, is presented.

Proof Nets. A multiplicative-exponential proof net is a graphical representation of a derivation in the sequent calculus of multiplicative-exponential classical linear logic. We will simply say 'proof net' instead of 'multiplicative-exponential proof net'. A proof net consists of occurrences of formulae and links connecting occurrences of formula. The links form in fact the nodes of a graph, with the exception of the box link, and the formulae are the types. Every formula occurrence is the conclusion of exactly one link and the premiss of at most one link.

An occurrence of a formula is a pair consisting of a formula and a label. We take the natural numbers as labels. Labels are used to be able to distinguish between two occurrences of the same formula, like in λ-calculus one wants to be able to distinguish between a variable x of type A and a variable y of type A.

Definition 3.2.1. A *formula occurrence* is a pair consisting of a formula and a label. Formula occurrences are denoted by $A_m, B_n, C_p, D_q, \ldots$.

Links are considered to be typed by formulae, but the types of links are not written explicitly.

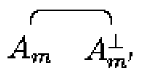
Definition 3.2.2. The alphabet of proof nets consists of the following:

1. for every formula A and for every label m there is a formula occurrence A_m,
2. for every formula A there exists an A-axiom link,
3. for every formula A there exists an A-cut link,
4. for all formulas A and B there exists an A-B-times link,
5. for all formulas A and B there exists an A-B-par link,
6. for every formula A there exists an A-weakening link,
7. for every formula A there exists an A-contraction link,
8. for every formula A there exists an A-dereliction link,
9. for all formulas $A, \vec{B}$ there exists an A-$\vec{B}$-box link.

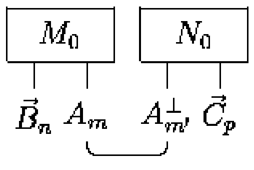
In Chapter 4 we consider proof nets over an alphabet that contains also proof net constants. In this chapter we do not consider proof net constants, since considering proof net with constants would not change the arguments nor the results (with exception of Proposition 3.2.8), but would complicate the syntax. Proof nets are built inductively from formula occurrences and links. Special formula occurrences are the *output occurrences*. These are the formula occurrences that are not the premiss of any link. The output occurrences of a proof net are defined in the inductive definition of proof nets, which is as follows. We indicate the part of a proof net M consisting of M without output occurrences by M_0.

Definition 3.2.3. The set of *proof nets*, denoted by PN, is the smallest set satisfying the following.

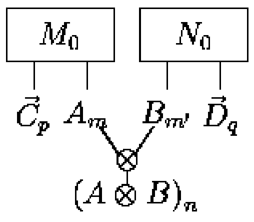
1. An A-axiom link is a proof net with output occurrences $A_m, A^{\perp}_{m'}$ for some fresh labels $m, m' \in \mathbb{N}$, written as



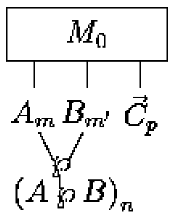
2. If M is a proof net with output occurrences A_m, $\vec{B}_n$ and N is a proof net with output occurrences $A^{\perp}_{m'}$, $\vec{C}_p$, then adding an A-cut link between the output occurrence A_m of M and the output occurrence $A^{\perp}_{m'}$ of N yields a proof net with output occurrences $\vec{B}_n$, $\vec{C}_p$, written as



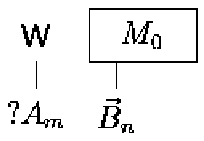
3. If M is a proof net with output occurrences $A_m, \vec{C}_p$ and N is a proof net with output occurrences $B_{m'}$, $\vec{D}_q$, then adding an A-B-times link between the output occurrence A_m of M and the output occurrence $B_{m'}$ of N yields a proof net with output occurrences $(A \otimes B)_n$, $\vec{C}_p$, $\vec{D}_q$, with $n \in \mathbb{N}$ a fresh label, written as



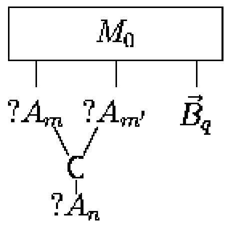
4. If M is a proof net with output occurrences $A_m, B_{m'}, \vec{C}_p$, then adding an A-B-par link between the output occurrences A_m and $B_{m'}$ yields a proof net with output occurrences $(A \wp B)_n, \vec{C}_p$ with $n \in \mathbb{N}$ a fresh label, written as



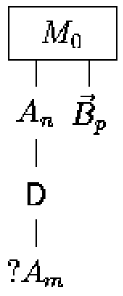
5. If M is a proof net with output occurrences $\vec{B}_n$, then adding an A-weakening link yields a proof net with output occurrences $?A_m$, $\vec{B}_n$, with $m \in \mathbb{N}$ a fresh label, written as



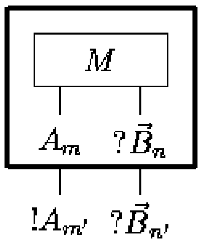
6. If M is a proof net with output occurrences $?A_m, ?A_{m'}, \vec{B}_q$, then adding an A-contraction link between the output occurrence $?A_m$ and the output occurrence $?A_{m'}$ yields a proof net with output occurrences $?A_n, \vec{B}_q$, with $n \in \mathbb{N}$ a fresh label, written as



7. If M is a proof net with output occurrences $A_n, \vec{B}_p$, then adding an A-dereliction link to the output occurrence A_n yields a proof net with output occurrences $?A_m, \vec{B}_p$, with $m \in \mathbb{N}$ a fresh label, written as



8. If M is a proof net with output occurrences $A_m, ?\vec{B}_n$, then adding an A-$\vec{B}$-box link yields a proof net with output occurrences $!A_{m'}, ?\vec{B}_{n'}$, with fresh labels $m', n'_1, \ldots, n'_p \in \mathbb{N}$, written as



The box link, which corresponds to the box rule of the sequent calculus, is used to indicate a part of a proof net in an explicit way. As we will see, the only parts of a proof net that can be duplicated by rewriting are the boxes.

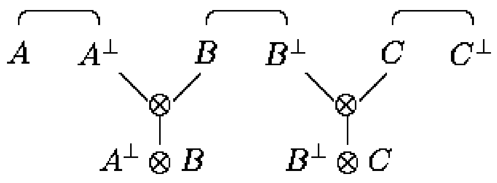
In the remainder of this chapter, we will often leave out the labels of the formula occurrences of a proof net.

The following example illustrates the advantage of proof nets over proofs in sequent calculus.

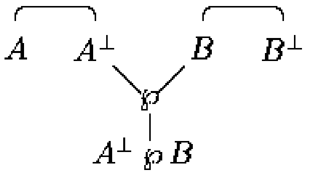
Example 3.2.4. We give an easy example of two different proofs in sequent calculus that are represented by the same proof net. The following two derivations in sequent calculus:

$$\dfrac{\dfrac{\vdash A, A^\perp \quad \vdash B, B^\perp}{\vdash A,\ A^\perp \otimes B,\ B^\perp} \quad \vdash C, C^\perp}{\vdash A,\ A^\perp \otimes B,\ B^\perp \otimes C, C^\perp} \qquad \dfrac{\vdash A, A^\perp \quad \dfrac{\vdash B, B^\perp \quad \vdash C, C^\perp}{\vdash B,\ B^\perp \otimes C,\ C^\perp}}{\vdash A,\ A^\perp \otimes B,\ B^\perp \otimes C, C^\perp}$$

are both represented by the following proof net:



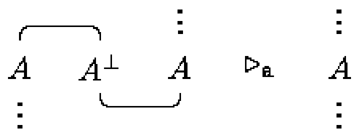
The definition of proof nets as originally given by Girard in [Gir87] differs from the one given in Definition 3.2.3. In the original definition, first the set of proof structures is defined. Proof structures are graphs built from the alphabet of proof nets given in Definition 3.2.2. Proof nets are defined as the proof structures satisfying a certain criterion, called the long trip condition [Gir87]. An alternative for the long trip condition has been given by Danos and Régnier in [DR89]. An example of a proof structure that is not a proof net is the following:



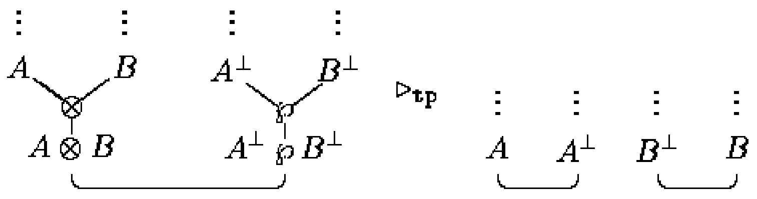
Rewriting Proof Nets. The rewrite relation on PN consists of cut-elimination. It is defined as follows.

Definition 3.2.5. The rewrite relation on PN, denoted by $\triangleright$, is the smallest relation that is compatible with the structure of proof nets and that is induced by the following rewrite rules.

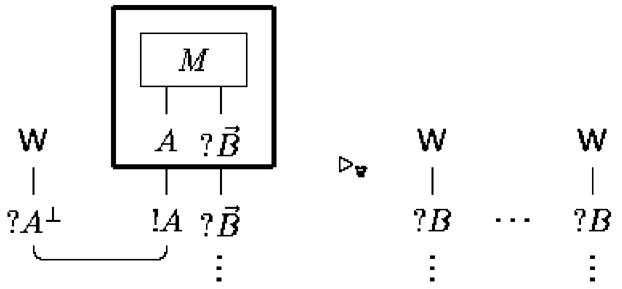
1. `axiom`



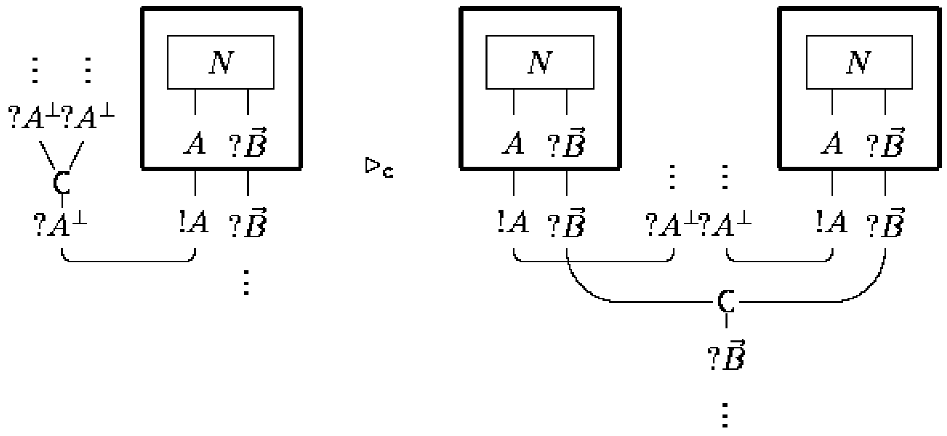
2. `times-par`



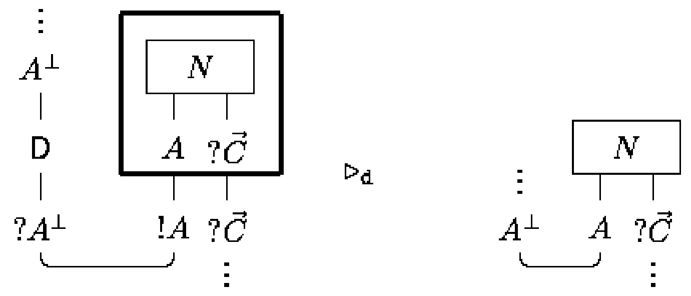
3. `weakening-box`



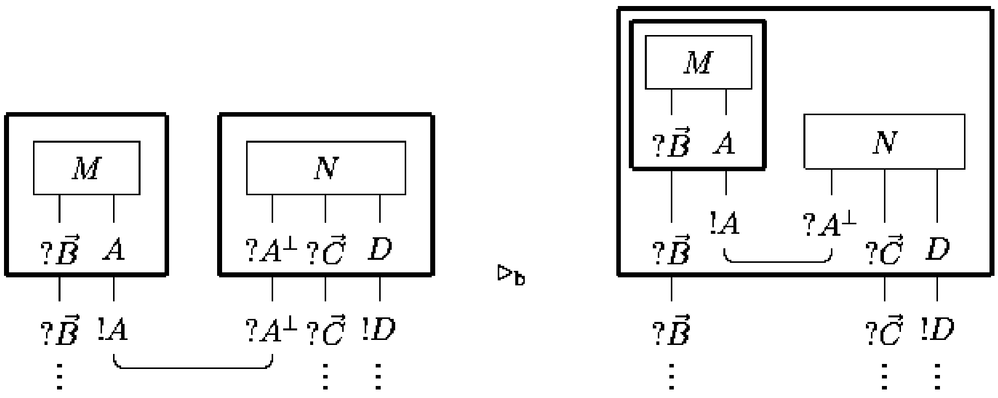
4. `contraction-box`



5. `dereliction-box`



6. `box-box`


M
$?\vec{B}$ A
$?\vec{B}$ $!A$
N
$?A^\perp$ $?\vec{C}$ D
$?A^\perp$ $?\vec{C}$ $!D$
$\rhd_b$
M
$?\vec{B}$ A
N
$!A$ $?A^\perp$
$?\vec{B}$ $?\vec{C}$ D
$?\vec{B}$ $?\vec{C}$ $!D$

Notation 3.2.6. The rewriting system of proof nets with cut-elimination as defined by the rewrite rules in Definition 3.2.5 is denoted by $(\mathsf{PN}, \rhd)$.

Labels of formulae are used only to have access to a certain occurrence of a formula in a proof net. We will suppose that all labels of formulae in a proof net are different. In particular, labels of a proof net are not used to trace a formula occurrence along a rewrite sequence.

We make two observations concerning $(\mathsf{PN}, \rhd)$. The first observation is that the set of proof nets with output formulae $A_1, \ldots, A_m$ is closed under the relation $\rhd$.

Proposition 3.2.7. *If M is a proof net with output formulae $A_1, \ldots, A_m$ and $M \rhd M'$, then M' is a proof net with output formulae $A_1, \ldots, A_m$.*

Proof. If M is a proof net and $M \rhd M'$, then it follows from an inspection of the rewrite rules inducing the relation $\rhd$ and of the derivation of $M \in \mathsf{PN}$ that M' is a proof net. □

So proof nets with cut elimination have an underlying abstract rewriting system of the form $(\mathsf{PN}, \rhd)$.

The second observation is that a proof net can be rewritten if and only if it contains a cut link. This proposition does not hold for the proof nets that possibly contain proof net constants, as we will consider in Chapter 4.

Proposition 3.2.8. *Let $M \in \mathsf{PN}$. Then M is in normal form with respect to $\rhd$ if and only if M does not contain a cut link.*

Proof. It is easy to see that a proof net that doesn't contain a cut link is in normal form with respect to $\rhd$.

Suppose M is a proof net that is obtained by adding a cut link between the output occurrence A of a proof net M_0 and the output occurrence $A^\perp$ of a proof net M_1. We show that this particular cut link can be eliminated by applying one of the rewrite rules defining the relation $\rhd$. If the output occurrence A in M_0 is the conclusion of an axiom link or if the output occurrence $A^\perp$ in M_1 is the conclusion of an axiom link, then the rewrite rule `axiom` can be applied that eliminates the cut link we consider. So suppose neither the output occurrence A nor the output occurrence $A^\perp$ is the conclusion of an axiom link. Since for every type B we have $(B^\perp)^\perp = B$ and A is not an atom, it is sufficient to consider the following two cases:

1. $A = B_0 \otimes B_1$ and $A^\perp = B_0^\perp \wp B_1^\perp$,
2. $A = !B$ and $A^\perp = ?B^\perp$.

We show that in both cases a rewrite rule applies that eliminates the cut link we consider.

1. Suppose $A = B_0 \otimes B_1$ and $A^\perp = B_0^\perp \wp B_1^\perp$. It must be the case that the output occurrence A of M is the conclusion of a times link and the output occurrence $A^\perp$ is the conclusion of a par link. Hence the rewrite rule `times-par` can be applied.
2. Suppose $A = !B$ and $A^\perp = ?B^\perp$. It must be the case that the output occurrence A is the conclusion of a box link. For $A^\perp$, we distinguish the following possibilities.
   (a) The output occurrence $A^\perp$ is the conclusion of a weakening link. Then the rewrite rule `weakening-box` can be applied.
   (b) The output occurrence $A^\perp$ is the conclusion of a contraction link. Then the rewrite rule `contraction-box` can be applied.

(c) The output occurrence $A^\perp$ is the conclusion of a dereliction link. Then the rewrite rule `dereliction-box` can be applied.

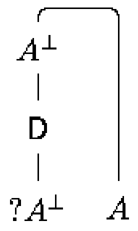
(d) The output occurrence $A^\perp$ is the conclusion of a box link. Then the rewrite rule `box-box` can be applied. □

In the following we will make use of a notion of residual that we introduce informally as follows. We label cut links, again with natural numbers. Labels of cut links are supposed to keep track of a cut link during a rewrite sequence. That is, if a cut link with label m is in a part of M that is not mentioned explicitly in the rewrite rule that is applied to obtain $M \rhd M'$, then the corresponding cut link in M' has also label m. In an application of the rewrite rule `axiom`, one cut link is eliminated. An application of the rewrite rule `times-par` eliminates one cut link and creates two cut links. The created cut links are supposed to have fresh labels. An application of the rewrite rule `weakening-box` eliminates a cut link, and moreover erases cut links that are possibly present in the part indicated by M in the rewrite rule. An application of the rewrite rule `contraction-box` creates two cut links that are supposed to have fresh labels. Moreover, the part indicated by N in the rewrite rule is duplicated. Copies of cut links in the box that is duplicated keep their labels. So every cut link in the part indicated by N has two residuals after eliminating the contraction-box cut link. An application of the rewrite rule `dereliction-box` creates a cut link that is supposed to have a fresh label. Finally, an application of the rewrite rule `box-box` also creates a new cut link that is supposed to have a fresh label.

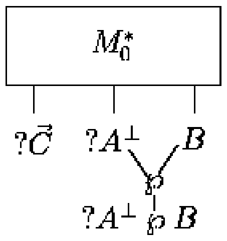
The label of a cut link does not determine a unique rewrite step, since there might be several cut links with the same label. However, one can suppose a cut link to be labelled by a composed label, containing both the label of a cut link, used to trace it along a rewrite sequence, and the labels of the formulae it connects, that are unique. We will sloppily talk about a cut link with label m in a proof net M, and suppose that it uniquely determines a cut link in M, and moreover that the label m can be used to trace that particular cut link along a rewrite sequence.

Lambda Terms and Proof Nets. To conclude this section, we briefly discuss the relationship between simply typed λ-terms and proof nets. A naive translation of simply typed λ-term M into a proof net M^* is defined inductively as follows.

1. A variable x of type A is translated into

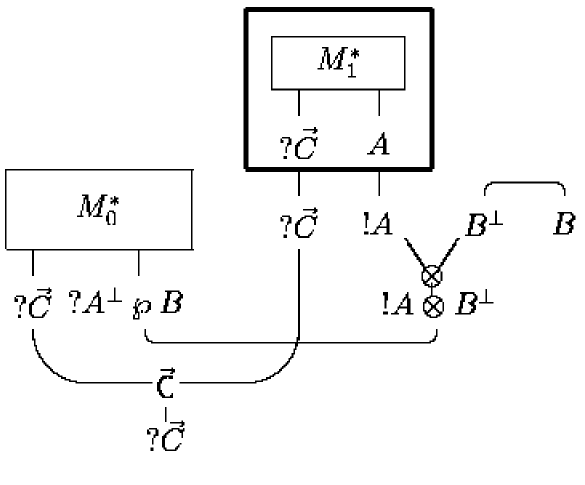


2. The translation of $\lambda x.M_0 : A \to B$ is



where the formula occurrence $A^\perp$ represents the occurrences of x in M_0. If $x \notin \mathsf{FV}(M_0)$, then an A weakening link is added to the translation of M, and next an A-B-par link is added as above.

3. The translation of $M_0 M_1 : B$ with $M_0 : A \to B$ and $M_1 : A$ is



Translations of λ-terms into proof nets are studied by Danos [Dan90] and Régnier [Rég92]. They present a translation that is less naive than the one given above in the sense that a λ-term in normal form is translated into a proof net in normal form. The translation above is considered in [Asp94] and also in [Raa92]. In [Raa92] moreover a translation due to Curien is studied.

3.3 Strong Normalisation

In this section we present a proof of strong normalisation of the rewriting system $(\mathsf{PN}, \triangleright)$. The proof proceeds in several steps and bears resemblance to the proof of strong normalisation of proof nets presented by Joinet in [Joi93] and the work concerning pure nets presented by Danos in Chapitre 8 of [Dan90]. First strong normalisation of proof nets containing only atomic axiom links is shown. This is also the first step in the proof by Joinet. The extension to the class of all proof nets is different from the one given by Joinet, and also the details of the first step are presented differently by Joinet. First we show weak normalisation of the rewrite relation generated by all rewrite rules except for the rewrite rule $\triangleright_{\mathsf{a}}$. Next we infer strong normalisation of the rewrite relation generated by all rewrite rules except for the rewrite rules $\triangleright_{\mathsf{a}}$ and $\triangleright_{\forall}$ by a method due to Nederpelt and Klop. Finally it is shown that this entails strong normalisation of the rewrite relation $\triangleright$. This result is extended to proof nets with arbitrary axiom links using expansion rules, that transform an axiom link into a proof net with the same outputs and only axiom links of less complex types.

We first restrict attention to proof nets with only atomic axiom links.

We will use the following notation.

Notation 3.3.1. The rewrite relation generated by the rewrite rule $\triangleright_{\mathsf{a}}$ is denoted by $\triangleright_{\mathsf{a}}$. The rewrite relation generated by all rewrite rules but the rule $\triangleright_{\mathsf{a}}$ is denoted by $\triangleright_{\neg\mathsf{a}}$.

A first observation is that the relation $\triangleright_{\mathsf{a}}$ is strongly normalising, since at every $\triangleright_{\mathsf{a}}$-rewrite step the number of axiom links decreases strictly. This holds as well in the presense of arbitrary axiom links.

In proof nets with only atomic axiom links, it is possible to postpone $\triangleright_{\mathsf{a}}$-rewrite steps. That is, every rewrite sequence $M \triangleright^* M'$ can be written as $M \triangleright^*_{\neg\mathsf{a}} N \triangleright^*_{\mathsf{a}} M'$. This is shown in the following lemma. It does not hold for proof nets possibly containing axiom links of compound formulae.

Lemma 3.3.2. *Let M be a proof net with only atomic axiom links. If $M \triangleright_{\mathsf{a}} M' \triangleright_{\neg\mathsf{a}} M''$, then there is a proof net N such that $M \triangleright_{\neg\mathsf{a}} N \triangleright^*_{\mathsf{a}} M''$.*

Proof. The cut link eliminated in the rewrite step $M' \triangleright_{\neg\mathsf{a}} M''$ was already present in M, since an $\triangleright_{\mathsf{a}}$-rewrite step does not create or copy cut links. Moreover, since all axiom links are atomic, this cut link can be eliminated in M already by an $\triangleright_{\neg\mathsf{a}}$-rewrite step. □

As a consequence, strong normalisation of proof nets with only atomic axiom links is implied by strong normalisation of the rewrite relation $\rhd_{\neg a}$. So we will now fix attention to the relation $\rhd_{\neg a}$.

Weak Normalisation. In this paragraph we show that the relation $\rhd_{\neg a}$ is weakly normalising for proof nets containing only atomic axiom links. We will define a notion of structural development. For its definition we first need the definition of the depth of an occurrence of a formula in a proof net, which is given as follows.

Definition 3.3.3. Let M be a proof net. The *depth* of a formula occurrence A_m in M, denoted by $\mathsf{depth}(A_m, M)$, is defined by induction on the structure of the formula A:

1. if A is the conclusion of an A-axiom link then $\mathsf{depth}(A_m, M) = 0$,
2. if $A = A_0 \otimes A_1$ and the formula occurrence A_m is the conclusion of an A_0-A_1-times link with as premisses the formula occurrences $(A_0)_{m_0}$ and $(A_1)_{m_1}$, then $\mathsf{depth}(A_m, M) = \max\{\mathsf{depth}((A_0)_{m_0}, M), \mathsf{depth}((A_1)_{m_1}, M)\} + 1$,
3. if $A = A_0 \wp A_1$ and the formula occurrence A_m is the conclusion of an A_0-A_1-par link with as premisses the formula occurrences $(A_0)_{m_0}$ and $(A_1)_{m_1}$, then $\mathsf{depth}(A_m, M) = \max\{\mathsf{depth}((A_0)_{m_0}, M), \mathsf{depth}((A_1)_{m_1}, M)\} + 1$,
4. if $A = ?A_0$ and the formula occurrence A_m is the conclusion of an A_0-weakening link, then $\mathsf{depth}(A_m, M) = 0$,
5. if $A = ?A_0$ and the formula occurrence A_m is the conclusion of an A_0-contraction link with as premisses the formula occurrences A_{m_0} and A_{m_1}, then $\mathsf{depth}(A_m, M) = \max\{\mathsf{depth}(A_{m_0}, M), \mathsf{depth}(A_{m_1}, M)\} + 1$,
6. if $A = ?A_0$ and the formula occurrence A_m is the conclusion of an A_0-dereliction link with as premiss the formula occurrence $(A_0)_{m_0}$, then we define $\mathsf{depth}(A_m, M) = \mathsf{depth}((A_0)_{m_0}, M) + 1$,
7. if $A = !A_0$ and the formula occurrence A_m is the conclusion of an A_0-B_0-…-B_m-box link with as premisses the formula occurrences $(A_0)_{m_0}, (?B_0)_{n_0}, \ldots, (?B_m)_{n_m}$, then $\mathsf{depth}(A_m, M) = \mathsf{depth}((A_0)_{m_0}, M) + 1$,
8. if $A = ?A_0$ and the formula occurrence A_m is the conclusion of a B-A_0-A_1-…-A_m-box link with as premisses the formula occurrences $B_n, (?A_0)_{m_0}, (?A_1)_{m_1}, \ldots, (?A_m)_{m_m}$, then $\mathsf{depth}(A_m, M) = \mathsf{depth}((?A_0)_{m_0}, M) + 1$.

The depth of a cut link is defined as the sum of the depths of the formulae it connects.

Definition 3.3.4. Let M be a proof net. The *depth* of a cut-link in M with label p and with premisses A_m and $A_n^\perp$, denoted by $\mathsf{depth}(p)$, is defined as $\mathsf{depth}(A_m, M) + \mathsf{depth}(A_n^\perp, M)$.

Now we can give the following definition which is essential for the definition of a structural development.

Definition 3.3.5. Let M be a proof net with only atomic axiom links, and let $\{m_1, \dots, m_p\}$ be a set of labels of cut links occurring in M, for some $p \geq 0$. By induction on the multiset $[\mathsf{depth}(m_1), \dots, \mathsf{depth}(m_p)]$ we define a proof net $f(M, \{m_1, \dots, m_p\})$ as follows.

1. If $p = 0$, then $f(M, \{m_1, \dots, m_p\}) = M$.
2. If $[\mathsf{depth}(m_1), \dots, \mathsf{depth}(m_p)] = [0, \dots, 0]$, then $f(M, \{m_1, \dots, m_p\})$ is obtained by eliminating all cut links $m_1, \dots, m_p$.
3. Otherwise, let m_q be a cut link of maximal depth. If m_q is neither a contraction-box cut link nor a box-box cut link, then $f(M, \{m_1, \dots, m_p\}) = f(M', X)$ with $M \overset{m_q}{\rhd} M'$ and X the set of all residuals of $\{m_1, \dots, m_p\}$ in M'. Note that the cut link m_q does not have a residual.

   If m_q is a contraction-box cut link, and $M \overset{m_q}{\rhd_{\mathsf{c}}} M'$, then $f(M, \{m_1, \dots, m_p\}) = f(M', X)$ with X the set consisting of all residuals of $\{m_1, \dots, m_p\}$ in M', and of the labels of the two cut links that are created by the rewrite step $M \overset{m_q}{\rhd_{\mathsf{c}}} M'$.

   If m_q is a box-box cut link, and $M \overset{m_q}{\rhd_{\mathsf{b}}} M'$, then $f(M, \{m_1, \dots, m_p\}) = f(M', X)$ with X the set consisting of all residuals of $\{m_1, \dots, m_p\}$ in M', and of the label of the cut link that is created by the rewrite step $M \overset{m_q}{\rhd_{\mathsf{b}}} M'$.

For proof nets with only atomic axiom links, it is easy to see that the proof net $f(M, \{m_1, \dots, m_p\})$ is well-defined in the second clause. Moreover, in the third clause the multiset $[\mathsf{depth}(m_1), \dots, \mathsf{depth}(m_p)]$ decreases in every step, since cut links that are multiplied have a lesser depth, and the cut links that are created and added to the multiset in the case of a $\rhd_{\mathsf{c}}$ or $\rhd_{\mathsf{b}}$ step have a lesser depth than the cut link that is eliminated in the rewrite step. Note that $M \rhd^* f(M, \{m_1, \dots, m_p\})$ for every set of cut links in M. The definition of a structural development is now given as follows. It is similar to the notion of complete structural reduction as defined for exponential cut links in [Dan90].

Definition 3.3.6. A structural development is defined as a rewrite sequence

$$M \rhd^* f(M, \{m_1, \dots, m_p\})$$

for some set of cut links $\{m_1, \ldots, m_p\}$ in M. A structural development is denoted by $M \rightsquigarrow M'$. A structural development from M to $f(M, \{m\})$ is denoted by $M \overset{m}{\rightsquigarrow} M'$.

Now we discuss the proof of weak normalisation as given in [Dan90], the method is basically due to Turing and is explained in [Gan80]. In the proof we make use of two measures, namely the height of a cut link and the height of a proof net, that we introduce informally. The *height of a formula occurrence*, is the sum of the heights of its premisses plus one, unless it is the conclusion of a ?-box link or it is a contraction link, then the height is just the sum of the height of its premisses. The *height of a cut link labelled by* m, denoted by $h(m)$, is the sum of the height of the formula occurrences it connects. The difference between height and depth is that ?-box links and contraction links do not contribute to the height of a formula occurrence, but they do contribute to the depth. Finally, the *height of a proof net* M, denoted by $h(M)$, is defined as the multiset $[h(m_1), \ldots, h(m_p)]$ with $m_1, \ldots, m_p$ all the cut links occurring in M.

Proposition 3.3.7. *Proof nets with only atomic axiom links are weakly normalising with respect to the rewrite relation* $\triangleright_{\neg\mathsf{a}}$.

Proof. Let M be a proof net with only atomic axiom links containing cut links $m_1, \ldots, m_p$. Consider a cut link m of maximal height such that in the structural development $M \overset{m}{\rightsquigarrow} M'$ no cut link of maximal height is multiplied. Such a cut link m exists. Now if $M \overset{m}{\rightsquigarrow} M'$, then the height of M' is strictly smaller with respect to the usual extension of the ordering on natural numbers to multisets of natural numbers. This is the case since $M \overset{m}{\rightsquigarrow} M'$ does not multiply a cut link of maximal height. This means that performing repeatedly a structural development of a cut link of maximal height as above, yields a rewrite sequence that eventually ends in a normal form with respect to $\triangleright_{\neg\mathsf{a}}$. □

The proof does not go through if also $\triangleright_{\mathsf{a}}$-steps are allowed since a $\triangleright_{\mathsf{a}}$-step might cause an increase of the height of some cut links.

Strong Normalisation. Now we will show strong normalisation of the rewrite relation $\triangleright$ for proof nets containing only atomic axiom links. As we already discussed, it is sufficient to show that the rewrite relation $\triangleright_{\neg\mathsf{a}}$ is strongly normalising. As a matter of fact, we can first concentrate on the rewrite relation generated by all rewrite rules except for the rule $\triangleright_{\mathsf{a}}$ and the rule $\triangleright_{\mathsf{w}}$. This is shown in the following lemma. First we need to introduce some notation.

Notation 3.3.8. The rewrite relation generated by all rewrite rules except for the rewrite rule $\triangleright_{\mathsf{a}}$ and the rewrite rule $\triangleright_{\mathsf{w}}$ is denoted by $\triangleright_{\neg\mathsf{a}\neg\mathsf{w}}$.

Lemma 3.3.9. *If $M \rhd_{\mathsf{w}} M' \rhd_{\neg\mathsf{a}\neg\mathsf{w}} M''$ then there exists a proof net N such that $M \rhd_{\neg\mathsf{a}\neg\mathsf{w}} N \rhd^{*}_{\mathsf{w}} M''$.*

Proof. As Lemma 3.3.2 this follows from an easy inspection of the rewrite rules. □

So in order to prove strong normalisation of $\rhd_{\neg\mathsf{a}}$, it is sufficient to show strong normalisation of $\rhd_{\neg\mathsf{a}\neg\mathsf{w}}$. Strong normalisation of this rewrite relation follows quite easily from weak normalisation, using Lemma 1.1.29 which is Corollary 5.19 in [Klo80]. In order to apply Lemma 1.1.29, we need to make the rewrite relation increasing. Therefore we restrict attention to the rewrite relation not generated by the rewrite rule $\rhd_{\mathsf{w}}$ (and not by $\rhd_{\mathsf{a}}$). The method of inferring strong normalisation from weak normalisation is due to Nederpelt [Ned73] and Klop [Klo80]. In order to apply this method for λ-calculus with β-reduction, one needs to consider the β-rewrite relation as generated by several rules. De Groote shows in [Gro93] how to infer strong normalisation from weak normalisation by distinguishing between rewrite steps $(\lambda x.M)N \to M[x := N]$ with $x \notin \mathsf{FV}(M)$ and steps with $x \in \mathsf{FV}(M)$. Cut elimination of proof nets forms in fact already a refinement of β-reduction, and here it is not necessary to introduce new rules.

Lemma 3.3.10. *Proof nets with only atomic axiom links are strongly normalising with respect to the rewrite relation $\rhd_{\neg\mathsf{a}\neg\mathsf{w}}$.*

Proof. We sketch the proof. Let M be a proof net containing only atomic axiom links that is weakly normalising with respect to the rewrite relation $\rhd_{\neg\mathsf{a}\neg\mathsf{w}}$. We assign a measure to M and to all its $\rhd_{\neg\mathsf{a}\neg\mathsf{w}}$-reducts as follows. We assign a weight to every cut link, except for contraction-box cut links and box-box cut links. The weights are supposed to distribute over ?-box links and over contraction links. In M, we assign to every cut link weight 1. Then, in a $\rhd_{\neg\mathsf{a}\neg\mathsf{w}}$ rewrite step, a cut link is eliminated and one or two cut links are created. We assign to the created cut links the weight of the eliminated cut link plus one, supposing that contraction-box cut links and box-box cut links have no weight. In this way we have that the weight of a proof net increases after finitely many rewrite steps. The rewrite relation $\rhd_{\neg\mathsf{a}\neg\mathsf{w}}$ is weakly normalising, hence the one with labels is weakly normalising as well. Moreover, local confluence holds. We conclude by Lemma 1.1.29 that the rewrite relation $\rhd_{\neg\mathsf{a}\neg\mathsf{w}}$ is strongly normalising. □

Theorem 3.3.11. *Proof nets with only atomic axiom links are strongly normalising.*

Proof. By Lemma 3.3.2 we can postpone the $\rhd_{\mathsf{a}}$-rewrite steps. To show strong normalisation of $\rhd_{\neg\mathsf{a}}$, it is sufficient to show strong normalisation of $\rhd_{\neg\mathsf{a}\neg\mathsf{w}}$ since by

Lemma 3.3.9 the $\triangleright_{\mathtt{w}}$ steps can be postponed. Hence the statement follows from Lemma 3.3.10. □

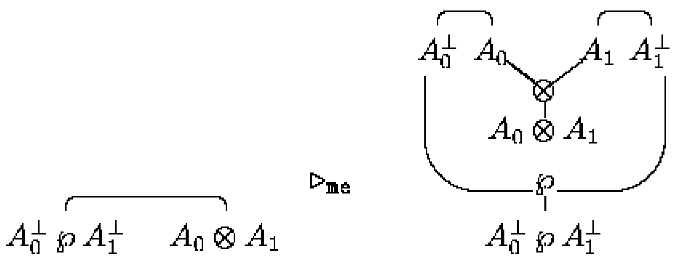
Expansion. The strong normalisation result for proof nets with only atomic axioms links is extended to the class of all proof nets. To that end we introduce expansion rules that give more structure to a proof net by replacing an axiom link of compound type A by a proof net with outputs A and $A^{\perp}$, containing only axiom links for formulae that are less complex than A.

In the remainder of this section we will consider proof nets with arbitrary axiom links.

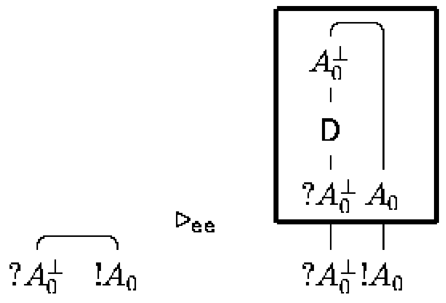
The expansion rules are defined as follows.

Definition 3.3.12. The rewrite relation $\triangleright_{\mathtt{e}}$ is defined as the smallest relation that is compatible with the structure of proof nets and that satisfies the following.

1. `multiplicative expansion`


$A_0^{\perp} \wp A_1^{\perp}$
$A_0 \otimes A_1$
$\triangleright_{\mathtt{me}}$
$A_0^{\perp}$
A_0
A_1
$A_1^{\perp}$
$A_0 \otimes A_1$
$\wp$
$A_0^{\perp} \wp A_1^{\perp}$

2. `exponential expansion`



Proposition 3.3.13. *The rewriting system* $(\mathsf{PN}, \rhd_{\mathsf{e}})$ *is strongly normalising and confluent.*

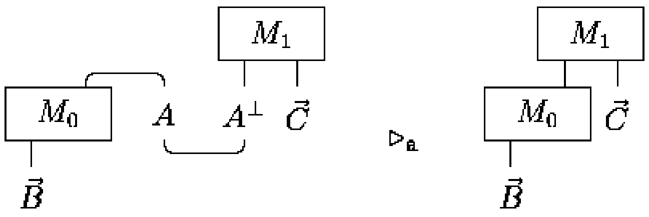
Proof. Strong normalisation follows since at every rewrite step the complexity of the axiom links decreases, and there is no duplication. Confluence follows from strong normalisation and local confluence. $\square$

Notation 3.3.14. We denote the $\rhd_{\mathsf{e}}$-normal form of a proof net M by $\mathsf{e}(M)$ and the $\rhd_{\mathsf{e}}$-normal form of an A-axiom link by $\mathsf{e}(A)$.

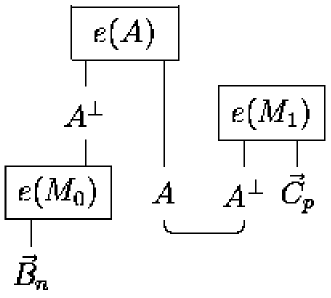
Note that $\mathsf{e}(M)$ is a proof net with only atomic axiom links.

Lemma 3.3.15. *If* $M \rhd M'$*, then* $\mathsf{e}(M) \rhd^{+} \mathsf{e}(M')$.

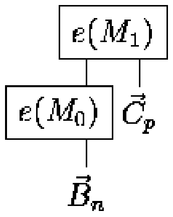
Proof. If $M \rhd_{\neg\mathsf{a}} M'$, then $\mathsf{e}(M) \rhd_{\neg\mathsf{a}} \mathsf{e}(M')$, since every cut link in M between formulae that are not the conclusion of an axiom link is present in $\mathsf{e}(M)$ as well. Suppose $M \rhd M'$ by an application of the rewrite rule `axiom`. Then the rewrite step $M \rhd_{\mathsf{a}} M'$ is of the following form:



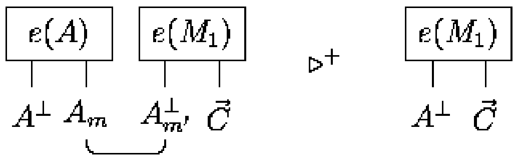
The proof net $\mathsf{e}(M)$ is of the form



The proof net $\mathsf{e}(M')$ is of the form

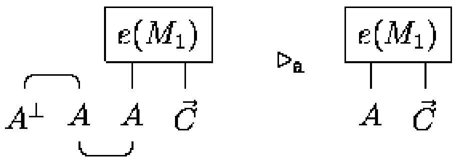


We show that

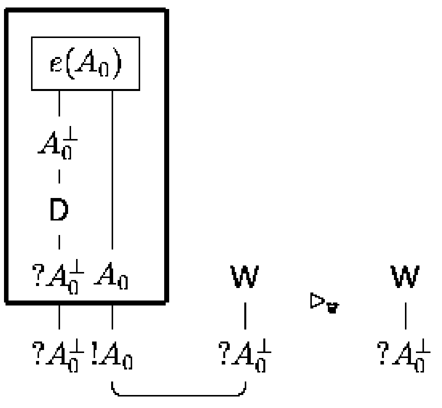


This yields that $\varepsilon(M) \triangleright^+ \varepsilon(M')$.
The proof proceeds by induction on $\mathsf{depth}(A^\perp_{m'}, \varepsilon(M_1))$.

I. Suppose $\mathsf{depth}(A^\perp_{m'}, \varepsilon(M_1)) = 0$. There are two possibilities. If A is an atom, then we clearly have

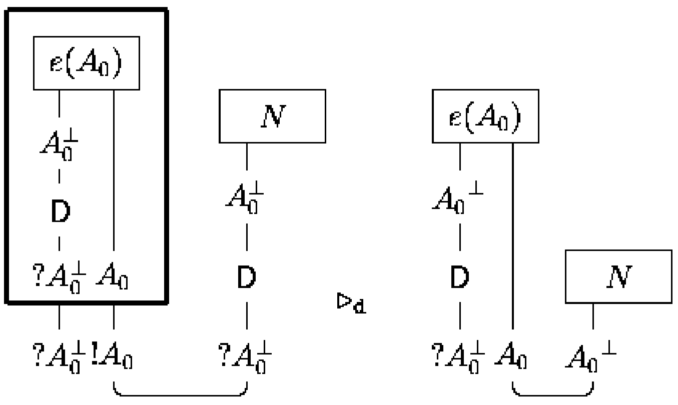


If $A = ?A_0^\perp$ and A is the conclusion of a weakening link, then



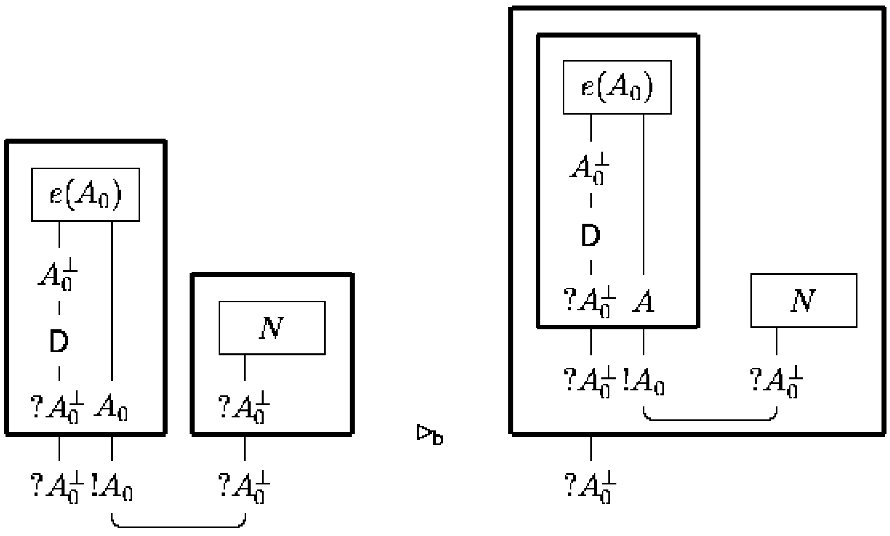
II. Suppose $\mathsf{depth}(A^\perp_{m'}, \varepsilon(M_1)) > 0$. If $A = A_0 \otimes A_1$, $A = A_0 ⅋ A_1$ or $A = ?A_0$ then the statement follows easily by induction hypothesis. Suppose $A = !A_0$. Three cases are distinguished.

1. $A_m^\perp$ in $\varepsilon(M_1)$ is the conclusion of a dereliction link. Then we have



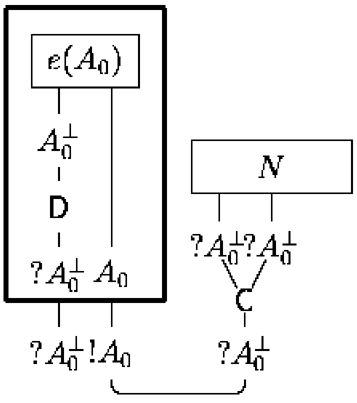
where N indicates a part of $\varepsilon(M_1)$. The induction hypothesis then yields the desired conclusion.

2. $A_m^\perp$ in $\varepsilon(M_1)$ is the conclusion of a ?-box link. Then we have

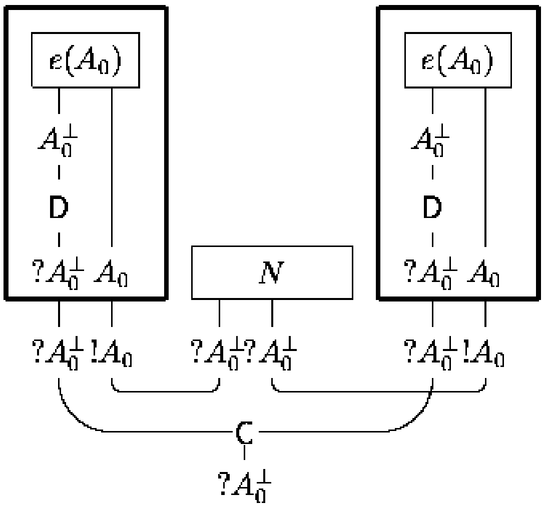


where N indicates a part of $\varepsilon(M_1)$. Then we can apply the induction hypothesis.

3. $A_m^\perp$ in $\varepsilon(M_1)$ is the conclusion of a contraction link. Then we have that the proof net



is rewritten to the proof net



Then we can apply the induction hypothesis. □

Corollary 3.3.16. (PN, $\triangleright$) *is strongly normalising.*

Proof. By Theorem 3.3.11 and Lemma 3.3.15. □

Chapter 4

Higher-Order Rewriting

This chapter is concerned with higher-order rewriting. A higher-order rewriting system describes transformations of functions. We will consider functions to be coded by means of abstraction like in the λ-calculus. This motivates to define higher-order rewriting systems as rewriting systems in which a binding mechanism for variables is present. The paradigmatic example of a higher-order rewriting system is the λ-calculus. It is however important to study not only λ-calculus, but also extensions of λ-calculus. Some extensions are definable, like λ-calculus with arithmetic, but other extensions are not, like for instance λ-calculus with surjective pairing. Even if an extension is definable in λ-calculus, it is often more convenient to add data-types explicitly. Moreover, this brings the syntax closer to the syntax of functional programming languages.

We consider an example of a higher-order rewriting system in order to illustrate the difference between first- and higher-order rewriting. An example of a first-order rewrite rule is $0 + x \to x$, which models the operation of adding zero in the natural numbers. The left-hand side and the right-hand side both model a natural number, and the variable x can be instantiated by some term that also models a natural number. Typical examples of well-known rules concerning transformations of functions are the rules describing how to obtain the derivative of a function that can be seen as a combination of two functions. For instance, the derivative of the sum of two functions is the sum of the derivatives: $\mathsf{dif}(f+g)(x) = \mathsf{dif}(f)(x) + \mathsf{dif}(g)(x)$. Writing a function f as $\lambda x.fx$, we can model this law by a rewrite rule

$$\mathsf{dif}(\lambda x.fx + \lambda x.gx)(y) \to \mathsf{dif}(\lambda x.fx)(y) + \mathsf{dif}(\lambda x.gx)(y).$$

This example illustrates that there is moreover another natural way in which higher-order rewrite rules arise, namely when reasoning about equations. Many equations in mathematics and logic contain bound variables, and techniques for higher-order rewriting can be used to answer questions concerning these equational theories.

There exist various formalisations of higher-order rewriting. We will mostly restrict attention to higher-order term rewriting, in which the objects that are rewritten have a term structure. Aczel defines in [Acz78] the class of contraction schemes, that contains besides untyped λ-calculus also for instance rewriting systems for pairing and primitive recursion. Not every first-order term rewriting system is however a contraction scheme. Around the same time, Hindley considered in [Hin78] $\lambda(a)$-reduction, which is λ-calculus extended with constants and rules for these constants.

The first to consider a uniform approach to higher-order rewriting was Klop, who defined in [Klo80] the class of Combinatory Reduction Systems, in the sequel shortly called CRSs. The class of CRSs contains λ-calculus and all first-order term rewriting systems, so CRSs form a generalisation of contraction schemes.

In subsequent years, quite some other formats of higher-order rewriting were introduced. The class of Expression Reduction Systems, in the sequel abbreviated as ERSs, is introduced by Khasidashvili in [Kha90]. A different approach to higher-order rewriting was taken by Nipkow in [Nip91]. He defined Higher-Order Rewrite Systems, in the sequel shortly called HRSs, that use simply typed λ-calculus with β-reduction and restricted $\overline{\eta}$-expansion (written as $\lambda_{\overline{\eta}}^{\rightarrow}$) as a meta-language. Wolfram defines in [Wol91] Higher-Order Term Rewriting Systems. Like HRSs, they have simply typed λ-calculus as meta-language, however, rules in a Higher-Order Term Rewriting System are more general than in HRSs. Asperti and Laneve consider in [Lan93] and in [AL94] Interaction Systems, in the sequel abbreviated as ISs, in order to study the theory of optimal reductions of Lévy [Lév78] in a setting that is more general than λ-calculus. ISs form a subclass of the class of CRSs and a superclass of the Interaction Nets defined by Lafont in [Laf90].

For all systems mentioned above, and especially for CRSs, ERSs, HRSs and ISs, quite some rewriting theory is developed. We will mention some results in the sections in which these systems are discussed. Since CRSs, ERSs, HRSs and ISs all have a different syntax, one of the reasons to study higher-order rewriting, namely to consider uniform approach to rewriting, is completely lost. A natural question is then in what way the results obtained for the different formats of higher-order rewriting systems relate to each other. Therefore we need to understand how the different classes of systems are related. Since contraction schemes and ERSs are similar to CRSs, and Higher-Order Term Rewriting Systems are similar to HRSs, a reasonable start is to compare CRSs and HRSs.

In collaboration with Vincent van Oostrom we made a detailed study of the relationship between CRSs and HRSs. This comparison, reported in [OR94a, OR93], revealed that the two formats have roughly speaking the same expressive power, and that the main difference between them is the meta-language which is employed. In the case of CRSs, it is untyped λ-calculus with developments, although this is slightly implicit, and in the case of HRSs it is $\lambda_{\overline{\eta}}^{\rightarrow}$ with reduction to normal form. The observation that formats of higher-order rewriting can differ in the meta-language that they employ leads to the conclusion that in order to come to a uniform approach it is necessary to parametrise over the meta-language.

Also in collaboration with Vincent van Oostrom, we defined the class of higher-order rewriting systems. In higher-order rewriting systems, the meta-language is is a parameter, called the *substitution calculus*. This parametrisation makes higher-order rewriting systems into a very expressive class. The structure of the objects that are rewritten is specified by the substitution calculus, so we do not make a commitment to term rewriting a priori.

Higher-order rewriting systems are defined in [Oos94] and in [OR94b]. The definition we present in Section 4.1 of this chapter differs from the ones given in [Oos94] and in [OR94b], which are mutually also slightly different. However, a closer inspection of the definitions reveals that the underlying concepts are identical.

For the convention concerning the names of the different formats of higher-order rewriting we refer to the Introduction.

In Section 4.1 we present a general definition of higher-order rewriting systems, in which the substitution calculus is not specified. The substitution calculus should satisfy some requirements, most of which are quite natural for a calculus that is meant to implement substitution.

In the two subsequent sections, we consider more concrete classes of higher-order rewriting systems, where the substitution calculus is fixed. In Section 4.2, we give the syntax of higher-order rewriting systems with $\lambda^{\rightarrow}_{\overline{\eta}}$ as substitution calculus. We first present a general definition and then restrict attention to the systems in which the rewrite rules are of a special form, namely the pattern higher-order rewriting systems. Such higher-order rewriting systems are in fact the HRSs as defined by Nipkow, with some small differences that we will discuss in Section 4.4. The results presented in Chapter 5 and Chapter 6 concern pattern higher-order rewriting systems with $\lambda^{\rightarrow}_{\overline{\eta}}$ as substitution calculus. In Chapter 5 we show that weakly orthogonal higher-order rewriting systems are confluent, and in Chapter 6 it is shown that outermost-fair rewriting is normalising for higher-order rewriting systems that are almost orthogonal and fully extended.

Section 4.3 is concerned with higher-order rewriting systems with proof nets as substitution calculus. In such a higher-order rewriting system, the objects that are rewritten have a graph structure. This section is less elaborate than the previous one.

The next four sections of this chapter are concerned with different formats of higher-order term rewriting. We show that HRSs, CRSs, ERSs and ISs can all be translated into higher-order rewriting systems with $\lambda^{\rightarrow}_{\overline{\eta}}$ as substitution calculus. In Section 4.4 this is done for HRSs as defined by Nipkow in [Nip91]. We explain moreover the (small) differences between HRSs and higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus. In Section 4.5 we discuss the CRSs defined by Klop in [Klo80]. We explain how a CRS can be represented as a higher-order rewriting system with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus, using the translation of CRSs into HRSs presented in [OR94a]. Section 4.6 concerns the ERSs introduced by Khasidashvili in [Kha90]. We will translate ERSs into higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus and comment on the differences between CRSs and ERSs. In Section 4.7 we discuss the class of ISs, that is defined by Asperti and Laneve in [Lan93] and in [AL94]. We present a translation of ISs into higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ that is not a special case of the translation of CRSs given in Section 4.5.

Finally, in Section 4.8 we give an example of a typed higher-order rewriting system by representing PCF, defined by Plotkin in [Plo77], as a higher-order rewriting system with system F as substitution calculus.

4.1 General Definition

In this section we present the syntax of higher-order rewriting systems in a general way, in the sense that the structure of the expressions that are rewritten is not specified. In Section 4.2, a more concrete case is considered in which the expressions have a term structure, and in Section 4.3 a more concrete case is considered in which the expressions have a graph structure. The advantage of the abstract approach of this section is that it permits to introduce the basic concepts of higher-order rewriting without too many syntactical details.

Material. We suppose the following material is at our disposal.

1. There is a non-empty set denoted by Types consisting of *types*. Types are denoted by $c, d, \ldots$.

2. For every $c \in \mathsf{Types}$ there is a set denoted by Var_c consisting of infinitely many *variables of type c*. The set denoted by Var consists of all variables:

$$\mathsf{Var} = \cup_{c \in \mathsf{Types}} \mathsf{Var}_c.$$

   Variables are denoted by $x, y, z, \ldots$.

3. For every $c \in \mathsf{Types}$ there is a set denoted by $\mathsf{PreStruc}_c$ consisting of *prestructures of type c*. The set denoted by $\mathsf{PreStruc}$ consists of all prestructures:

$$\mathsf{PreStruc} = \cup_{c \in \mathsf{Types}} \mathsf{PreStruc}_c.$$

   Prestructures are denoted by $a, b, \ldots$. It is assumed that

$$\mathsf{Var}_c \subseteq \mathsf{PreStruc}_c$$

for every $c \in \mathsf{Types}$.

4. For all types $c, d \in \mathsf{Types}$ there is a *substitution operator*

$$\_[\_ := \_]_{c,d} : \mathsf{PreStruc}_d \times \mathsf{Var}_c \times \mathsf{PreStruc}_c \to \mathsf{PreStruc}_d$$

that substitutes a prestructure of type c for a variable of type c in a prestructure of type d.

Note that it is not necessarily the case that the sets Types and $\mathsf{PreStruc}$ are disjoint. Usually we write $a[x := b]$ instead of $a[x := b]_{c,d}$ if the types are clear from the context or irrelevant.

Substitution Calculus. A substitution calculus is a rewriting system using the ingredients mentioned above. The definition makes use of the properties of subject conversion and type conversion which are defined in Definition 1.3.4 of Chapter 1.

Definition 4.1.1. A *substitution calculus* is an abstract rewriting system with typing of the form $(\mathsf{PreStruc} \cup \mathsf{Types}, \to_{SC}, :)$ that satisfies the following requirements.

1. The rewrite relation $\to_{SC}$ satisfies the following requirements:
   (a) $\to_{SC} \subseteq \mathsf{PreStruc} \times \mathsf{PreStruc}$,
   (b) $\to_{SC}$ is confluent,
   (c) $\to_{SC}$ is strongly normalising.
2. The typing relation $: \subseteq \mathsf{PreStruc} \times \mathsf{Types}$ is defined by

$$a : c \text{ if } a \in \mathsf{PreStruc}_c.$$

3. The rewrite relation $\to_{SC}$ satisfies the properties subject conversion and type conversion with respect to the typing relation :, that is
   (a) if $a \leftrightarrow_{SC} a'$ and $a : c$, then $a' : c$,
   (b) if $c \leftrightarrow_{SC} c'$ and $a : c$, then $a : c'$.
4. For all types $c, d \in \mathsf{Types}$ the relation $\leftrightarrow^*_{SC}$ and the operator $\_[\_ := \_]_{c,d}$ behave well with respect to each other in the following sense:
   (a) if $a \leftrightarrow^*_{SC} a'$ then $b[x := a]_{c,d} \leftrightarrow^*_{SC} b[x := a']_{c,d}$,
   (b) if $b \leftrightarrow^*_{SC} b'$ then $b[x := a]_{c,d} \leftrightarrow^*_{SC} b'[x := a]_{c,d}$.

The rewrite relation $\rightarrow_{SC}$ is meant to compute substitutions on a meta-level. A prestructure that is not in normal form with respect to the rewrite relation $\rightarrow_{SC}$ can be thought of as containing substitutions that are not yet executed. The first requirement on a substitution calculus in the previous definition, concerning its rewrite relation, ensures that every prestructure can be rewritten to a unique normal form with respect to $\rightarrow_{SC}$. That is, for every prestructure there is a unique representative in the same equivalence class with respect to $\leftrightarrow^{*}_{SC}$ in which all substitutions have been computed. A natural question is whether it is sufficient to require the rewrite relation $\rightarrow_{SC}$ to be weakly normalising instead of strongly normalising, and equipped with a normalising strategy. Conceptually this doesn't pose any problem, however, technically it might. Quite some proofs make use of an induction measure based on $\rightarrow_{SC}$, like for instance noetherian induction on an $\rightarrow_{SC}$-rewrite sequence, and it is not always clear whether the proof can be changed such that weak normalisation is sufficient.

The second requirement specifies the typing relation. Prestructures are typed a priori, so every prestructure is typed. Moreover, every type is inhabited since we suppose that there are infinitely many variables of every type.

The last requirement expresses that if a prestructure is of the form $b[x := a]$, then we can compute b and a separately. By the third requirement, both $b[x := a']_{c,d}$ and $b'[x := a]_{c,d}$ are well-defined, if $a \leftrightarrow^{*}_{SC} a'$ and $b \leftrightarrow^{*}_{SC} b'$.

Definition 4.1.2. A *structure* is a prestructure that is in normal form with respect to the rewrite relation $\rightarrow_{SC}$. The set of structures is denoted by Struc.

We denote by $a{\downarrow}_{SC}$ the normal form of a with respect to $\rightarrow_{SC}$, following Notation 1.1.18 of Chapter 1. We suppose that $\mathsf{Var} \subseteq \mathsf{Struc}$, so variables are in normal form with respect to the rewrite relation $\rightarrow_{SC}$. Alternatively we could have required that $\rightarrow_{SC} \subseteq (\mathsf{PreStruc} \setminus \mathsf{Var}) \times \mathsf{PreStruc}$.

In all the substitution calculi that are considered in this thesis, the set of types and the set of prestructures are disjoint. However, it could be interesting to consider a substitution calculus with dependent types.

Higher-Order Rewriting Systems.

Definition 4.1.3. Let SC be a substitution calculus. Let $c \in \mathsf{Types}$. A *rewrite rule of type c for SC* is a pair of structures $(\mathtt{l}, \mathtt{r})$ of type c. Rewrite rules are denoted by $\mathtt{l} \rightarrow \mathtt{r}, \mathtt{g} \rightarrow \mathtt{d}, \ldots$.

A higher-order rewriting system in the abstract setting is specified by a substitution calculus SC and a set of rewrite rules for SC.

Definition 4.1.4. A *higher-order rewriting system* is a pair $(SC, \mathcal{R})$ consisting of a substitution calculus SC and a set of rewrite rules $\mathcal{R}$ for SC.

The rewrite rules generate a rewrite relation on the set of objects. The idea is that a structure a can be rewritten if it is in the same equivalence class with respect to $\leftrightarrow^*_{\mathcal{SC}}$ as a prestructure of the form $b[x := \mathtt{l}]$, in which the left-hand side of a rewrite rule is substituted for a variable. Such a structure a can then be rewritten to a structure a' that is obtained by rewriting the prestructure $b[x := \mathtt{r}]$, obtained by replacing $\mathtt{l}$ in $b[x := \mathtt{l}]$ by $\mathtt{r}$, to $\rightarrow_{\mathcal{SC}}$-normal form. Since the rewrite relation of the substitution calculus is confluent and strongly normalising, such a structure a' exists and is unique. These are the considerations on which the following definitions of redex occurrence, redex and rewrite step are based.

Definition 4.1.5. Let $\mathcal{H} = (\mathcal{SC}, \mathcal{R})$ be a higher-order rewriting system.

1. Let a be a structure. A *redex occurrence in a* is a triple $(b, x, \mathtt{l} \rightarrow \mathtt{r})$ consisting of a structure, a variable, and a rewrite rule such that $a = b[x := \mathtt{l}]\downarrow_{\mathcal{SC}}$. The set of all redex occurrences in the structure a is denoted by $\mathfrak{U}_{\mathcal{R}}(a)$.
2. A *redex* is a pair $(a, (b, x, \mathtt{l} \rightarrow \mathtt{r}))$ such that $(b, x, \mathtt{l} \rightarrow \mathtt{r})$ is a redex occurrence in a. The set of all triples $(b, x, \mathtt{l} \rightarrow \mathtt{r})$ consisting of a structure, a variable and a rewrite rule is denoted by $\mathfrak{U}_{\mathcal{R}}$.
3. A *rewrite step* is a triple $(a, (b, x, \mathtt{l} \rightarrow \mathtt{r}), a')$ such that
   (a) $a = b[x := \mathtt{l}]\downarrow_{\mathcal{SC}}$,
   (b) $a' = b[x := \mathtt{r}]\downarrow_{\mathcal{SC}}$.
4. Both the redex occurrence $(b, x, \mathtt{l} \rightarrow \mathtt{r})$ and the redex $(a, (b, x, \mathtt{l} \rightarrow \mathtt{r}))$ are said to be *contracted* in the rewrite step $(a, (b, x, \mathtt{l} \rightarrow \mathtt{r}), a')$.

Notation 4.1.6. A rewrite step $(a, (b, x, \mathtt{l} \rightarrow \mathtt{r}), a')$ as in Definition 4.1.5 is usually denoted by $(b, x, \mathtt{l} \rightarrow \mathtt{r}) : a \rightarrow a'$.

Since the substitution calculus is required to be confluent, one can view a rewrite step in a higher-order rewriting system as consisting of three phases: an expansion in the substitution calculus, followed by a replacement of the right-hand side of a rewrite rule for the left-hand side of that rewrite rule, followed by a reduction to normal form in the substitution calculus:

$$a \;{}_{\mathcal{SC}}\!\twoheadleftarrow b[x := \mathtt{l}] \rightsquigarrow b[x := \mathtt{r}] \twoheadrightarrow_{\mathcal{SC}} a'.$$

A higher-order rewriting system $(\mathcal{SC}, \mathcal{R})$ has an underlying abstract rewriting system of the form $(\mathsf{Struc}, \rightarrow_{\mathcal{R}})$, with $a \rightarrow_{\mathcal{R}} a'$ if there is a rewrite step $(b, x, \mathtt{l} \rightarrow \mathtt{r}) : a \rightarrow a'$. In fact, since the rewrite relation $\rightarrow_{\mathcal{SC}}$ satisfies the property of subject conversion with respect to the typing relation :, this abstract rewriting system is in fact an abstract rewriting system with typing of the form $(\mathsf{Struc} \cup \mathsf{Types}, \rightarrow_{\mathcal{R}}, :)$.

Moreover, a higher-order rewriting system $(\mathcal{SC}, \mathcal{R})$ determines an underlying functional rewriting system. It is of the form $(\mathsf{Struc}, \mathfrak{U}_{\mathcal{R}}, \rightarrow)$, where $\rightarrow$ is defined by $\rightarrow((b, x, \mathrm{l} \rightarrow \mathrm{r}))(a) = a'$ if $(b, x, \mathrm{l} \rightarrow \mathrm{r}) : a \rightarrow a'$, and $\rightarrow((b, x, \mathrm{l} \rightarrow \mathrm{r}))(a)$ being undefined otherwise.

The formalisation of the notion of redex and redex occurrence as in Definition 4.1.5 is not completely satisfactory since there might be many different redexes or redex occurrences determining a rewrite step between objects a and a' in, intuitively speaking, the same way. Note that a restriction is already imposed by requiring b in the definition of a rewrite step to be a structure and not just a prestructure. This is the best we can do in the setting of the present section. What matters is that the notion of redex should be formalised in such a way that it completely determines a rewrite step, and a redex occurrence in a term should be formalised such that it uniquely determines a rewrite step issuing from that term. For every substitution calculus, the notions of redex occurrence, redex and rewrite step can be formalised as in Definition 4.1.5. However, once an actual substitution calculus is fixed, another formalisation might be more convenient.

Since a higher-order rewriting system has the structure of a functional rewriting system, it is possible to define a descendant relation for some property P. We inspect the definition of the rewrite relation in order to see what is a natural way of tracing a property along a rewrite sequence. As remarked above already, a rewrite step in a higher-order rewriting system can be decomposed as an expansion in the substitution calculus, a change from substituting the left-hand side of a rewrite rule to substituting the right-hand side of a rewrite rule and a reduction in the substitution calculus: $a \;_{\mathcal{SC}}\!\twoheadleftarrow b[x := \mathrm{l}] \rightsquigarrow b[x := \mathrm{r}] \twoheadrightarrow_{\mathcal{SC}} a'$. If we want to trace a property along a rewrite step, it seems natural to make use of this decomposition. That means that we first trace the property along an expansion in the substitution calculus, then along a replacement and finally along a reduction to normal form in the substitution calculus.

In general, it is not necessarily the case that a property P which is to be traced along a $\rightarrow_{\mathcal{R}}$-rewrite sequence can be traced along conversions in the substitution calculus. It is in fact sufficient that we can trace a property say P' along conversions in the substitution calculus, that is related to P in a suitable way, for instance via an isomorphism. In that case the property P is coded as the property P'.

It is possible to work this all out technically in the abstract setting of this section. That is, we could define a descendant relation Des tracing a property P for a higher-order rewriting system $\mathcal{H} = (\mathcal{SC}, \mathcal{R})$ via a descendant relation $\mathsf{Des}_{\mathcal{SC}}$ tracing a property P' for the substitution calculus $\mathcal{SC}$. However, we feel that this doesn't really add to the understanding of these matters. Therefore we only give the definition of a descendant relation for the more concrete case of a higher-order rewriting system with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus in Section 4.2. One thing should be clear from this discussion, namely that in order to define a descendant relation for a higher-order rewriting system in a natural way, its substitution calculus should be a functional rewriting system.

4.2 Lambda Calculus as Substitution Calculus

In this section we present the syntax of the class of higher-order rewriting systems that have simply typed λ-calculus with β-reduction and restricted η-expansion as substitution calculus. Simply typed λ-calculus with β-reduction and restricted η-expansion is denoted by $\lambda^{\rightarrow}_{\overline{\eta}}$. This is the prime example of a substitution calculus. In the Sections 4.4, 4.5, 4.6 and 4.7 we will see that various formalisms of higher-order rewriting can be represented as a higher-order rewriting system with $\lambda^{\rightarrow}_{\overline{\eta}}$ as substitution calculus.

In this section, first the calculus $\lambda^{\rightarrow}_{\overline{\eta}}$ is presented. We restrict attention to the differences between $\lambda^{\rightarrow}_{\overline{\eta}}$ and simply typed λ-calculus with β-reduction as defined in Section 2.5 of Chapter 2. Then we present the definition of higher-order rewriting systems with $\lambda^{\rightarrow}_{\overline{\eta}}$ as substitution calculus. This definition is still very general, since for instance a rewrite rule like $x \rightarrow a$, in a slightly different syntax, is allowed. Next we present an important subclass consisting of the so-called pattern higher-order rewriting systems. The restriction to pattern higher-order rewriting systems is very usual in the literature. As already mentioned, the results concerning confluence presented in Chapter 5 and the results concerning normalisation presented in Chapter 6 are obtained for pattern higher-order rewriting systems with $\lambda^{\rightarrow}_{\overline{\eta}}$ as substitution calculus, satisfying certain conditions. In this chapter we will collect some material needed in Chapter 5 and Chapter 6. For instance, the notions of orthogonal, almost orthogonal and weakly orthogonal higher-order rewriting systems are defined.

The reader familiar with HRSs (Higher-Order Rewrite Systems) as defined by Nipkow in [Nip91], might find it helpful to keep in mind that higher-order rewriting systems with $\lambda^{\rightarrow}_{\overline{\eta}}$ as substitution calculus are in fact HRSs with two differences: left- and right-hand side of a rewrite rule are not required to be of base type and the rewrite relation is defined in a different way.

The Calculus $\lambda^{\rightarrow}_{\overline{\eta}}$. We present the syntax of the calculus $\lambda^{\rightarrow}_{\overline{\eta}}$ as far as it differs from the syntax of simply typed λ-calculus as presented in Section 2.5 of Chapter 2. The only differences concern the notation and the rewrite relation, since next to β-reduction also restricted η-expansion is considered.

Simply typed λ-terms are built from simply typed variables and simply typed constants. Simply typed λ-terms are also called *preterms*, because they play the rôle of prestructures as in the general definition of higher-order rewriting systems presented in Section 4.1.

Notation 4.2.1. Simply typed λ-terms or preterms are denoted by $s, t, r, \ldots$. We write $x.s$ instead of $\lambda x.s$. The set of simply typed λ-terms of type A is denoted by $\Lambda^{\rightarrow}_{A}$. We write also $s : A$ to express that the λ-term s is of type A.

So the main difference between the notation employed in this section and the notation employed in Chapter 2 is that in the present section we write $x.s$ instead of $\lambda x.s$. The advantage of the notation $x.s$ is that it is less heavy than $\lambda x.s$, and moreover, in this way we keep the symbol λ for special occasions.

The substitution operator for $\lambda^{\rightarrow}_{\overline{\eta}}$ is typed explicitly. We recall the typed version of the definition. As before, we assume that bound variables are renamed whenever necessary.

Definition 4.2.2. For all simple types A and B the *substitution operator*

$$\_[\_ := \_]_{A,B} : \Lambda^{\rightarrow}_{B} \times \mathsf{Var}_A \times \Lambda^{\rightarrow}_{A} \to \Lambda^{\rightarrow}_{B}$$

is a mapping that is defined by induction on the definition of $\Lambda^{\rightarrow}_{B}$:

1. $x[x := s]_{A,B} = s$,
2. $y[x := s]_{A,B} = y$,
3. $f[x := s]_{A,B} = f$,
4. $(y.t_0)[x := s]_{A,B} = y.(t_0[x := s]_{A,B})$,
5. $(t_0 t_1)[x := s]_{A,B} = (t_0[x := s]_{A,B})(t_1[x := s]_{A,B})$.

It follows by induction on the derivation of $t \in \Lambda^{\rightarrow}_{B}$ that $t[x := s]_{A,B} \in \Lambda^{\rightarrow}_{B}$. In the sequel we mostly write $t[x := s]$ instead of $t[x := s]_{A,B}$ if the types are either clear from the context or irrelevant.

We now consider the rewrite relation of $\lambda^{\rightarrow}_{\overline{\eta}}$. It differs from the one of simply typed λ-calculus as presented in Chapter 2 since it is generated not only by the rewrite rule for β-reduction $(x.s)t \to_{\beta} s[x := t]$ but also by the rewrite rule for η-expansion, which is given as $s \to_{\overline{\eta}} x.sx$ provided $x \notin \mathsf{FV}(s)$.

The rewrite relation $\to_{\beta\overline{\eta}}$ on the set of preterms will be defined as the union of the relations $\to_{\beta}$ and $\to_{\overline{\eta}}$. The notions of redex, redex occurrence and rewrite step for β-reduction are defined in Definition 2.1.9 of Chapter 2. We now define the corresponding notions for $\overline{\eta}$-reduction.

Definition 4.2.3.

1. Let s be a simply typed λ-term. An *$\overline{\eta}$-redex occurrence in s* is a pair $(\phi, \overline{\eta})$ such that

   (a) $s|_\phi : A \to B$,
   (b) if $\phi = \chi\, 0$, then $\chi\, 1 \notin \mathsf{Pos}(s)$,
   (c) $s|_\phi$ is not of the form $x.s_0$.

2. An *$\overline{\eta}$-redex* is a pair $(s, (\phi, \overline{\eta}))$ such that $(\phi, \overline{\eta})$ is an $\overline{\eta}$-redex occurrence in s.

3. An *$\overline{\eta}$-rewrite step* is a triple $(s, (\phi, \overline{\eta}), s')$ such that

   (a) $(\phi, \overline{\eta})$ is an $\overline{\eta}$-redex occurrence in s,
   (b) $s' = s[\phi \leftarrow x.s_0 x]$ with $s_0 = s|_\phi$ and $x \notin \mathsf{FV}(s_0)$.

4. Both the $\overline{\eta}$-redex occurrence $(\phi, \overline{\eta})$ and the $\overline{\eta}$-redex $(s, (\phi, \overline{\eta}))$ are said to be *contracted* in the $\overline{\eta}$-rewrite step $(s, (\phi, \overline{\eta}), s')$.

The essential property of restricted η-expansion is that it doesn't create β-redex occurrences. This is guaranteed by the last two conditions in the definition of an $\overline{\eta}$-redex occurrence above. In the sequel we will suppose the conditions on $\overline{\eta}$-reduction to be verified without mentioning them explicitly.

Notation 4.2.4. An $\overline{\eta}$-rewrite step $(s, (\phi, \overline{\eta}), s')$ as in Definition 4.2.3 is usually denoted by $(\phi, \overline{\eta}) : s \to s'$ or by $s \xrightarrow{\phi}_{\overline{\eta}} s'$.

The set of simply typed λ-terms with $\overline{\eta}$-expansion has an underlying abstract rewriting system of the form $(\Lambda^{\to}, \to_{\overline{\eta}})$ with $s \to_{\overline{\eta}} s'$ if $s \xrightarrow{\phi}_{\overline{\eta}} s'$ for some $\phi \in \mathsf{Pos}(M)$.

Definition 4.2.5. The relation $\to_{\beta\overline{\eta}}$ on $\Lambda^{\to}$ is defined as the union of $\to_{\overline{\eta}}$ and $\to_\beta$.

The underlying abstract rewriting system of $\lambda^{\to}_{\overline{\eta}}$ is of the form $(\Lambda^{\to}, \to_{\beta\overline{\eta}})$ with $s \to_{\beta\overline{\eta}} s'$ if $s \xrightarrow{\phi}_\beta s'$ or $s \xrightarrow{\phi}_{\overline{\eta}} s'$ for some position $\phi \in \mathsf{Pos}(M)$. In order to give the underlying functional rewriting system of $\lambda^{\to}_{\overline{\eta}}$ we need the following definition.

Definition 4.2.6.

1. Let s be a preterm. The set of all β-redex occurrences and $\overline{\eta}$-redex occurrences in s is denoted by $\mathfrak{U}_{\beta\overline{\eta}}(s)$.

2. The set consisting of all pairs of the form (ϕ, β) and $(\phi, \overline{\eta})$ with ϕ a position is denoted by $\mathfrak{U}_{\beta\overline{\eta}}$.

The calculus $\lambda_{\overline{\eta}}^{\rightarrow}$ is a functional rewriting system of the form $(\Lambda^{\rightarrow}, \mathfrak{U}_{\beta\overline{\eta}}, \rightarrow)$, with $\rightarrow(\phi, \beta)(s) = s'$ if $s \xrightarrow{\phi}_{\beta} s'$ and $\rightarrow(\phi, \beta)(s)$ being undefined otherwise, and further $\rightarrow(\phi, \overline{\eta})(s) = s'$ if $s \xrightarrow{\phi}_{\overline{\eta}} s'$ and $\rightarrow(\phi, \overline{\eta})(s)$ being undefined otherwise.

We conclude the presentation of $\lambda_{\overline{\eta}}^{\rightarrow}$ by making an observation concerning $\overline{\eta}$-normal forms that will be used in the sequel.

Remark 4.2.7. Recall the definition of the arity of a type, given in Definition 2.5.2 of Chapter 2. In an $\overline{\eta}$-normal form, a variable or constant of type A has exactly $\mathsf{ar}(A)$ arguments. We will make use of this simple observation in some proofs.

Example 4.2.8. The $\overline{\eta}$-normal form of a constant f of type $A_1 \rightarrow \ldots \rightarrow A_m \rightarrow B$ with B a base type is $x_1 \ldots x_m . f s_1 \ldots s_m$, with $s_1, \ldots, s_m$ the $\overline{\eta}$-normal forms of $x_1, \ldots, x_m$.

$\lambda_{\overline{\eta}}^{\rightarrow}$ is a Substitution Calculus. The next thing to be done is to show that the calculus $\lambda_{\overline{\eta}}^{\rightarrow}$ has all the properties a substitution calculus should satisfy, according to the general definition of a substitution calculus given in Definition 4.1.1 of Section 4.1.

A useful property is the following. It expresses that the set of $\overline{\eta}$-normal forms is closed under substitution and β-reduction.

Proposition 4.2.9.

1. *If t and s are both in $\overline{\eta}$-normal form, then $t[x := s]$ is in $\overline{\eta}$-normal form.*
2. *If t is in $\overline{\eta}$-normal form and $t \rightarrow_{\beta} t'$, then t' is in $\overline{\eta}$-normal form.*

Proof. The first point follows by induction on the structure of t and the second point follows by the first point. □

Proposition 4.2.10. *$\lambda_{\overline{\eta}}^{\rightarrow}$ is strongly normalising and confluent.*

Proof. A proof can be found at several places in the literature, for instance in [Čub93] , [Aka93] and [DCK94]. □

The second and third condition on the subterm $s|_{\phi}$ in the definition of an $\overline{\eta}$-redex occurrence, given in Definition 4.2.3, are essential for strong normalisation. This is illustrated by the following two examples. First, $xz \rightarrow_{\overline{\eta}} (y.xy)z \rightarrow_{\beta} xz$ with $x : 0 \rightarrow 0$, $y : 0$ and $z : 0$. Second, $x.x \rightarrow_{\overline{\eta}} y.(x.x)y \rightarrow_{\beta} y.y$ with $x : 0$ and $y : 0$.

Note that every type is inhabited and that every preterm is typed. We now verify that $\lambda_{\overline{\eta}}^{\rightarrow}$ satisfies the property of subject conversion. The condition concerning type conversion is trivially satisfied.

Proposition 4.2.11. *If $s : A$ and $s \leftrightarrow_{\beta\overline{\eta}} s'$, then $s' : A$.*

Proof.

1. Suppose $(\phi, \beta) : s \to s'$. So $s|_\phi = (x.s_0)s_1$ and $s' = s[\phi \leftarrow s_0[x := s_1]]$.

   Suppose $s : A$. We have $(x.s_0)s_1 : B$ for some type B. Then $s_0 : B$, $x : C$ and $s_1 : C$ for some type C. In that case also $s_0[x := s_1] : B$ and hence $s' : A$.

   Suppose $s' : A$. We have $s_0[x := s_1] : B$ for some type B. Moreover, we have $x : C$ and $s_1 : C$ for some type C. So $(x.s_0)s_1 : B$ and hence $s : A$.

2. Suppose $(\phi, \overline{\eta}) : s \to s'$. So $s|_\phi = s_0$ and $s' = s[\phi \leftarrow x.s_0x]$.

   Suppose $s : A$. We have $s|_\phi = s_0$ and $s_0 : B$ for some type $B = B_0 \to B_1$. Then $x.s_0x : B$ and hence $s : A$.

   Suppose $s' : A$. Then $x.s_0x : B$ for some type B. We have $B = B_0 \to B_1$ for some types B_0, B_1. Then $s_0 : B$ and hence $s' : A$. □

Here we make essential use of the fact that we consider typed terms, with variables typed a priori, and a typed substitution operator. So in $t[x := s]_{A,B}$, we have $t : B$, $x : A$ and $s : A$. Typable terms are not closed under β-expansion, since for instance $x[y := (z.zz)(z.zz)]$ is typable but $(y.x)(z.zz)(z.zz)$ isn't.

The relation $\leftrightarrow_{\beta\overline{\eta}}$ and the operator for substitution are shown to interact properly in the following proposition.

Proposition 4.2.12. *Let $s : A$ and $t : B$.*

1. *If $s \leftrightarrow^*_{\beta\overline{\eta}} s'$ then $t[x := s] \leftrightarrow^*_{\beta\overline{\eta}} t[x := s']$.*
2. *If $t \leftrightarrow^*_{\beta\overline{\eta}} t'$ then $t[x := s] \leftrightarrow^*_{\beta\overline{\eta}} t'[x := s]$.*

Proof. Let $s : A$ and $t : B$.

1. Suppose $(\phi, \beta) : s \to s'$. It is shown by an easy induction on the definition of $\Lambda^{\to}_{\mathcal{B}}$ that $t[x := s] \twoheadrightarrow_\beta t[x := s']$.

   Suppose $(\phi, \overline{\eta}) : s \to s'$. It is shown by induction on the structure of t that $t[x := s] \leftrightarrow^*_{\beta\overline{\eta}} t[x := s']$.

   (a) Suppose $t = y_1. \ldots .y_m.zt_1 \ldots t_n$ with $z \neq x$. By induction hypothesis we have that $t_p[x := s] \leftrightarrow^*_{\beta\overline{\eta}} t_p[x := s']$ for every $p \in \{1, \ldots, n\}$. Hence we have $t[x := s] \leftrightarrow^*_{\beta\overline{\eta}} t[x := s']$.

   (b) Suppose $t = y_1. \ldots .y_m.ft_1 \ldots t_n$. This case is entirely similar to the previous case.

   (c) Suppose $t = y_1.\ldots.y_m.xt_1\ldots t_n$. If we have $(\epsilon,\overline{\eta}) : s \rightarrow s'$ and $n \geq 0$, then $y_1.\ldots.y_m.s't_1\ldots t_n \rightarrow_\beta y_1.\ldots.y_m.st_1\ldots t_n$. Otherwise, we have $y_1.\ldots.y_m.st_1\ldots t_n \rightarrow_{\overline{\eta}} y_1.\ldots.y_m.s't_1\ldots t_n$. We conclude that $t[x := s] \leftrightarrow^*_{\beta\overline{\eta}} t[x := s']$.

2. Suppose $t \rightarrow_\beta t'$. By the substitution lemma, $((y.t_0)t_1)[x := s] \rightarrow_\beta (t_0[y := t_1])[x := s]$. This yields that $t[x := s] \rightarrow_\beta t'[x := s]$ if $t \rightarrow_\beta t'$.

   Suppose $t \rightarrow_{\overline{\eta}} t'$. We prove by induction on the structure of t that $t[x := s] \leftrightarrow^*_{\beta\overline{\eta}} t'[x := s]$.

   (a) Suppose $t = y_1.\ldots.y_m.ft_1\ldots t_n$ or $t = y_1.\ldots.y_m.zt_1\ldots t_n$ with $z \neq x$. We treat only the case $t = y_1.\ldots.y_m.ft_1\ldots t_n$ since in the other case the reasoning is exactly the same. There are two possibilities.

      i. Suppose $t' = y_1.\ldots.y_m.ft_1\ldots t'_p\ldots t_n$ with $t_p \rightarrow_{\overline{\eta}} t'_p$. By the induction hypothesis, we have that $t_p[x := s] \leftrightarrow^*_{\beta\overline{\eta}} t'_p[x := s]$. This yields that $t[x := s] \leftrightarrow^*_{\beta\overline{\eta}} t'[x := s]$.

      ii. Suppose $t' = y_1.\ldots.y_m.z'.ft_1\ldots t_nz'$. Then we clearly have $t[x := s] \leftrightarrow^*_{\beta\overline{\eta}} t'[x := s]$.

   (b) Suppose $t = y_1.\ldots.y_m.xt_1\ldots t_n$. Two cases are distinguished.

      i. Suppose $t' = y_1.\ldots, y_m.xt_1\ldots t'_p\ldots t_n$ with $t_p \rightarrow_{\overline{\eta}} t'_p$. Then $t[x := s] \leftrightarrow^*_{\beta\overline{\eta}} t'[x := s]$.

      ii. Suppose $t' = y_1.\ldots.y_m.z'.xt_1\ldots t_nz'$. Then we have either $y_1.\ldots.y_m.st_1\ldots t_n \rightarrow_{\overline{\eta}} y_1.\ldots.y_m.z'.st_1\ldots t_n$ or, if s is of the form $z_1.\ldots.z_{n+1}.s_0$, $y_1.\ldots.y_m.z'.st_1\ldots t_n \leftrightarrow^*_\beta y_1.\ldots.y_m.st_1\ldots t_n$. We have in both cases that $t[x := s] \leftrightarrow^*_{\beta\overline{\eta}} t'[x := s]$. □

We conclude that the calculus $\lambda^{\rightarrow}_{\overline{\eta}}$ is an adequate substitution calculus in the sense that is satisfies all requirements of a substitution calculus as formulated in Definition 4.1.1.

As discussed in Section 4.1, for the definition of a descendant relation of a higher-order rewriting system we will need the descendant relation of its substitution calculus. In Definition 2.1.13 of Chapter 2, the definition of the descendant relation Des_β for λ-calculus with β-reduction is given. By Proposition 4.2.9, we need only Des_β if all preterms we consider are in $\overline{\eta}$-normal form.

Notation 4.2.13. Let $\sigma : s \twoheadrightarrow_\beta s'$ be a rewrite sequence to β-normal form. The set of descendants of ϕ in s' is usually denoted by $\mathsf{Des}_\beta(s,\sigma)(\phi)$.

Note that for every σ and σ' such that $\sigma : s \twoheadrightarrow_\beta s'$ and $\sigma' : s \twoheadrightarrow_\beta s'$ with s' a β-normal form, we have that $\mathsf{Des}_\beta(s,\sigma)(\phi) = \mathsf{Des}_\beta(s,\sigma')(\phi)$. We will use the descendant relation to trace constants in λ-calculus. We won't trace bound variables.

Higher-Order Rewriting Systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as Substitution Calculus.
In the remainder of this section, we will suppose all preterms to be in $\overline{\eta}$-normal form.

In this paragraph we define higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus. A higher-order rewriting system with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus will be defined as a triple consisting of the substitution calculus $\lambda_{\overline{\eta}}^{\rightarrow}$, a rewrite alphabet consisting of simply typed constants called rewrite operators, and a set of rewrite rules. So the difference with the general definition as presented in Section 4.1 is that in this case there is also a rewrite alphabet. Intuitively, the rewrite alphabet consists of constants whose operational semantics is defined by the rewrite rules.

Definition 4.2.14. A *rewrite alphabet for* $\lambda_{\overline{\eta}}^{\rightarrow}$ is a set, denoted by $\mathcal{A}$, consisting of simply typed constants that are called *rewrite operators*. Rewrite operators are denoted by $f, g, h, \ldots$.

Definition 4.2.15. Let $\mathcal{A}$ be a rewrite alphabet for $\lambda_{\overline{\eta}}^{\rightarrow}$. A *rewrite rule over* $\mathcal{A}$ *of type* A is a pair of terms $(\mathtt{l}, \mathtt{r})$ of type A such that

1. $\mathtt{l}$ and $\mathtt{r}$ are closed,

2. all constants in $\mathtt{l}$ and in $\mathtt{r}$ are in $\mathcal{A}$.

Rewrite rules are denoted by $\mathtt{l} \rightarrow \mathtt{r}, \mathtt{g} \rightarrow \mathtt{d}, \ldots$.

Example 4.2.16. Consider a set $\mathcal{A}$ consisting of four simply typed constants: $f : 0$, $f' : 0$, $g : 0 \rightarrow 0$, and $g' : 0 \rightarrow 0$. Then $\mathcal{A}$ is a rewrite alphabet for $\lambda_{\overline{\eta}}^{\rightarrow}$.

An example of a rewrite rule for $\mathcal{A}$ of type 0 is $f \rightarrow f'$. An example of a rewrite rule for $\mathcal{A}$ of type $0 \rightarrow 0$ is $x.gx \rightarrow x.g'x$.

The pair $gx \rightarrow f$ is not a rewrite rule since gx is not a closed term. The pair $f \rightarrow h$ is not a rewrite rule for $\mathcal{A}$ since $h \notin \mathcal{A}$.

Definition 4.2.17. A *higher-order rewriting system with* $\lambda_{\overline{\eta}}^{\rightarrow}$ *as substitution calculus* is a triple $(\lambda_{\overline{\eta}}^{\rightarrow}, \mathcal{A}, \mathcal{R})$ with $\mathcal{A}$ a rewrite alphabet for $\lambda_{\overline{\eta}}^{\rightarrow}$ and $\mathcal{R}$ a set of rewrite rules over $\mathcal{A}$.

Definition 4.2.18. Let $\mathcal{H} = (\lambda_{\overline{\eta}}^{\rightarrow}, \mathcal{A}, \mathcal{R})$ be a higher-order rewriting system. A *term* of $\mathcal{H}$ is a preterm that is in normal form with respect to $\rightarrow_{\beta\overline{\eta}}$. Terms are like preterms denoted by $s, t, r, \ldots$. The set of terms is denoted by Terms.

We give the standard example in higher-order rewriting: the untyped λ-calculus.

Example 4.2.19. Untyped λ-calculus is a higher-order rewriting system in the following way. There is one base type, denoted by 0, that represents the set of terms. The alphabet contains two rewrite operators, one for abstraction and one for application:

$$\begin{aligned} \mathsf{abs} &: (0 \to 0) \to 0 \\ \mathsf{app} &: 0 \to 0 \to 0 \end{aligned}$$

The rewrite rules for β- and η-reduction are:

$$\begin{aligned} z.z'.\mathsf{app}(\mathsf{abs}(x.zx))z' &\to_{beta} z.z'.zz' \\ z.\mathsf{abs}(x.\mathsf{app}zx) &\to_{eta} z.z \end{aligned}$$

The rewrite rules of a higher-order rewriting system induce a rewrite relation on the set of terms. First we consider the definition of a relation on the set of terms induced by the rewrite rules, that is precisely an instance of the definition of the rewrite relation for the general case, given in Definition 4.1.5 of Section 4.1. This relation is called the pre-rewrite relation. The rewrite relation that we will use in the sequel is a restricted version of the pre-rewrite relation, and will be defined in Definition 4.2.25.

Definition 4.2.20. Let $\mathcal{H} = (\lambda_{\overline{\eta}}^{\to}, \mathcal{A}, \mathcal{R})$ be a higher-order rewriting system. The *pre-rewrite relation* on the set Terms, denoted by $\to_{\mathcal{R}}^{p}$, is defined as follows:

$$s \to_{\mathcal{R}}^{p} s'$$

if

$$s = t[x := \mathrm{l}]\downarrow_{SC} \quad \text{and} \quad s' = t[x := \mathrm{r}]\downarrow_{SC}$$

for a term t, a variable x and a rewrite rule $\mathrm{l} \to \mathrm{r}$.

As remarked in Section 4.1, a rewrite step $s \to_{\mathcal{R}} s'$ as in the previous definition can be decomposed as an expansion in $\lambda_{\overline{\eta}}^{\to}$, followed by a replacement of the right-hand side for the left-hand side of a rewrite rule, followed by a reduction in $\lambda_{\overline{\eta}}^{\to}$: $s \;{}_{\beta\overline{\eta}}\!\twoheadleftarrow t[x := \mathrm{l}] \rightsquigarrow t[x := \mathrm{r}] \twoheadrightarrow_{\beta\overline{\eta}} s'$. Since the left- and right-hand side of a rewrite rule have the same type, $t[x := \mathrm{r}]$ is well-formed if $t[x := \mathrm{l}]$ is.

Example 4.2.21. Consider the higher-order rewriting system $\mathcal{H} = (\lambda_{\overline{\eta}}^{\to}, \mathcal{A}, \mathcal{R})$ with $\mathcal{A} = \{f : 0, f' : 0\}$ and $\mathcal{R}$ consisting only of the rewrite rule $f \to f'$. The pre-rewrite relation $\to_{\mathcal{R}}^{p}$ is illustrated by the following.

1. We have $y \to_{\mathcal{R}}^{p} y$ since $y = (y[x := f])\downarrow_{\beta\overline{\eta}}$ and $y = (y[x := f'])\downarrow_{\beta\overline{\eta}}$.

2. We have $f \to_{\mathcal{R}}^{p} f'$ since $f = (x[x := f])\downarrow_{\beta\overline{\eta}}$ and $f' = (x[x := f'])\downarrow_{\beta\overline{\eta}}$.

3. We have $yff \rightarrow^{p}_{\mathcal{R}} yf'f'$ since $yff = ((yxx)[x := f])\downarrow_{\beta\bar{\eta}}$ and $yf'f' = ((yxx)[x := f'])\downarrow_{\beta\bar{\eta}}$.

This example illustrate two properties of the pre-rewrite relation that are not desirable for a basic definition of a rewrite relation.

First, we have $s \rightarrow^{p}_{\mathcal{R}} s$ for every term s, as illustrated in the first part of Example 4.2.21. This is a serious shortcoming of the pre-rewrite relation, since it means that we will never have strong normalisation.

Second, it is possible that several occurrences of the left-hand side of a rule are replaced by the right-hand side of a rule, in the replacement part of a rewrite step. This is illustrated in the third part of Example 4.2.21. Replacing more than one occurrence of a left-hand side can be useful, in order to increase the efficiency of rewriting for instance, however it is not the most basic form of rewriting.

The usual definition of a rewrite relation doesn't have these two properties.

For these reasons, we define the rewrite relation as a restricted version of the pre-rewrite relation. First a definition is needed. We suppose that for every type A there is a distinguished variable of type A that is denoted by $\square$. It is distinguished in the sense that it never occurs bound.

Definition 4.2.22. A *precontext* of type A is a preterm with exactly one occurrence of $\square : A$. A *context* of type A is a term with exactly one occurrence of $\square : A$. A (pre)context of type A in which the distinguished variable $\square : A$ occurs at position ϕ is denoted by $C\square_{\phi}$.

We define the replacement of the occurrence of $\square$ in a context by preterm.

Definition 4.2.23. Let $s : A$ be a preterm and let $C\square_{\phi}$ be a context of type A. The preterm $C[s]_{\phi}$ is defined as

$$C[s]_{\phi} = C\square_{\phi}[\phi \leftarrow s].$$

So contexts just serve to denote the notion of replacement $[\phi \leftarrow t]$ in a different way. Often the type of a context is not mentioned explicitly, and often we do not mention the position where $\square$ occurs.

Example 4.2.24. The term $C\square_{0} = x.\square$ with $\square : A$ is a context of type A. If moreover $x : A$, then $C[x]_{0} = x.x$.

Note that if s is a closed preterm and t is a preterm containing exactly one occurrence of x at position ϕ, then there is no difference between a substitution $t[x := s]$ and a replacement $t[\phi \leftarrow s]$.

We now give the definition of the rewrite relation that will be used in the sequel.

Definition 4.2.25. Let $\mathcal{H} = (\lambda^{\rightarrow}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$ be a higher-order rewriting system.

1. Let s be a term. A *redex occurrence in* s is a pair $(C\Box, \mathrm{l} \to \mathrm{r})$ such that $s = C[\mathrm{l}]{\downarrow_{\beta\overline{\eta}}}$.
2. A *redex* is a pair $(s, (C\Box, \mathrm{l} \to \mathrm{r}))$ such that $(C\Box, \mathrm{l} \to \mathrm{r})$ is a redex occurrence in s.
3. A *rewrite step* is a triple $(s, (C\Box, \mathrm{l} \to \mathrm{r}), s')$ such that
   (a) $s = C[\mathrm{l}]{\downarrow_{\beta\overline{\eta}}}$,
   (b) $s' = C[\mathrm{r}]{\downarrow_{\beta\overline{\eta}}}$.
4. Both the redex occurrence $(C\Box, \mathrm{l} \to \mathrm{r})$ and the redex $(s, (C\Box, \mathrm{l} \to \mathrm{r}))$ are said to be *contracted* in the rewrite step $(s, (C\Box, \mathrm{l} \to \mathrm{r}), s')$.

Notation 4.2.26. A rewrite step $(s, (C\Box, \mathrm{l} \to \mathrm{r}), s')$ as in Definition 4.2.25 is usually denoted as $(C\Box, \mathrm{l} \to \mathrm{r}) : s \to s'$.

Definition 4.2.27. Let $\mathcal{H} = (\lambda^{\rightarrow}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$ be a higher-order rewriting system.

1. Let s be a term. The set of redex occurrences in s is denoted by $\mathfrak{U}(s)$.
2. The set of consisting of all pairs of a context and a rewrite rule of the same type is denoted by $\mathfrak{U}_{\mathcal{R}}$. Elements of $\mathfrak{U}_{\mathcal{R}}$ are denoted by $u, v, w, \ldots$.

Example 4.2.28. Consider again the higher-order rewriting system defined in Example 4.2.21.

1. The term y cannot be rewritten.
2. We have $f \to_{\mathcal{R}} f'$ since $f = f$ and $f' = f'$, hence we have an instance of Definition 4.2.25 with $C\Box = \Box$.
3. We have $yff \to_{\mathcal{R}} yf'f$ and $yff \to_{\mathcal{R}} yff'$. It is not the case that $yff \to_{\mathcal{R}} yf'f'$.

The underlying abstract rewriting system of a higher-order rewriting system is $(\mathsf{Terms}, \to_{\mathcal{R}})$ with $s \to_{\mathcal{R}} s'$ if there is a context $C\Box$ and a rewrite rule $\mathrm{l} \to \mathrm{r}$ such that $(C\Box, \mathrm{l} \to \mathrm{r}) : s \to s'$. Moreover, a higher-order rewriting system is a functional rewriting system of the form $(\mathsf{Terms}, \mathfrak{U}_{\mathcal{R}}, \to)$ with $\to(C\Box, \mathrm{l} \to \mathrm{r})(s) = s'$ if $(C\Box, \mathrm{l} \to \mathrm{r}) : s \to s'$ and $\to(C\Box, \mathrm{l} \to \mathrm{r})(s)$ being undefined otherwise.

A natural question is now whether expressivity of the rewrite relation is lost by restricting the replacement part of a rewrite step to a single replacement. Van Oostrom gives in Definition 3.1.21 of [Oos94] two conditions on a substitution calculus that imply that the pre-rewrite relation $\rightarrow^{p}_{\mathcal{R}}$ as defined in Definition 4.2.20 can be mimicked by a rewrite relation $\rightarrow_{\mathcal{R}}$ as defined in Definition 4.2.25. Because of the standardisation theorem, the calculus $\lambda^{\rightarrow}_{\overline{\eta}}$ satisfies these conditions. For instance, for the higher-order rewriting system defined in Example 4.2.21, we have $yff \rightarrow_{\mathcal{R}} yf'f \rightarrow_{\mathcal{R}} yf'f'$.

We give two more elaborate examples illustrating the rewrite relation of a higher-order rewriting system. The first one is concerned with the representation of a first-order term rewriting system in the framework of higher-order rewriting systems. This term rewriting system models addition and multiplication in a simple way. The second one is concerned with a 'real' higher-order example, modelling differentiation.

Example 4.2.29. Consider the first-order term rewriting system defined by the rewrite rules

$$\begin{array}{rcl} \mathsf{A}(\mathsf{0},x) & \rightarrow & x \\ \mathsf{A}(\mathsf{S}(x),y) & \rightarrow & \mathsf{S}(\mathsf{A}(x,y)) \\ \mathsf{M}(\mathsf{0},x) & \rightarrow & \mathsf{0} \\ \mathsf{M}(\mathsf{S}(x),y) & \rightarrow & \mathsf{A}(\mathsf{M}(x,y),y) \end{array}$$

We represent this term rewriting system in the format of higher-order rewriting systems. There is one base type, denoted by $\mathtt{N}$. It can be thought of as modelling the natural numbers. The alphabet contains rewrite operators $\mathsf{0} : \mathtt{N}, \mathsf{S} : \mathtt{N} \rightarrow \mathtt{N}, \mathsf{A} : \mathtt{N} \rightarrow \mathtt{N} \rightarrow \mathtt{N}, \mathsf{M} : \mathtt{N} \rightarrow \mathtt{N} \rightarrow \mathtt{N}$. The rewrite rules get the following form:

$$\begin{array}{rcl} x.\mathsf{A0}x & \rightarrow_{\mathtt{R}_1} & x.x \\ x.y.\mathsf{A}(\mathsf{S}x)y & \rightarrow_{\mathtt{R}_2} & x.y.\mathsf{S}(\mathsf{A}xy) \\ x.\mathsf{M0}x & \rightarrow_{\mathtt{R}_3} & x.\mathsf{0} \\ x.y.\mathsf{M}(\mathsf{S}x)y & \rightarrow_{\mathtt{R}_4} & x.y.\mathsf{A}(\mathsf{M}xy)y \end{array}$$

We give an example of a rewrite sequence. On the left, a rewrite sequence in the higher-order rewriting system is given, and on the right the corresponding rewrite sequence in the first-order term rewriting system is given.

$$\begin{array}{rcl}
\mathsf{M(S0)(A(S0)0)} & {}_{\beta\overline{\eta}}\twoheadleftarrow & \mathsf{M(S(0),A(S(0),0))} \\
(x.y.\mathsf{M}(\mathsf{S}x)y)(\mathsf{0})(\mathsf{A(S0)0}) & \rightarrow & \rightarrow \\
(x.y.\mathsf{A}(\mathsf{M}xy)y)(\mathsf{0})(\mathsf{A(S0)0}) & \twoheadrightarrow_{\beta\overline{\eta}} & \\
\mathsf{A(M0(A(S0)0))(A(S0)0)} & {}_{\beta\overline{\eta}}\twoheadleftarrow & \mathsf{A(M(0,A(S(0),0)),A(S(0),0))} \\
\mathsf{A}((x.\mathsf{M0}x)(\mathsf{A(S0)0}))(\mathsf{A(S0)0}) & \rightarrow & \rightarrow \\
\mathsf{A}((x.\mathsf{0})(\mathsf{A(S0)0}))(\mathsf{A(S0)0}) & \twoheadrightarrow_{\beta\overline{\eta}} & \\
\mathsf{A0(A(S0)0)} & {}_{\beta\overline{\eta}}\twoheadleftarrow & \mathsf{A(0,A(S(0),0))} \\
(x.\mathsf{A0}x)(\mathsf{A(S0)0}) & \rightarrow & \rightarrow \\
(x.x)(\mathsf{A(S0)0}) & \twoheadrightarrow_{\beta\overline{\eta}} & \\
\mathsf{A(S0)0} & {}_{\beta\overline{\eta}}\twoheadleftarrow & \mathsf{A(S(0),0)} \\
(x.y.\mathsf{A}(\mathsf{S}x)y)(\mathsf{0})(\mathsf{0}) & \rightarrow & \rightarrow \\
(x.y.\mathsf{S}(\mathsf{A}xy))(\mathsf{0})(\mathsf{0}) & \twoheadrightarrow_{\beta\overline{\eta}} & \\
\mathsf{S(A00)} & {}_{\beta\overline{\eta}}\twoheadleftarrow & \mathsf{S(A(0,0))} \\
\mathsf{S}((x.\mathsf{A0}x)(\mathsf{0})) & \rightarrow & \rightarrow \\
\mathsf{S}((x.x)(\mathsf{0})) & \twoheadrightarrow_{\beta\overline{\eta}} & \\
\mathsf{S0} & & \mathsf{S(0)}
\end{array}$$

The following redexes are contracted in the rewrite sequence:

$$\begin{array}{l}
(\square\mathsf{0(A(S0)0)}, \mathrm{R}_4) \\
(\mathsf{A}(\square(\mathsf{A(S0)0}))(\mathsf{A(S0)0}), \mathrm{R}_3) \\
(\square(\mathsf{A(S0)0}), \mathrm{R}_1) \\
(\square\mathsf{00}, \mathrm{R}_2) \\
(\mathsf{S}(\square\mathsf{0}), \mathrm{R}_1)
\end{array}$$

Example 4.2.30. We consider a higher-order rewriting system $(\lambda^{\rightarrow}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$ that models some aspects of differentiation. There is one base type, denoted by R, that models the set of real numbers. Let $\mathcal{A}$ be a rewrite alphabet consisting of the following rewrite operators:

$$\begin{array}{rcl}
\mathsf{dif} & : & (\mathrm{R} \rightarrow \mathrm{R}) \rightarrow (\mathrm{R} \rightarrow \mathrm{R}) \\
; & : & (\mathrm{R} \rightarrow \mathrm{R}) \rightarrow (\mathrm{R} \rightarrow \mathrm{R}) \rightarrow (\mathrm{R} \rightarrow \mathrm{R}) \\
\mathsf{sin} & : & \mathrm{R} \rightarrow \mathrm{R} \\
\mathsf{cos} & : & \mathrm{R} \rightarrow \mathrm{R} \\
\times & : & \mathrm{R} \rightarrow \mathrm{R} \rightarrow \mathrm{R} \\
\mathsf{min} & : & \mathrm{R} \rightarrow \mathrm{R}
\end{array}$$

The set $\mathcal{R}$ consists of the following rewrite rules over $\mathcal{A}$:

$$\begin{array}{rcl}
y.\mathsf{dif}(x.\mathsf{sin}x)y & \rightarrow & y.\mathsf{cos}y \\
y.\mathsf{dif}(x.\mathsf{cos}x)y & \rightarrow & y.\mathsf{min}(\mathsf{sin}y) \\
z.z'.y.\mathsf{dif}(x.(z;z')x)y & \rightarrow & z.z'.y.\mathsf{dif}(x.zx)(z'y)\times\mathsf{dif}(x.z'x)y
\end{array}$$

Let $\pi : \mathtt{R}$ be a rewrite operator. An example of a rewrite sequence is the following:

$$\begin{array}{rl}
\mathsf{dif}(x.(\mathsf{sin};\mathsf{cos})x)\pi & {}_{\beta\overline{\eta}}\twoheadleftarrow \\
(z.z'.y.\mathsf{dif}(x.(z;z')x)y)\mathsf{sin}\ \mathsf{cos}\,\pi & \rightarrow \\
(z.z'.y.\mathsf{dif}(x.zx)(z'y) \times \mathsf{dif}(x.z'x)y)\mathsf{sin}\ \mathsf{cos}\,\pi & \twoheadrightarrow_{\beta\overline{\eta}} \\
\mathsf{dif}(x.\mathsf{sin}x)(\mathsf{cos}\pi) \times \mathsf{dif}(x.\mathsf{cos}x)\pi & {}_{\beta\overline{\eta}}\twoheadleftarrow \\
(y.\mathsf{dif}(x.\mathsf{sin}x)y)(\mathsf{cos}\pi) \times \mathsf{dif}(x.\mathsf{cos}x)\pi & \rightarrow \\
(y.\mathsf{cos}y)(\mathsf{cos}\pi) \times \mathsf{dif}(x.\mathsf{cos}x)\pi & \twoheadrightarrow_{\beta\overline{\eta}} \\
\mathsf{cos}(\mathsf{cos}\pi) \times \mathsf{dif}(x.\mathsf{cos}x)\pi & {}_{\beta\overline{\eta}}\twoheadleftarrow \\
\mathsf{cos}(\mathsf{cos}\pi) \times (y.\mathsf{dif}(x.\mathsf{cos}x)y)\pi & \rightarrow \\
\mathsf{cos}(\mathsf{cos}\pi) \times (y.\mathsf{min}(\mathsf{sin}y))\pi & \twoheadrightarrow_{\beta\overline{\eta}} \\
\mathsf{cos}(\mathsf{cos}\pi) \times \mathsf{min}(\mathsf{sin}\pi) &
\end{array}$$

Finally, we give the definition of a descendant relation tracing positions in terms for a higher-order rewriting system. It is defined using the descendant relation Des_β for β-reduction.

Definition 4.2.31. Let $\mathcal{H} = (\lambda_{\overline{\eta}}^{\rightarrow}, \mathcal{A}, \mathcal{R})$ be a higher-order rewriting system. Let P be the property on $\mathsf{Pos} \times \mathsf{Terms}$ defined by $P(\phi, s)$ if and only if $\phi \in \mathsf{Pos}(s)$. A descendant relation

$$\mathsf{Des} : \mathsf{Terms} \times \mathfrak{U}_{\mathcal{R}} \times \mathsf{Pos} \rightarrow \mathcal{P}(\mathsf{Pos})$$

for P is defined as follows. Let $(C\Box, \mathtt{l} \rightarrow \mathtt{r}) : s \rightarrow s'$ be a rewrite step. Let $\sigma : C[\mathtt{l}] \twoheadrightarrow_{\beta\overline{\eta}} s$ and $\sigma' : C[\mathtt{r}] \twoheadrightarrow_{\beta\overline{\eta}} s'$. Then

$$\mathsf{Des}(s, (C\Box, \mathtt{l} \rightarrow \mathtt{r}))(\phi) = \mathsf{Des}_{\beta\overline{\eta}}(C[\mathtt{r}], \sigma')(\phi')$$

if $\phi \in \mathsf{Des}_{\beta\overline{\eta}}(C[\mathtt{l}], \sigma)(\phi')$ for a position ϕ not in $\mathtt{l}$.

Pattern Higher-Order Rewriting Systems. In this paragraph we will discus an important subclass of the higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus: the pattern higher-order rewriting systems. The restriction to pattern systems makes the rewrite relation decidable. It excludes for instance a rewrite rule of the form $x \rightarrow f$, which, so far, was allowed provided x and f have the same type. Moreover, it permits to give another formalisation of the notions of redex occurrence and redex than the one given in Definition 4.2.25. This second formalisation will be used in Chapter 6.

Definition 4.2.32. A term s is a *pattern* if the following condition is satisfied: every free variable x in s occurs in a subterm of the form $xs_1 \ldots s_m$ for some $m \geq 0$, such that $s_1, \ldots, s_m$ are η-equivalent to distinct variables that are bound in s,

Example 4.2.33. Let $f : 0 \to 0$ and $g : (0 \to 0) \to 0$ be rewrite operators. The terms fx, $x.yx$ and $g(x.yx)$ with $x : 0$ and $y : 0 \to 0$ are patterns. The term yx is not a pattern since the free variable y has a free variable as argument. The term $f(yx)$ is not a pattern for the same reason. Finally, gy is not a pattern since it is not a term.

Definition 4.2.34. A *rule-pattern* is a closed term $\mathtt{l} = x_1.\ldots.x_m.fs_1\ldots s_n$ with $m, n \geq 0$, such that

1. $s_1, \ldots, s_n$ are patterns,
2. every variable x_p such that $x_p \in \{x_1, \ldots, x_m\}$ occurs at least once free in the subterm $fs_1 \ldots s_n$.

We say that f is the *head-symbol* of $\mathtt{l} = x_1.\ldots.x_m.fs_1\ldots s_n$.

Example 4.2.35. The left-hand side of every rule of the higher-order rewriting system in Example 4.2.29 is a rule-pattern. The term $y.x.yx$ with $x : 0$ and $y : 0 \to 0$ is a pattern but not a rule-pattern since it doesn't have a head-symbol.

Definition 4.2.36. Let $\mathcal{A}$ be a rewrite alphabet for $\lambda^{\to}_{\overline{\eta}}$. A *pattern rewrite rule of type A* is a pair of closed terms of type A, written as $\mathtt{l} \to \mathtt{r}$, such that

1. $\mathtt{l}$ is a rule-pattern of the form $x_1 \ldots x_m.fs_1 \ldots s_n$,
2. $\mathtt{r}$ is a term of the form $x_1 \ldots x_m.t$.

Definition 4.2.37. A higher-order rewriting system $\mathcal{H} = (\lambda^{\to}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$ is said to be a *pattern higher-order rewriting system* if every rewrite rule is a pattern rewrite rule.

Two important properties of pattern higher-order rewriting systems are the following. First, Miller has proved in [Mil91] that it is decidable whether two patterns can be unified. Moreover, a most general unifier can be computed. In fact, he uses a definition that is slightly more general than the one that has been given in Definition 4.2.32. Two λ-terms s and t are said to be unifiable if there is a mapping θ that assigns λ-terms to variables that is extended to a homomorphism such that $(s^\theta)\downarrow_{\beta\overline{\eta}} = (t^\theta)\downarrow_{\beta\overline{\eta}}$. The unification algorithm by Miller has been simplified by Nipkow in [Nip93a]. Since a rule-pattern is a pattern, we have by the decidability of unification that the rewrite relation of a pattern higher-order rewriting system is decidable. The restriction to pattern rewrite rules is very usual. Nipkow makes a restriction to pattern rewrite rules in [Nip91] and CRSs as defined in [Klo80] also have pattern rewrite rules.

We will now present a second formalisation of the notions redex occurrence and redex. This formalisation is well-defined by the following proposition, which is Proposition 3.2.17 in [Oos94].

Proposition 4.2.38. *Let $\mathcal{H} = (\lambda^{\rightarrow}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$ be a pattern higher-order rewriting system. Let* $\mathtt{l} \to \mathtt{r}$ *be a rewrite rule and let s be a term.*

1. *Let $C\Box$ be a context such that $C[\mathtt{l}] \twoheadrightarrow s$. Let ϕ' be the position of the head-symbol of* $\mathtt{l}$ *in $C[\mathtt{l}]$. Then there is a unique position $\phi \in \mathsf{Pos}(s)$ such that for every rewrite sequence $\sigma : C[\mathtt{l}] \twoheadrightarrow_{\beta\overline{\eta}} s$ we have that $\phi \in \mathsf{Des}_{\beta\overline{\eta}}(C[\mathtt{l}]_\chi, \sigma)(\phi')$.*
2. *Let $\phi \in \mathsf{Pos}(s)$. There is at most one position χ and one context $C\Box_\chi$ such that $\phi \in \mathsf{Des}_{\beta\overline{\eta}}(C[\mathtt{l}]_\chi, \sigma)(\phi')$ with ϕ' the position of the head-symbol of* $\mathtt{l}$ *in $C[\mathtt{l}]_\chi$.*

Note that by the second part of Proposition 4.2.38, we have that the context $C\Box_\phi$ in Definition 4.2.25 is unique. Now an alternative way to define the notions of redex, redex occurrences and rewrite step is as follows.

Definition 4.2.39. Let $\mathcal{H} = (\lambda^{\rightarrow}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$ be a pattern higher-order rewriting system.

1. Let s be a term. A *redex occurrence in s* is a pair $(\phi, \mathtt{l} \to \mathtt{r})$ such that there is a context $C\Box$ and a rewrite rule $\mathtt{l} \to \mathtt{r}$ with
   (a) $\sigma : C[\mathtt{l}] \twoheadrightarrow_{\beta\overline{\eta}} s$,
   (b) $\phi \in \mathsf{Des}_{\beta\overline{\eta}}(C[\mathtt{l}], \sigma)(\phi')$, with ϕ' the position of the head-symbol of $\mathtt{l}$ in $C[\mathtt{l}]$.
2. A *redex* is a pair $(s, (\phi, \mathtt{l} \to \mathtt{r}))$ such that $(\phi, \mathtt{l} \to \mathtt{r})$ is a redex occurrence in s.
3. A *rewrite step* is a triple $(s, (\phi, \mathtt{l} \to \mathtt{r}), s')$ such that
   (a) $s = C[\mathtt{l}]{\downarrow}_{\beta\overline{\eta}}$,
   (b) $s' = C[\mathtt{r}]{\downarrow}_{\beta\overline{\eta}}$,
   (c) if $\sigma : C[\mathtt{l}] \twoheadrightarrow_{\beta\overline{\eta}} s$, then $\phi \in \mathsf{Des}_{\beta\overline{\eta}}(C[\mathtt{l}], \sigma)(\phi')$ where ϕ' is the position of the head-symbol of $\mathtt{l}$ in $C[\mathtt{l}]$.

Notation 4.2.40. A rewrite step $(s, (\phi, \mathtt{l} \to \mathtt{r}), s')$ as in Definition 4.2.39 is usually denoted by $(\phi, \mathtt{l} \to \mathtt{r}) : s \to s'$ or by $s \overset{(\phi,\mathtt{l}\to\mathtt{r})}{\to_{\mathcal{R}}} s'$.

If pattern higher-order rewriting systems are considered, one can choose which definition of redex, redex occurrence and rewrite step is the most convenient one. The first definition of redex and redex occurrence as given in Definition 4.2.25 will be used in Chapter 5, and the second one, given in Definition 4.2.39 above, will be used in Chapter 6. Also in the case we choose to formalise redex occurrences and redexes as in Definition 4.2.39, we use the notation of Definition 4.2.27 for the set of redex occurrences in a term s, namely $\mathfrak{U}_{\mathcal{R}}(s)$ and the set of all pairs consisting of a position and a rewrite rule, namely $\mathfrak{U}_{\mathcal{R}}$.

Orthogonality, Almost Orthogonality and Weak Orthogonality.
In the remainder of this section we consider pattern higher-order rewriting systems with $\lambda^{\to}_{\overline{\eta}}$ as substitution calculus.

Orthogonality expresses that coinitial rewrite steps do not interfere with each other. There are different ways to formalise this principle. In this paragraph we will give a syntactic one, consisting of two conditions on the set of rewrite rules: rewrite rules are required to be left-linear and non-overlapping. This is the usual definition of orthogonality, given in [Klo92]. A different definition of orthogonality, more abstract in nature, is considered in [GLM92], [Oos94] and [Mel96]. In this more abstract approach, a rewriting system is said to be orthogonal if all complete developments of a set of redex occurrences are finite, end in the same term and induce the same residual relation. For pattern higher-order rewriting systems with $\lambda^{\to}_{\overline{\eta}}$ as substitution calculus, this is implied by the syntactic restrictions requiring rewrite rules to be left-linear and non-overlapping, see Theorem 3.2.26 in [Oos94, p.100].

Besides the definition of orthogonality, we will also present the definition of two relaxed versions, namely almost orthogonality and weak orthogonality. These notions will be used in Chapter 6 and Chapter 5.

Definition 4.2.41. Let $\mathcal{H} = (\lambda^{\to}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$ be a pattern higher-order rewriting system.

1. A rewrite rule $\mathbf{l} \to \mathbf{r}$ with $\mathbf{l} = x_1. \ldots .x_m.f s_1 \ldots s_n$ is said to be *left-linear* if every variable x_p such that $x_p \in \{x_1, \ldots, x_m\}$ occurs exactly once free in the subterm $f s_1 \ldots s_n$.

2. $\mathcal{H}$ is said to be *left-linear* if every rewrite rule in $\mathcal{R}$ is left-linear.

Example 4.2.42. All rewrite rules of the higher-order rewriting systems of Example 4.2.29 and Example 4.2.30 are left-linear. The rewrite rule

$$x.\mathsf{E}xx \to x.\mathsf{T}$$

which models a test for equality, is not left-linear.

For the definition of orthogonality, almost orthogonality and weak orthogonality we need the definition of non-overlapping redex occurrences. This definition is formulated using the second formalisation of the notion of redex occurrence, given in Definition 4.2.39. We make use of the definition of disjointness of positions, which is given in Definition 2.1.5 of Chapter 2.

Definition 4.2.43. Let $\mathcal{H} = (\lambda^{\to}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$ be a pattern higher-order rewriting system. Let s be a term and let $(\phi\, 0^m, \mathbf{l} \to \mathbf{r})$ and $(\phi'\, 0^n, \mathbf{l}' \to \mathbf{r}')$ be two redex occurrences in s, with m the arity of the head-symbol of $\mathbf{l}$ and n the arity of the head-symbol of $\mathbf{l}'$.

1. The redex occurrences $(\phi\, 0^m, \mathrm{l} \to \mathrm{r})$ and $(\phi' 0^n, \mathrm{l}' \to \mathrm{r})'$ are said to be *disjoint*, denoted by
$$(\phi\, 0^m, \mathrm{l} \to \mathrm{r}) \mid (\phi'\, 0^n, \mathrm{l}' \to \mathrm{r}')$$
if $\phi \mid \phi'$.

2. Let $\mathrm{l} = x_1.\dots.x_p.t$. The redex occurrence $(\phi\, 0^m, \mathrm{l} \to \mathrm{r})$ *nests* the redex occurrence $(\phi'\, 0^n, \mathrm{l}' \to \mathrm{r}')$, denoted by
$$(\phi\, 0^m, \mathrm{l} \to \mathrm{r}) \preceq (\phi'\, 0^n, \mathrm{l}' \to \mathrm{r}')$$
if $\phi' = \phi\chi\psi$ with $t|_{\chi 0^p} = x$ such that $x \in \{x_1, \dots, x_p\}$, and ψ an arbitrary position.

3. The redex occurrences $(\phi\, 0^m, \mathrm{l} \to \mathrm{r})$ and $(\phi' 0^n, \mathrm{l}' \to \mathrm{r}')$ are said to be *overlapping*, denoted by
$$(\phi\, 0^m, \mathrm{l} \to \mathrm{r}) \sharp (\phi'\, 0^n, \mathrm{l}' \to \mathrm{r}')$$
if

   (a) $(\phi\, 0^m, \mathrm{l} \to \mathrm{r})$ and $(\phi'\, 0^n, \mathrm{l}' \to \mathrm{r}')$ are different,

   (b) $\phi \preceq \phi'$ and the redex occurrence $(\phi\, 0^m, \mathrm{l} \to \mathrm{r})$ does not nest the redex occurrence $(\phi'\, 0^n, \mathrm{l}' \to \mathrm{r}')$, or $\phi' \preceq \phi$ and the redex occurrence $(\phi'\, 0^n, \mathrm{l}' \to \mathrm{r}')$ does not nest the redex occurrence $(\phi\, 0^m, \mathrm{l} \to \mathrm{r})$,

Notation 4.2.44. Let u and v be redex occurrences in a term s. We write $u \parallel v$ to denote that u and v are not overlapping. Note that it might be the case that $u = v$.

Example 4.2.45. We consider the higher-order rewriting system for parallel-or. The alphabet consists of the rewrite operators $\mathsf{por} : 0 \to 0 \to 0$, $\mathsf{T} : 0$ and $\mathsf{F} : 0$. The arity of the rewrite operator por is 2 and the arity of the rewrite operators T and F is 0. The rewrite rules for parallel or are the following:

$$\begin{array}{rcl} x.\mathsf{porT}x & \to & x.\mathsf{T} \\ x.\mathsf{por}x\mathsf{T} & \to & x.\mathsf{T} \\ \mathsf{porFF} & \to & \mathsf{F} \end{array}$$

Consider the term $\mathsf{por(porT(porT}x\mathsf{))(porTT)}$. This terms contains the redex occurrences

$$\begin{array}{l} (0100, x.\mathsf{porT}x \to x.\mathsf{T}), \\ (01100, x.\mathsf{porT}x \to x.\mathsf{T}), \\ (100, x.\mathsf{porT}x \to x.\mathsf{T}), \\ (100, x.\mathsf{por}x\mathsf{T} \to x.\mathsf{T}). \end{array}$$

We have

$$\begin{array}{l}(0100, x.\mathsf{porT}x \to x.\mathsf{T}) \mid (100, x.\mathsf{porT}x \to x.\mathsf{T}),\\ (0100, x.\mathsf{porT}x \to x.\mathsf{T}) \preceq (01100, x.\mathsf{porT}x \to x.\mathsf{T}),\\ (100, x.\mathsf{porT}x \to x.\mathsf{T}) \sharp (100, x.\mathsf{por}x\mathsf{T} \to x.\mathsf{T}).\end{array}$$

Definition 4.2.46. Let $\mathcal{H} = (\lambda^{\to}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$ be a pattern higher-order rewriting system. A set consisting of redex occurrences that are pairwise non-overlapping is said to be *simultaneous* or to consist of *simultaneous redex occurrences.*

We will define the classes of orthogonal, almost orthogonal and weakly orthogonal higher-order rewriting systems. To that end we need moreover the definition of ambiguous and weakly head-ambiguous redex occurrences.

Definition 4.2.47. Let $\mathcal{H} = (\lambda^{\to}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$ be a pattern higher-order rewriting system. Let s be a term and let $u = (\phi\, 0^m, \mathrm{l} \to \mathrm{r})$ and $u' = (\phi'\, 0^n, \mathrm{l}' \to \mathrm{r}')$ be two redex occurrences in s.

1. The redex occurrences u and u' are said to be *weakly ambiguous* if
   (a) u and u' are overlapping,
   (b) we have $u : s \to s'$ and $u' : s \to s'$.
2. The redex occurrences u and u' are said to be *weakly head-ambiguous* if they are weakly ambiguous and $\phi = \phi' = \epsilon$.

Example 4.2.48. The overlapping redex occurrences in Example 4.2.45 are weakly head-ambiguous.

In the rewriting system defined by the rewrite rules

$$\begin{array}{lll} fa & \to_{\mathrm{R}_1} & fb \\ a & \to_{\mathrm{R}_2} & b \end{array}$$

the redex occurrences $(0, \mathrm{R}_1)$ and $(1, \mathrm{R}_2)$ in the term fa are weakly ambiguous but not weakly head-ambiguous.

Definition 4.2.49. Let $\mathcal{H} = (\lambda^{\to}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$ be a pattern higher-order rewriting system.

1. $\mathcal{H}$ is said to be *orthogonal* $\mathcal{H}$ is left-linear and there there is no term with overlapping redex occurrences.
2. $\mathcal{H}$ is said to be *almost orthogonal* if $\mathcal{H}$ is left-linear and every two overlapping redex occurrences are weakly head-ambiguous.

3. $\mathcal{H}$ is said to be *weakly orthogonal* if $\mathcal{H}$ is left-linear and every two overlapping redex occurrences are weakly ambiguous.

Note that the notion of orthogonality concerns the left-hand sides of rewrite rules only. The notions almost orthogonality and weak orthogonality however concern both left- and right-hand side of the rewrite rule. Therefore an extension from orthogonal to weakly orthogonal rewriting systems usually causes serious complications.

In a pattern higher-order rewriting system all developments of sets of simultaneous redex occurrences are finite. In Chapter 6, we will use this result, which is proved by van Oostrom as Theorem 3.1.45 in [Oos94].

Theorem 4.2.50. *In a pattern higher-order rewriting system that is left-linear, all developments of sets of simultaneous redex occurrences are finite.*

Earlier proofs of this result, in more restricted settings, in have been given by Klop in [Klo80] for orthogonal CRSs and by Khasidashvili in [Kha92] for orthogonal ERSs.

We conclude this section by giving the definition of a fully extended higher-order rewriting system. This concept will be needed in Chapter 6.

Definition 4.2.51.

1. Let $\mathtt{l} \rightarrow \mathtt{r} = x_1.\ldots.x_m.s \rightarrow x_1.\ldots.x_m.t$ be a rewrite rule. It is said to be *fully extended* if every $x \in \{x_1, \ldots, x_m\}$ occurs in s at position $\phi\, 0^p$, with p the arity of x, in a subterm of the form $x s_1 \ldots s_p$, with $s_1, \ldots, s_p$ the $\overline{\eta}$-normal forms of all variables that are bound in s at position ϕ.

2. A pattern higher-order rewriting system is said to be *fully extended* if every rewrite rule is fully extended.

An example of a rewrite rule that is not fully extended is the rule for η-reduction. In the format of higher-order rewriting systems, it is written as $z.\mathsf{abs}(x.\mathsf{app}zx) \rightarrow z.z$. In this representation, the side-condition in the traditional way of writing the rule, namely as $\lambda x.Mx \rightarrow M$ if $x \notin \mathsf{FV}(M)$, is internalised. This illustrates that a rewrite rule that is not fully extended contains implicitly a 'no occur' check. The rewrite relation of fully extended rewriting systems is not restricted by such conditions.

Sharing. We conclude this section with a short digression concerning sharing in higher-order rewriting systems. It reflects discussions with Zurab Khasidashvili and Vincent van Oostrom. Usually, the rewrite sequence $\sigma : s \twoheadrightarrow t$ is considered to be more efficient than a rewrite sequence $\sigma' : s \twoheadrightarrow t$ if σ consists of less rewrite steps than σ'. Taking this point of view, a way to turn a rewrite sequence into a more efficient one is by sharing equal subterms. This can be done by representing terms as graphs.

In [Wad71], Wadsworth presents the first graph implementation for (untyped) λ-calculus. Lévy formulated in [Lév78] the notion of optimal rewrite sequence for λ-calculus. This is a criterion for efficiency. A rewrite sequence is said to be optimal if redexes that are residuals of the same redex, or that are created in the same way, are shared. The graph implementation by Wadsworth is not optimal in this sense. The first optimal implementations of untyped λ-calculus are due to Lamping [Lam90] and Kathail. More recently, a lot of attention was devoted to the subject of optimal rewriting, see for instance [Asp94] , [GAL92], [Lan93], [AL94].

We give an example that reveals that sharing of subterms and also of contexts can be expressed in higher-order rewriting systems in a natural way, if the rewrite relation is defined on the set of preterms instead of on the set of terms only. In the examples, a λ-term that is simply typable is rewritten in an optimal way. We claim this can be done for every λ-term that is simply typable.

We consider the λ-term $(\lambda x.(xN_0)(xN_1))(\lambda y.(\lambda z.z)y)$. The representation of this term in the graph implementation by Wadsworth cannot be rewritten in an optimal way. We consider λ-calculus as a higher-order rewriting system, and perform first an expansion of the term to a preterm that is in the same $\leftrightarrow^*_{SC}$-equivalence class. This preterm can be rewritten in an optimal way:

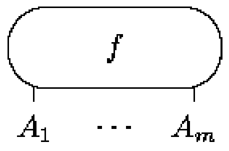
$$\begin{array}{rl}
\mathsf{app}\ (\mathsf{abs}\, \mathsf{x}.\mathsf{app}(\mathsf{app}\ \mathsf{x}\, \mathsf{N}_0)(\mathsf{app}\ \mathsf{x}\, \mathsf{N}_1))(\mathsf{abs}\, \mathsf{y}.\mathsf{app}\ (\mathsf{abs}\, \mathsf{z}.\mathsf{z})\mathsf{y}) & {}_{SC}\twoheadleftarrow \\
\mathsf{app}\ (\mathsf{abs}\ \mathsf{x}.(\mathsf{z}.\mathsf{app}(\mathsf{z}\mathsf{N}_0)(\mathsf{z}\mathsf{N}_1))(\mathsf{z}'.\mathsf{app}\ \mathsf{x}\mathsf{z}'))(\mathsf{abs}\, \mathsf{y}.\mathsf{app}\ (\mathsf{abs}\, \mathsf{z}.\mathsf{z})\mathsf{y}) & \rightarrow_{beta} \\
(z.\mathsf{app}\ (\mathsf{z}\mathsf{N}_0)(\mathsf{z}\mathsf{N}_1))(\mathsf{z}'.\mathsf{app}\ (\mathsf{abs}\, \mathsf{y}.\mathsf{app}(\mathsf{abs}\, \mathsf{z}.\mathsf{z})\mathsf{y})\mathsf{z}') & \rightarrow_{beta} \\
(z.\mathsf{app}\ (\mathsf{z}\mathsf{N}_0)(\mathsf{z}\mathsf{N}_1))(\mathsf{z}'.\mathsf{app}\ (\mathsf{abs}\, \mathsf{z}.\mathsf{z})\mathsf{z}') & \rightarrow_{beta} \\
(z.\mathsf{app}\ (\mathsf{z}\mathsf{N}_0)(\mathsf{z}\mathsf{N}_1))(\mathsf{z}'.\mathsf{z}') & \twoheadrightarrow_{SC} \\
\mathsf{app}\ \mathsf{N}_0\mathsf{N}_1. &
\end{array}$$

4.3 Proof Nets as Substitution Calculus

In the previous section we have studied higher-order rewriting systems with $\lambda^{\rightarrow}_{\overline{\eta}}$ as substitution calculus. We consider $\lambda^{\rightarrow}_{\overline{\eta}}$ as the prime example of a substitution calculus. However, it is clearly not the only possible choice. In this section we consider higher-order rewriting systems with a less traditional substitution calculus, namely the rewriting system $(\mathsf{PN}, \rhd)$ consisting of proof nets with cut-elimination. It is not completely straightforward to see that $(\mathsf{PN}, \rhd)$ can serve as a substitution calculus, however, since it forms a refinement of simply typed λ-calculus, it seems worthwhile to investigate the possibility.

In this section, we will show that proof nets, that also may contain proof net constants, with cut-elimination satisfy the requirements imposed on a substitution calculus in Definition 4.1.1. We will make use of the definitions and results presented in Chapter 3. Then we define higher-order rewriting systems with proof nets as substitution calculus. As examples, we represent a first-order term rewriting system and linear λ-calculus as higher-order systems with $(\mathsf{PN}, \rhd)$ as substitution calculus.

Proof Nets with Constants. In this section, we will consider proof nets that differ from the ones that are defined in Definition 3.2.3 of Chapter 3 in that they may contain *proof net constants*. The alphabet of proof nets, given in Definition 3.2.2 of Chapter 3, is extended with possibly infinitely many proof net constants. A proof net constant has a finite number of outputs. We write an A_1-…-A_m-constant as



The set of proof nets with proof net constants is defined as the set of proof nets as in Definition 3.1.1 of Chapter 3, with in addition the following clause: an A_1-…-A_m-constant is a proof net with output occurrences $A_1, \ldots, A_m$, written as above.

Material. The *types* of $(\mathsf{PN}, \rhd)$ are the formulae of multiplicative-exponential linear logic, given in Definition 3.1.1 of Chapter 3. We will call an output A of a proof net also an *output of type* A. The prestructures of type A, in this section mostly called *prenets* of type A, are the proof nets that have an output of type A. So a prenet can have different types, and hence it is not necessarily the case that the set of prenets of type A and the set of prenets of type B with $A \neq B$ are disjoint. We write $M : A$ if the prenet M is of type A. The *variables of type* A are all A-axiom links. So a variable of type A is also a variable of type $A^{\perp}$. It is clear that for every type A there are infinitely many variables of type A. Further, a variable of type A is a prenet of type A.

For the definition of a substitution operator we need the notion of a free variable occurrence in a prenet. A free variable occurrence in a prenet will be defined as an output occurrence that has only an axiom link or a weakening link, and dereliction, contraction and box links as premisses. To formalise this concept we need the following definition.

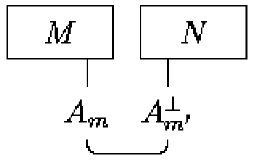
Definition 4.3.1. Let M be a prenet. The *variable depth* of a formula occurrence A_m in M, denoted by $\mathsf{vd}(A_m, M)$, is defined as follows:

1. if the formula occurrence A_m is the conclusion of an A-axiom link, or if $A = ?A_0$ and the formula occurrence A_m is the conclusion of an A_0-weakening link, then $\mathsf{vd}(A_m, M) = 0$,
2. if $A = A_0 \otimes A_1$ and the formula occurrence A_m is the conclusion of an A_0-A_1-times link with as premisses the formula occurrences $(A_0)_{m_0}$ and $(A_1)_{m_1}$, then $\mathsf{vd}(A_m, M) = \max\{\mathsf{vd}((A_0)_{m_0}, M), \mathsf{vd}((A_1)_{m_1}, M)\} + 1$,
3. if $A = A_0 \wp A_1$ and the formula occurrence A_m is the conclusion of an A_0-A_1-par link with as premisses the formula occurrences $(A_0)_{m0}$ and $(A_1)_{m1}$, then $\mathsf{vd}(A_m, M) = \max\{\mathsf{vd}((A_0)_{m0}, M), \mathsf{vd}((A_1)_{m1}, M)\} + 1$,
4. if $A = ?A_0$ and the formula occurrence A_m is the conclusion of an A_0-contraction link with as premisses the formula occurrences A_{m_0} and A_{m_1}, then $\mathsf{vd}(A_m, M) = \max\{\mathsf{vd}(A_{m_0}, M), \mathsf{vd}(A_{m_1}, M)\}$,
5. if $A = ?A_0$ and the formula occurrence A_m is the conclusion of an A_0-dereliction link with as premiss the formula occurrence $(A_0)_{m_0}$, then $\mathsf{vd}(A_m, M) = \mathsf{vd}((A_0)_{m_0}, M)$,
6. if $A = ?A_1$ and the formula occurrence A_m is the conclusion of a B-A_1-A_2-$\ldots$-A_m-box link, having as premisses the formula occurrences $B_n, (?A_1)_{m_1}, (?A_2)_{m_2}, \ldots, (?A_m)_{m_m}$, then $\mathsf{vd}(A_m, M) = \mathsf{vd}((?A_1)_{m_1}, M)$,
7. if $A = !A_0$ and the formula occurrence A_m is the conclusion of an A_0-B_1-$\ldots$-B_m-box link, having as premisses the formula occurrences $(A_0)_{m_0}, (?B_1)_{n_1}, \ldots, (?B_m)_{n_m}$, then $\mathsf{vd}(A_m, M) = \mathsf{vd}((A_0)_{m_0}, M) + 1$.

Definition 4.3.2. A *free variable occurrence of type A* in a prenet M is an output occurrence A_m in M such that $\mathsf{vd}(A_m, M) = 0$.

Now we can define the substitution of a prenet N of type A for a variable occurrence of type A in a prenet M.

Definition 4.3.3. Let M be a proof net with a free variable occurrence A_m of type A. Let N be a proof net of type $A^\perp$ with an output occurrence $A^\perp_{m'}$. The substitution of N in M for A_m, denoted by $M[m := N]$, using the label indicating the occurrence of the formula A in M, is the following prenet:



So the operation for substitution is concatenation by a cut link. Inspired by the various translations of λ-terms into proof nets, we give the following definition of a bound variable occurrence in a proof net.

Definition 4.3.4. An output occurrence $(A \parr B)_{m'}$ of M is said to be a *bound variable occurrence of type* A if A_m is a variable occurrence in a prenet M_0, and M is obtained from M_0 by adding an A-B-par link between the output occurrences A_m and B_n.

The definition of a closed prenet is as the one for a λ-term, making use of the notions of free and bound variable occurrences as introduced above.

Definition 4.3.5. A prenet is said to be *closed* if it does not contain free variable occurrences.

There certainly are alternative ways to define the notions of free and bound variable occurrences in a proof net. Here we restrict attention to just the one given above.

In the remainder of this section we will omit the labels of formulae in a proof net whenever we consider that to be convenient.

(PN,$\triangleright$) is a Substitution Calculus. We now sketch why the calculus (PN,$\triangleright$) satisfies the conditions imposed on a substitution calculus in Definition 4.1.1.

First, rewriting of proof nets consists of cut-elimination as defined in Definition 3.2.5. The set of prenets is closed under cut-elimination. We also have the stronger property that the set of prenets of type A is closed under cut-elimination, since the reduct of a proof net M is a proof net with outputs of the same type as M. In Chapter 3, we have shown that (PN,$\triangleright$) is strongly normalising. Since it is routine to verify that (PN,$\triangleright$) is locally confluent, we find that (PN,$\triangleright$) is confluent. So the rewrite relation $\triangleright$ on PreNets has all properties that the rewrite relation of a substitution calculus should have.

Second, the typing relation is properly defined on PreNets $\times$ Form. It is clear that every prenet is typed, and that every type is inhabited.

Third, we need to verify that the rewrite relation $\triangleright$ satisfies the properties of subject conversion and type conversion with respect to the typing relation. The latter holds, since types are not rewritten. The property of subject reduction is easily seen to hold, since the set of prenets of a certain type is closed under rewriting. In fact, since output occurrences are never erased, we also find that the rewrite relation $\rhd$ satisfies the property of subject conversion with respect to the typing relation.

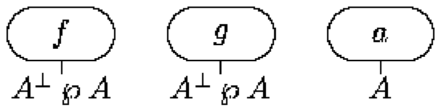
Finally, it is easy to see that the rewrite relation and the typing relation behave well with respect to each other.

We conclude that prenets with cut-elimination form a decent substitution calculus.

Higher-Order Rewriting Systems with $(\mathsf{PN}, \rhd)$ as Substitution Calculus.

Definition 4.3.6. A *rewrite alphabet for* $(\mathsf{PN}, \rhd)$ is a set, denoted by $\mathcal{A}$, consisting of proof net constants. Proof net constants are also called *rewrite operators*.

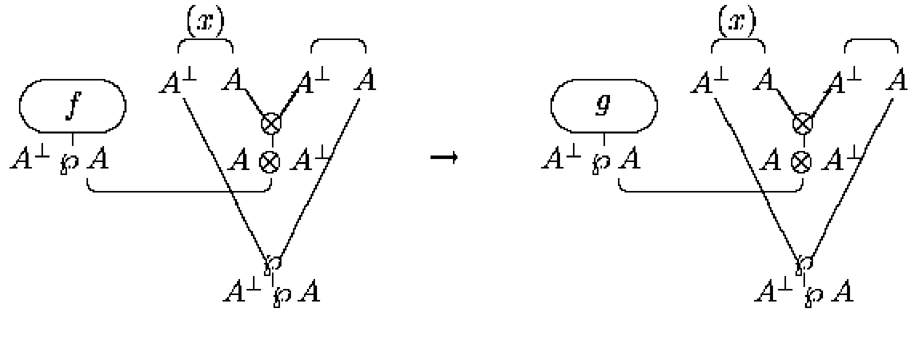
Example 4.3.7. An example of a rewrite alphabet for $(\mathsf{PN}, \rhd)$ is the set $\mathcal{A}$ consisting of the following rewrite operators:



Definition 4.3.8. Let $\mathcal{A}$ be a rewrite alphabet for $(\mathsf{PN}, \rhd)$. A *rewrite rule of type A over $\mathcal{A}$* is a pair of nets $(\mathtt{l}, \mathtt{r})$ such that

1. $\mathtt{l}$ and $\mathtt{r}$ are of type A,
2. $\mathtt{l}$ and $\mathtt{r}$ have the same output types,
3. all constants in $\mathtt{l}$ and $\mathtt{r}$ are in $\mathcal{A}$,

Example 4.3.9. An example of a rewrite rule over the alphabet given in Example 4.3.7 is the following:



It represents the term rewriting rule $f(x) \to g(x)$. We have indicated by (x) the axiom link that corresponds to the variable x of the rule in the format of term rewriting.

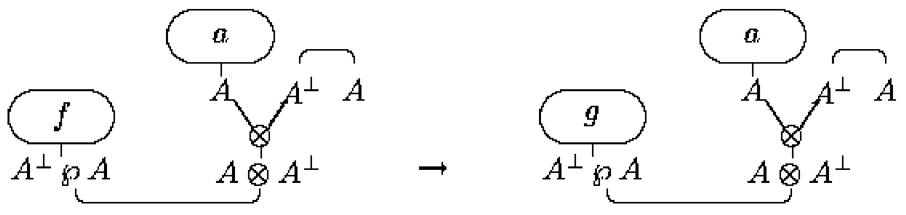
Definition 4.3.10. A *higher-order rewriting system with* $(\mathsf{PN}, \rhd)$ *as substitution calculus* is a triple $((\mathsf{PN}, \rhd), \mathcal{A}, \mathcal{R})$ with $\mathcal{A}$ a rewrite alphabet for $(\mathsf{PN}, \rhd)$ and $\mathcal{R}$ a set of rewrite rules over $\mathcal{A}$.

The rewrite rules induce a rewrite relation on the set of nets. It is defined as follows.

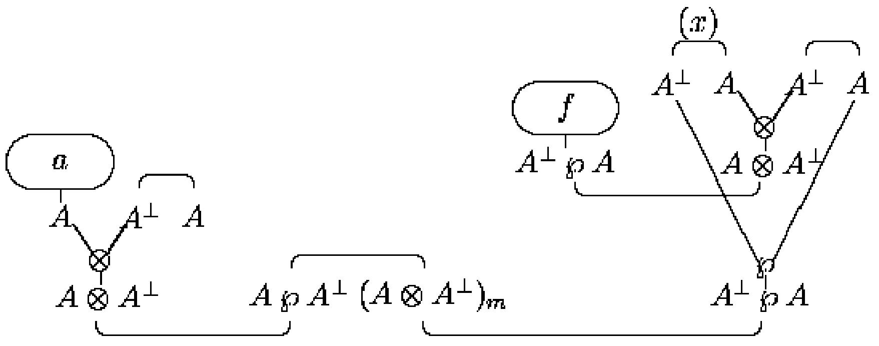
Definition 4.3.11. Let $((\mathsf{PN}, \rhd), \mathcal{A}, \mathcal{R})$ be a higher-order rewriting system. The *rewrite relation* on the set of nets, denoted by $\rightarrow_{\mathcal{R}}$, is defined as follows. We have $M \rightarrow_{\mathcal{R}} M'$ if there exist a rewrite rule $\mathrm{l} \rightarrow \mathrm{r}$ of type A, and a net N with a variable occurrence A_m of type A such that

1. $N[m := \mathrm{l}] \rhd^* M$,
2. $N[m := \mathrm{r}] \rhd^* M'$.

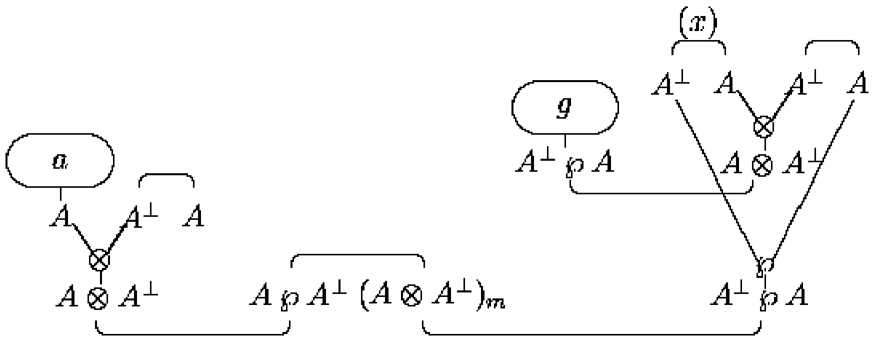
Example 4.3.12. We give an example of a rewrite step in the higher-order rewriting system $((\mathsf{PN}, \rhd), \mathcal{A}, \mathcal{R})$ with $\mathcal{A}$ the rewrite alphabet of Example 4.3.7 and $\mathcal{R}$ the set consisting of the rewrite rule of Example 4.3.9.



We abbreviate this rewrite step as $M \rhd M'$. This rewrite step is obtained because the prenet



is of the form $N[m := \mathrm{l}]$, with N the left two parts of the picture above, and it is rewritten to M, and the prenet



is of the form $N[m := 1]$ and it is rewritten to M'.

As a second example of a higher-order rewriting system with $(\mathsf{PN}, \rhd)$ as substitution calculus we consider linear λ-calculus. It is also possible to represent λ-calculus without the linearity constraint as a higher-order rewriting system with $(\mathsf{PN}, \rhd)$ as substitution calculus.

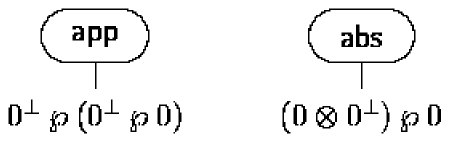
Example 4.3.13. In the format of higher-order rewriting systems with λ-calculus as substitution calculus, the rewrite rule for β-reduction is written as

$$z.z'.\mathsf{app}(\mathsf{abs}(x.zx))z \to z.z'.zz'.$$

We now write this rule in the format of higher-order rewriting systems with $(\mathsf{PN}, \rhd)$ as substitution calculus. We simplify the example in two ways. First, by restricting attention to linear λ-calculus, in which every λ binds exactly one variable. This means that we won't need boxes in the representation. Second, we do not write the rule as a pair of closed terms, but we represent it as

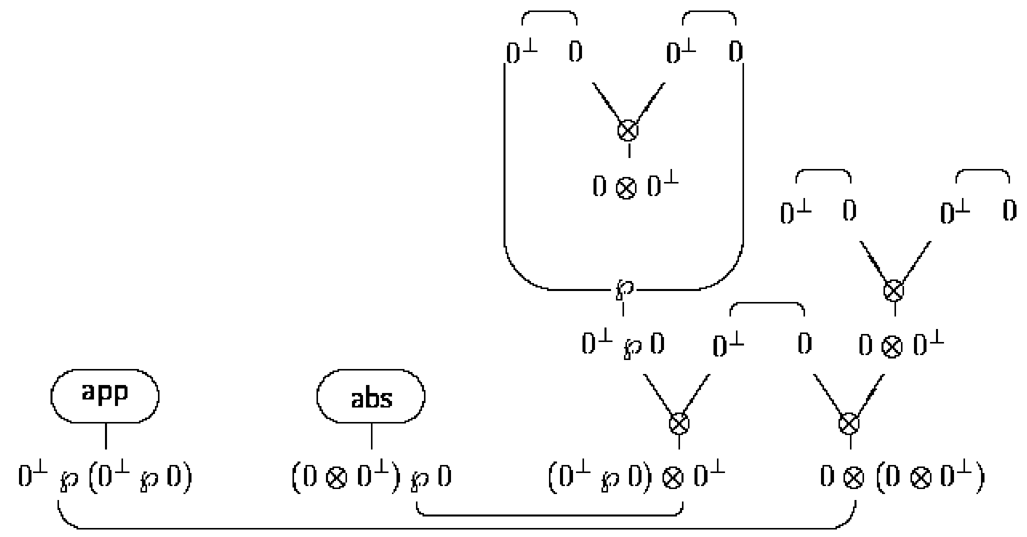
$$\mathsf{app}(\mathsf{abs}(x.zx))z \to zz'.$$

This also simplifies the picture since we need less par-links. The rewrite alphabet contains the following two rewrite operators:

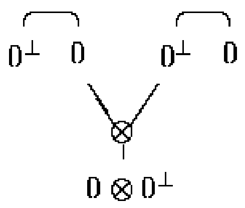


app
$0^\perp \parr (0^\perp \parr 0)$
abs
$(0 \otimes 0^\perp) \parr 0$

The left-hand side is as follows:


app
abs
$0^{\perp} \wp (0^{\perp} \wp 0)$
$(0 \otimes 0^{\perp}) \wp 0$
$(0^{\perp} \wp 0) \otimes 0^{\perp}$
$0 \otimes (0 \otimes 0^{\perp})$

The right-hand side is



Higher-order rewriting systems with proof nets as substitution calculus certainly need further investigation. Quite some choices need to be made, and the alternatives need to be compared in detail. Moreover, a lot of notions that are usual in term rewriting do not immediately have their counterpart in a two-dimensional syntax. The aim of this section is to make clear that it is worthwhile to pursue a study of this subject.

4.4 Higher-Order Rewrite Systems

Higher-Order Rewrite Systems, in the sequel abbreviated as HRSs, are introduced by Nipkow in [Nip91]. In HRSs, simply typed λ-terms are rewritten modulo $\beta\overline{\eta}$-reduction. HRSs form a proper extension of both first-order term rewriting and λ-calculus, introduced to investigate the meta-theory of systems like λProlog and Isabelle. Wolfram defines in [Wol91, Wol93] Higher-Order Term Rewriting Systems, which are similar to HRSs. We will comment on this below.

Quite a lot of rewriting theory for HRSs has been developed already. Results concerning confluence have been obtained for HRSs in which the left-hand side of a rewrite rule is required to be a pattern, as in Definition 4.2.32. In [Nip91], Nipkow shows that a HRS is locally confluent if every critical pair is confluent. In [Nip93b] it is shown that orthogonal HRSs are confluent, using parallel reduction. We will discuss this result in more detail in Chapter 5. Both confluence results are recapitulated in detail by Mayr and Nipkow in [MN94]. In [Oos96], van Oostrom gives a condition on critical pairs of a HRS that implies confluence. This generalises a result proved by Huet in [Hue80] for first-order term rewriting, stating that a term rewriting system is confluent if its critical pairs are parallel closed (where the meaning of parallel is different from the one considered in Chapter 5).

The unification algorithm for patterns presented by Miller in [Mil91] is simplified by Nipkow in [Nip93a]. Prehofer defines in [Pre95] several classes of simply typed λ-terms satisfying relaxed versions of the restriction to patterns, for which second-order unification is decidable.

Van de Pol presents a technique for proving termination of a HRS in [Pol94]. He shows that a HRS is strongly normalising if it can be interpreted in a suitable way in a well-founded higher-order algebra. For this result to hold, it is not necessary that left-hand sides of rewrite rules are patterns. Further results on termination of HRSs are obtained by van de Pol and Schwichtenberg [PS95] and Kahrs [Kah95]. A technique for proving termination of a first-order rewriting system, namely by means of the so-called recursive path ordering, has been lifted to case of HRSs by Lysne and Piris in [LP95].

In the remainder of this section, we first recall the syntax of HRSs, and then we show how to represent a HRS as a higher-order rewriting system with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus. As we will see, HRSs and higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus are very similar. The only two differences are the following: first, in a HRS the left- and right-hand side of a rewrite rule must be of base type, whereas in higher-order rewriting systems such a restriction is not made, and second, the definition of the rewrite relation of a HRS differs from the one of a higher-order rewriting system.

Higher-Order Rewrite Systems. In the presentation of the syntax of HRSs we fix attention to the differences with higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus.

Terms of a HRS are built from simply typed variables and simply typed constants as in simply typed λ-calculus. A *rewrite alphabet* for a HRS is a set of simply typed constants. Terms are supposed to be in $\beta\overline{\eta}$-normal form. There is a small difference in notation: abstraction is in a HRS denoted by $\lambda x.s$, and as $x.s$ in a higher-order rewriting system. The definition of a rewrite rule is slightly different from the one given for a higher-order rewriting system. It makes use of the definition of a pattern that is given in Definition 4.2.32.

Definition 4.4.1. Let $\mathcal{A}$ be a rewrite alphabet. A *rewrite rule over* $\mathcal{A}$ is a pair of terms over $\mathcal{A}$, denoted by $\mathtt{l} \rightarrow \mathtt{r}$, such that

1. $\mathtt{l}$ and $\mathtt{r}$ are of the same base type,
2. all free variables in $\mathtt{r}$ occur also in $\mathtt{l}$,
3. $\mathtt{l}$ is of the form $fs_1 \dots s_m$ with $s_1, \dots, s_m$ patterns.

The left- and right-hand side of a rewrite rule in a higher-order rewriting system with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus are not required to be of base type. The same holds for the Higher-Order Term Rewriting Systems as defined by Wolfram in [Wol91, Wol93]. Moreover, the left-hand side of a rewrite rule as defined in [Wol91, Wol93] is not necessarily a pattern.

Definition 4.4.2. A *HRS* is a pair $(\mathcal{A}, \mathcal{R})$ consisting of a rewrite alphabet $\mathcal{A}$ and a set of rewrite rules over $\mathcal{A}$.

Example 4.4.3. The representation of untyped λ-calculus as a HRS is almost as its representation as a higher-order rewriting system, given in Example 4.2.19. There is one base type, denoted by 0, and there are two constants, namely $\mathsf{abs} : (0 \to 0) \to 0$ and $\mathsf{app} : 0 \to 0 \to 0$. The rule for β-reduction is written as follows:

$$\mathsf{app}(\mathsf{abs}(\lambda x.zx))z' \to zz'.$$

The difference with the representation of λ-calculus as a higher-order rewriting system is that in that case rewrite rules are required to consist of closed terms.

The second aspect in which HRSs differ from higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus is the rewrite relation. The rewrite relation of a HRS is defined using the concepts of context and assignment.

We suppose that for every base type 0 there is a distinguished variable of type 0, denoted by $\Box$, that never occurs bound. A *context* is a term with one occurrence of $\Box$. The difference with a context of a higher-order rewriting system, defined in Definition 4.2.22 of Section 4.2, is that in the present case $\Box$ is required to be of some base type. An *assignment* is a mapping $\theta : \mathsf{Var} \to \mathsf{Terms}$ that respects the typing. It is extended to a homomorphism $\theta : \mathsf{Terms} \to \mathsf{Terms}$ as follows:

1. $x^\theta = \theta(x)$,
2. $f^\theta = f$,
3. $(\lambda x.s)^\theta = \lambda x.s^\theta$,
4. $(s_0 s_1)^\theta = s_0^\theta s_1^\theta$.

Now we can give the definition of the rewrite relation.

Definition 4.4.4. Let $(\mathcal{A}, \mathcal{R})$ be a HRS. The *rewrite relation*, denoted by $\rightarrow_{\mathcal{R}}$, is defined as follows:

$$s \rightarrow_{\mathcal{R}} s'$$

if there is an assignment θ, a context $C\Box$ and a rewrite rule $\mathrm{l} \rightarrow \mathrm{r}$ such that

$$s = C[\mathrm{l}^{\theta}\downarrow_{\beta\overline{\eta}}] \quad \text{and} \quad s' = C[\mathrm{r}^{\theta}\downarrow_{\beta\overline{\eta}}].$$

Note that s and s' in Definition 4.4.4 are in $\beta\overline{\eta}$-normal form since l and r are of base type. Since the context $C\Box$ is in normal form and $\mathrm{l}^{\theta}\downarrow_{\beta\overline{\eta}}$ and $\mathrm{r}^{\theta}\downarrow_{\beta\overline{\eta}}$ are not of the form $\lambda x.t$, we have that $C[\mathrm{l}^{\theta}\downarrow_{\beta\overline{\eta}}]$ and $C[\mathrm{r}^{\theta}\downarrow_{\beta\overline{\eta}}]$ are in $\beta\overline{\eta}$-normal form.

Rewrite rules often originate from a set of equations by giving every equation a direction. Such a rewriting system obtained from a set of equations E should implement the equational theory of E in a faithful way. That is, $s \leftrightarrow^{*}_{\mathcal{R}} s'$ if and only if $s =_E s'$. Here $=_E$ denotes the equivalence relation generated by E, see for instance [MN94] for the definition. As observed by Nipkow in [Nip91], this important requirement is not necessarily met by HRSs with a rewrite relation as defined in 4.4.4, if the rewrite rules are not necessarily of base type. We consider the example given in [Nip91]. The equation $\lambda x.fx(zx) = \lambda x.f'(zx)x$ gives rise to the rewrite rule $\lambda x.fx(zx) \rightarrow \lambda x.f'(zx)x$. We have $fa(za) =_E f'(za)a$, but there is no rewrite sequence between $fa(za)$ and $f'(za)a$ or vice versa, since both $fa(za)$ and $f'(za)a$ are in normal form for the rewrite rule above.

We remark that if the rewrite relation is defined as in [Wol93] or as in Definition 4.2.25 of Section 4.2, rules of compound type do not pose this problem. The rewrite rule $\lambda x.fx(zx) \rightarrow \lambda x.f'(zx)x$ then does define a rewrite step $fa(za) \rightarrow f'(za)a$.

A Translation. It is clear that hardly anything needs to be done to represent a HRS as a higher-order rewriting system with $\lambda^{\rightarrow}_{\overline{\eta}}$ as substitution calculus. In fact, the main thing is to transform a rewrite rule $\mathrm{l} \rightarrow \mathrm{r}$ of a HRS into a rewrite rule $x_1.\ldots.x_m.\mathrm{l} \rightarrow x_1.\ldots.x_m.\mathrm{r}$, with $x_1, \ldots, x_m$ the variables that occur free in l. Following [Oos94], we call the latter rewrite rule the closure of the former.

Definition 4.4.5. Let $\mathrm{l} \rightarrow \mathrm{r}$ be a pair of terms in a higher-order rewriting system $\mathcal{H} = (\lambda^{\rightarrow}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$. The *closure* of $\mathrm{l} \rightarrow \mathrm{r}$ is the pair $x_1.\ldots.x_m.\mathrm{l} \rightarrow x_1.\ldots.x_m.\mathrm{r}$, with $\{x_1, \ldots, x_m\} = \mathsf{FV}(\mathrm{l})$.

Example 4.4.6. The closure of the rewrite rule for β-reduction as in Example 4.4.3 is

$$z.z'.\mathsf{app}(\mathsf{abs}(\lambda x.zx))z' \rightarrow z.z'.zz'.$$

This is exactly the way the rule for β-reduction is represented in Example 4.2.19.

If every rewrite rule of a HRS is transformed into its closure, we obtain a higher-order rewriting system with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus. It is easy to see that whenever $s \rightarrow_{\mathcal{R}} s'$ in a HRS, we have $s \rightarrow_{\mathcal{R}} s'$ in its associated higher-order rewriting system. As remarked above, it is not possible to translate an arbitrary higher-order rewriting system into a HRS such that rewrite steps are preserved.

4.5 Combinatory Reduction Systems

Combinatory Reduction Systems, abbreviated as CRSs, are introduced by Klop in [Klo80]. CRSs form a generalisation of contraction schemes, that were introduced by Aczel in [Acz78]. In both cases the format of first-order term rewriting systems is extended with a binding mechanism for variables.

The study of higher-order rewriting started with the introduction of CRSs by Klop in [Klo80]. For CRSs quite some rewriting theory is developed. We mention some results. In [Klo80], Klop proves that orthogonal CRSs are confluent. This result is obtained in the following way. First, it is shown that every set of redex occurrences admits a finite development. This is used to prove confluence. Then, using the confluence result, it is shown that all developments are finite. Also in [Klo80] it is shown that λ-calculus with surjective pairing is not confluent. Another result in [Klo80] concerns normalisation of orthogonal CRSs. A method is given that permits to infer strong normalisation of a CRS from weak normalisation. This is done by mimicking its rewrite sequences in another CRS, in which subterms are never erased. This generalises a method due to Nederpelt, which is reported in [Ned73] (see also [NGV94]). In [Raa93], orthogonal CRSs are shown to be confluent using parallel reductions. CRSs with explicit substitution are studied in [Ros96]. Further, work on implementation of CRSs is done by Kahrs [Kah94] and by Rose [Ros96].

We recall the syntax of CRSs, following the presentation in [KOR93]. We will see that CRSs differ more from higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus than HRSs do. Then we explain how to represent a CRS as a higher-order rewriting system, using the translation of CRSs into HRSs that is defined in [OR94a].

Combinatory Reduction Systems. We suppose that there is a set denoted by Var consisting of infinitely many *variables*. Variables are denoted by $x, y, z, \ldots$. We suppose moreover that there is a set denoted by MetaVar that contains for every m with $m \geq 0$ infinitely many *metavariables* of arity m. The *arity* of a metavariable expresses how many arguments it is supposed to have. Finally, there is in every CRS a binary operator for *abstraction*, denoted by $[\_]\_$. A *rewrite alphabet* for a CRS is a set, denoted by Σ, that consists of function symbols, denoted by f, g, h. Every function symbol is equipped with a fixed *arity* that, like the arity of a metavariable, expresses how many arguments the function symbol is supposed to have.

In CRSs a distinction is made between terms and metaterms, as in contraction schemes. In this respect CRSs differ from higher-order rewriting systems and from HRSs. The left- and right-hand side of a rewrite rule of a CRS will be metaterms. The rewrite rules induce a rewrite relation on the set of terms by assigning terms to metavariables, as will be explained below. First we define metaterms and terms.

Definition 4.5.1. Let Σ be a rewrite alphabet for a CRS.

1. The set of *metaterms over* Σ, denoted by $\mathsf{MetaTerms}$, is defined as the smallest set that satisfies the following requirements:
   (a) $x \in \mathsf{MetaVar}$ for every variable x,
   (b) if $s \in \mathsf{MetaTerms}$ and $x \in \mathsf{Var}$, then $[x]s \in \mathsf{MetaTerms}$,
   (c) if $s_1, \ldots, s_m \in \mathsf{MetaTerms}$ and $f \in \Sigma$ is a function symbol of arity m, then $f(s_1, \ldots, s_m) \in \mathsf{MetaTerms}$,
   (d) if $s_1, \ldots, s_m \in \mathsf{MetaTerms}$ and Z is a metavariable of arity m, then $Z(s_1, \ldots, s_m) \in \mathsf{MetaTerms}$.
2. A *term* is a metaterm without occurrences of metavariables. The set of terms is denoted by Terms.

Terms and metaterms are denoted by $s, t, r, \ldots$. We write f and Z instead of $f()$ and $Z()$ for function symbols and metavariables of arity 0.

A variable in a metaterm of a CRS is bound if it is in the scope of an abstraction $[x]$, and is free otherwise. Formally, we have the following definition.

Definition 4.5.2. The *set of free variables of a metaterm* s, denoted by $\mathsf{FV}(s)$, is defined inductively as follows:

1. $\mathsf{FV}(x) = \{x\}$,
2. $\mathsf{FV}([x]s) = \mathsf{FV}(s) \setminus \{x\}$,
3. $\mathsf{FV}(f(s_1, \ldots, s_m)) = \cup_{p=1}^{p=m} \mathsf{FV}(s_p)$,
4. $\mathsf{FV}(Z(s_1, \ldots, s_m)) = \cup_{p=1}^{p=m} \mathsf{FV}(s_p)$.

A variable occurring in s that is not free is said to be *bound* in s. A metaterm not containing free variables is said to be *closed*.

Definition 4.5.3. Let Σ be a rewrite alphabet. A *rewrite rule over* Σ is a pair of metaterms over Σ, denoted by $\mathtt{l} \to \mathtt{r}$, such that

1. $\mathtt{l}$ and $\mathtt{r}$ are closed,
2. $\mathtt{l}$ is of the form $f(s_1, \dots, s_m)$,
3. all metavariables in $\mathtt{l}$ occur in a submetaterm of the form $Z(x_1, \dots, x_m)$, with $x_1, \dots, x_m$ different variables that are bound in $\mathtt{l}$,
4. all metavariables in $\mathtt{r}$ occur in $\mathtt{l}$.

At this point we can clarify the difference between CRSs and contraction schemes. In a *contraction scheme* as defined by Aczel in [Acz78], the left-hand side of a rule is required to be of the form $f([\vec{x}_1]\mathtt{l}_1, \dots, [\vec{x}_m]\mathtt{l}_m)$, such that for every $n \in \{1, \dots, m\}$:

1. $\vec{x_n}$ is a list of different variables,
2. $\mathtt{l}_n$ is either of the form $Z_n(\vec{x}_n)$ or of the form $g_n([\vec{y}_{n1}]Z_1(\vec{x}_n, \vec{y}_{n1}), \dots, [\vec{y}_{np}]Z_1(\vec{x}_n, \vec{y}_{np}))$.

Hence every rewrite rule of a contraction scheme is a rewrite rule of a CRS, but not vice versa. For instance, a rewrite rule of a first-order term rewriting system that has in the left-hand side more than two nested function symbols, as in $f(g(h(x)))$, cannot be expressed as a contraction scheme.

Definition 4.5.4. A *CRS* is a pair $(\Sigma, \mathcal{R})$ consisting of a rewrite alphabet Σ and a set of rewrite rules over Σ.

Example 4.5.5. The representation of λ-calculus as a CRS is as follows. The rewrite alphabet consists of a unary function symbol abs and a binary function symbol app. The rewrite rule for β-reduction is written as

$$\mathsf{app}(\mathsf{abs}([x]Z(x)), Z') \to Z(Z').$$

We now define how the rewrite rules induce a rewrite relation on the set of terms. This is not completely straightforward. The definition makes use of the concepts of context and assignment. A *context* is a term with one occurrence of a distinguished variable denoted by $\Box$. As before, the convention is that this variable doesn't occur bound. For the definition of an assignment we need the following auxiliary definition of a substitute. The terminology is due to Kahrs [Kah94].

Definition 4.5.6. An *m-ary substitute* is a mapping

$$\boldsymbol{\lambda}(x_1, \dots, x_m).s : \mathsf{Terms}^m \to \mathsf{Terms}$$

with $s \in \mathsf{Terms}$, that is defined by

$$(\boldsymbol{\lambda}(x_1, \dots, x_m).s)(s_1, \dots, s_m) = s[x_1 := s_1 \dots x_m := s_m].$$

Definition 4.5.7. An *assignment* is a mapping θ that assigns to a finite number of metavariables a substitute, respecting the arity. It is extended to a homomorphism $\theta : \mathsf{Terms} \to \mathsf{Terms}$ in the following way:

1. $x^\theta = x$,
2. $([x]s)^\theta = [x]s^\theta$,
3. $(f(s_1, \ldots, s_m))^\theta = f(s_1^\theta, \ldots, s_m^\theta)$,
4. $(Z(s_1, \ldots, s_m))^\theta = \theta(Z)(s_1^\theta, \ldots, s_m^\theta)$.

Note that various kinds of problems concerning variable clashes occur. We just suppose variables to be renamed whenever necessary. Now we collected all ingredients that are necessary to define the rewrite relation of a CRS.

Definition 4.5.8. Let $(\Sigma, \mathcal{R})$ be a CRS. The *rewrite relation* on the set Terms, denoted by $\to_\mathcal{R}$, is defined as follows:

$$s \to_\mathcal{R} s'$$

if there is a context $C\Box$, an assignment θ and a rewrite rule $\mathrm{l} \to \mathrm{r}$ such that

$$s = C[\mathrm{l}^\theta] \quad \text{and} \quad s' = C[\mathrm{r}^\theta].$$

The rewrite relation is not defined on metaterms, so in particular the left-hand side of a rewrite rule cannot be rewritten to the right-hand side of that rewrite rule. For proving a critical pair lemma as for first-order rewriting and for HRSs, one would have to consider a rewrite relation on metaterms.

Example 4.5.9. We illustrate the definition of the rewrite relation of a CRS by considering a rewrite step in slow-motion. We consider the CRS for λ-calculus, given in Example 4.5.5. Consider the assignment θ with $\theta(Z) = \boldsymbol{\lambda}(y).\mathsf{app}(y, y)$ and $\theta(Z') = \mathsf{abs}([y]\mathsf{app}(y, y))$. Then we have the following rewrite step:

$$\begin{array}{rcl}
(\mathsf{app}(\mathsf{abs}([x]Z(x)), Z'))^\theta & = & \mathsf{app}(\mathsf{abs}([x](\boldsymbol{\lambda}(y).\mathsf{app}(y, y))(x)), \mathsf{abs}([y]\mathsf{app}(y, y))) \\
 & = & \mathsf{app}(\mathsf{abs}([y]\mathsf{app}(y, y)), \mathsf{abs}([y]\mathsf{app}(y, y))) \\
 & \to & \\
(Z(Z'))^\theta & = & (\boldsymbol{\lambda}(y).\mathsf{app}(y, y))(\mathsf{abs}([y]\mathsf{app}(y, y))) \\
 & = & \mathsf{app}(\mathsf{abs}([y]\mathsf{app}(y, y)), \mathsf{abs}([y]\mathsf{app}(y, y))).
\end{array}$$

A Translation. We now discuss how to represent a CRS as a pattern higher-order rewriting system with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus. In [OR94a] it is shown how to translate CRSs into HRSs, and vice versa. Since we have shown in the previous section that HRSs are very similar to higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus, it should be no surprise that we follow [OR94a] in associating a higher-order rewriting system to a CRS. We recall the main points of the translation and refer to [OR94a] for detailed proofs.

A first thing that should be noted is that CRSs are, unlike HRSs and higher-order rewriting systems, not typed. Hence all terms of a CRS will be translated into terms of type 0, where 0 can be thought of as the set of terms. This means that the obvious translation of a term $[x]s$ of a CRS into a term $x.s'$ of a higher-order rewriting system, with s' the translation of s, does not work. The solution in [OR94a] is to add an operator $\mathsf{lambda} : (0 \to 0) \to 0$, and to translate an abstraction $[x]s$ into a term $\mathsf{lambda}(x.s')$, with s' the translation of s. This is the most important point of the translation of terms of a CRS. We will give the formal definition of the translation of terms and metaterms, but to that end we first need to translate the rewrite alphabet of a CRS.

Definition 4.5.10.

1. The translation of a metavariable Z of arity m is a variable denoted by $\iota(Z)$, of type $0 \to \ldots \to 0 \to 0$, with $m+1$ times 0.
2. Let Σ be a rewrite alphabet for a CRS. The translation of a function symbol f of arity m is a rewrite operator denoted by $\iota(f)$, of type $0 \to \ldots \to 0 \to 0$, with $m+1$ times 0.

The translation of the set of metaterms is given as follows.

Definition 4.5.11. Let Σ be a rewrite alphabet of a CRS. The translation of a metaterm s over Σ, denoted by $\iota(s)$, is defined by induction on the definition of MetaTerms:

1. $\iota(x) = x$,
2. $\iota([x]s_0) = \mathsf{lambda}(\iota(x).\iota(s_0))$,
3. $\iota(f(s_1, \ldots, s_m)) = \iota(f)\iota(s_1)\ldots\iota(s_m)$,
4. $\iota(Z(s_1, \ldots, s_m)) = \iota(Z)\iota(s_1)\ldots\iota(s_m)$.

Note that every metaterm of a CRS is translated into a term of type 0 of a higher-order rewriting system. The next step is to translate the rewrite rules of a CRS. This is done in a straightforward way, using the translation of metaterms and the definition of the closure of a pair of terms which is given in Definition 4.4.5.

Definition 4.5.12. A rewrite rule $\mathtt{l} \rightarrow \mathtt{r}$ of a CRS is translated into the closure of $\iota(\mathtt{l}) \rightarrow \iota(\mathtt{r})$.

Now we can define the translation of a CRS $(\Sigma, \mathcal{R})$ as the higher-order rewriting system $(\lambda_{\overline{\eta}}^{\rightarrow}, \iota(\Sigma), \iota(\mathcal{R}))$. The following theorem expresses that this translation is adequate in the sense that the one-step rewrite relation is preserved.

Theorem 4.5.13. *Let $(\Sigma, \mathcal{R})$ be a CRS and let $(\lambda_{\overline{\eta}}^{\rightarrow}, \iota(\Sigma), \iota(\mathcal{R}))$ be its associated higher-order rewriting system. Then $s \rightarrow_{\mathcal{R}} s'$ in $(\Sigma, \mathcal{R})$ if and only if $\iota(s) \rightarrow \iota(s')$ in $(\lambda_{\overline{\eta}}^{\rightarrow}, \iota(\Sigma), \iota(\mathcal{R}))$.*

The proof of this result can be found in [OR94a]. It makes use of the translation of assignments, then, it is shown that $\iota(C[\mathtt{l}^{\theta}]) = \iota(C)[\iota(\mathtt{l}^{\theta})] = \iota(C)[\iota(\mathtt{l})^{\iota(\theta)}]$. An inspection of the proof in [OR94a] reveals that ι is in fact a morphism (in the sense of Definition 1.2.6 of Chapter 1) between CRSs and higher-order rewriting systems. In [Oos94], it is moreover shown that the translation preserves orthogonality.

4.6 Expression Reduction Systems

Expression Reduction Systems, shortly called ERSs in the sequel, were introduced by Khasidashvili around 1986. An early reference is [Kha90]. The definition of ERSs makes use of ideas of Pkhakadze [Pkh77]. ERSs are similar to CRSs, but have been introduced independently.

We mention some of the rewrite theory that is developed for ERSs. In [Kha92], Khasidashvili proves that all developments in an orthogonal ERS are finite. This is used to show that orthogonal ERSs are confluent. In [Kha94], a strategy for orthogonal ERSs is defined that yields the longest possible rewrite sequence to normal form if a term is strongly normalising, and an infinite rewrite sequence otherwise. In [GK94, GK96b] relative normalisation is studied. The terminology relative normalisation expresses that one is interested in rewrite sequences that end in some set of terms, which are considered to be (partial) results. This set of terms is not necessarily the set of normal forms but may contain terms that model partial results, like for instance head normal forms. It is shown that if such a set satisfies certain requirements, then every term not in the set contains a redex that must be contracted in any rewrite sequence ending in the set. This generalises the theory of needed redexes that is developed by Huet and Lévy for first-order term rewriting in [HL91]. ERSs with a conditional rewrite relation are studied in [KO95]. Moreover, in [GK96a] criteria for efficiency of relative normalisation are studied.

In the remainder of this section, we recall the syntax of ERSs. We fix attention to the difference with CRSs, but as we will see, although CRSs and ERSs are conceptually very close, they differ syntactically in many aspects. We conclude this section by translating an ERS into a higher-order rewriting system with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus.

Expression Reduction Systems. We suppose that there is a set denoted by Var consisting of infinitely many *variables*. Variables are written as $x, y, z, \ldots$. We suppose moreover that there is a set denoted by MetaVar consisting of infinitely many *metavariables*. Metavariables are denoted by $Z, Z', \ldots$. The difference with CRSs is that all metavariables in an ERS have arity 0. Finally, for every $m \geq 1$ there is an operator for *metasubstitution*, denoted by $(\_/\_, \ldots, \_/\_)\_$. As we will see below, it is meant to express the simultaneous substitution of m terms for m variables in some term. A *rewrite alphabet* for an ERS is a set, denoted by Σ, that consists of

1. function symbols, denoted by $f, g, h, \ldots$,
2. quantifier symbols, denoted by $\xi, \xi', \ldots$.

Every function symbol and every quantifier symbol has a fixed *arity*. The arity of a function symbol is a natural number, that expresses how many arguments the function symbol is supposed to have. The arity of a quantifier symbol is a pair of natural numbers. The first natural number in this pair expresses how many variables the quantifier symbol binds, and the second one expresses how many arguments it is supposed to have. For instance, the operator λ of λ-calculus is represented in an ERS by a quantifier of arity $(1, 1)$, since it binds one variable and takes one argument. There is moreover a notion of *scope indicator* in ERSs, used to express in which arguments of the quantifier variables are bound. We won't consider scope indicators in the sequel, but just consider an example to illustrate their use. For instance, the integral over a function f of one argument, from a term s to a term t, can be expressed using a quantifier int of arity $(1, 3)$ and scope indicator 3 as $\mathsf{int}\, x(s, t, f(x))$.

As in CRSs, metaterms and terms are distinguished.

Definition 4.6.1. The set MetaTerms of *metaterms* is defined as the smallest set that satisfies the following requirements:

1. (a) $x \in$ MetaTerms if x is a variable,
   (b) $Z \in$ MetaTerms if Z is a metavariable,
   (c) if $s_1, \ldots, s_m \in$ MetaTerms and f is a function symbol of arity m, then $f(s_1, \ldots, s_m) \in$ MetaTerms,
   (d) if $x_1, \ldots, x_m$ are different variables, $s_1, \ldots, s_n \in$ MetaTerms and if ξ is a quantifier symbol of arity (m, n), then $\xi x_1 \ldots x_m(s_1, \ldots, s_n) \in$ MetaTerms,

   (e) if $x_1, \ldots, x_m$ are different variables and $s, s_1, \ldots, s_m \in$ MetaTerms, then $(s_1/x_1, \ldots, s_m/x_m)s \in$ MetaTerms.

2. A *term* is a metaterm without occurrences of metavariables or of metasubstitutions. The set of terms is denoted by Terms.

Variables in a metaterm can be bound by a quantifier or by a metasubstitution.

Definition 4.6.2. The set of *free variables* of a metaterm s, denoted by $\mathsf{FV}(s)$, is defined by induction on the structure of s as follows.

1. $\mathsf{FV}(x) = \{x\}$,
2. $\mathsf{FV}(Z) = \emptyset$,
3. $\mathsf{FV}(f(s_1, \ldots, s_m)) = \cup_{p=1}^{p=m} \mathsf{FV}(s_p)$,
4. $\mathsf{FV}(\xi x_1 \ldots x_m(s_1, \ldots, s_n)) = \cup_{p=1}^{p=n} \mathsf{FV}(s_p) \setminus \{x_1, \ldots, x_m\}$,
5. $\mathsf{FV}((s_1/x_1, \ldots, s_m/x_m)s) = \cup_{p=1}^{p=m} \mathsf{FV}(s_p) \cup \mathsf{FV}(s) \setminus \{x_1, \ldots, x_m\}$.

A variable occurring in a metaterm s that is not free is said to be *bound* in s.

There are various definitions of a rewrite rule of an ERS that are slightly different. We will adopt the following one. Rewrite rules of ERSs are similar to rewrite rules of CRSs, but there are some syntactical differences.

Definition 4.6.3. Let Σ be a rewrite alphabet for an ERS. A *rewrite rule over* Σ is a pair of metaterms over Σ, denoted by $\mathtt{l} \to \mathtt{r}$, such that

1. $\mathtt{l}$ and $\mathtt{r}$ are closed,
2. all metavariables in $\mathtt{r}$ occur in $\mathtt{l}$,
3. $\mathtt{l}$ is not a metavariable,
4. $\mathtt{l}$ does not contain metasubstitutions.

Definition 4.6.4. An *ERS* is a pair $(\Sigma, \mathcal{R})$ consisting of a rewrite alphabet Σ and a set of rewrite rules over Σ.

We represent λ-calculus as an ERS to show the differences with the representation of λ-calculus as a CRS.

Example 4.6.5. The rewrite rule for β-reduction is in the format of ERSs written as follows:

$$\mathsf{app}(\mathsf{abs}x(Z), Z') \rightarrow (Z'/x)Z.$$

Here, app is a function symbol of arity 2 and abs is a quantifier symbol of arity $(1, 1)$, with scope indicator (1).

The rewrite rules induce a rewrite relation on the set of terms. As in the case of CRSs and HRSs, the definition of the rewrite relation makes use of the notions of a context and an assignment. A *context* in an ERS is, as a context in a CRS, a term with one occurrence of the distinguished variable $\square$. An *assignment* is a mapping $\theta : \mathsf{MetaVar} \rightarrow \mathsf{Terms}$. It is extended to a homomorphism in the following way:

1. $x^\theta = x$,
2. $Z^\theta = \theta(Z)$,
3. $(f(s_1, \ldots, s_m))^\theta = f(s_1^\theta, \ldots, s_m^\theta)$,
4. $(\xi x_1 \ldots x_m(s_1, \ldots, s_n))^\theta = \xi x_1 \ldots x_m(s_1^\theta, \ldots, s_n^\theta)$,
5. $((s_1/x_1, \ldots, s_m/x_m)s)^\theta = s^\theta[x_1 := s_1^\theta \ldots x_m := s_m^\theta]$.

It is supposed that variables are renamed whenever necessary. From the definition of an assignment, it is clear that a metasubstitution serves as a scheme for a simultaneous substitution of m terms for m variables in some term.

For the definition of the rewrite relation of an ERS yet another definition is needed, which has no counterpart in the framework of CRSs or HRSs. Let $\mathrm{l} \rightarrow \mathrm{r}$ be a rewrite rule and let $\theta : \mathsf{MetaVar} \rightarrow \mathsf{Terms}$ be an assignment. The assignment θ is said to be *admissible* for the rewrite rule $\mathrm{l} \rightarrow \mathrm{r}$ if the following holds: for every variable x occurring in $\theta(Z)$ it is the case that in the two metaterms obtained by replacing Z by x in l and in r either every occurrence of x is bound or every occurrence of x is free. Consider for example the rewrite rule $f(Z) \rightarrow (a/x)Z$. The assignment θ mapping Z to x is not admissible for this rewrite rule, since the variable x is free in $f(x)$ but bound in $(a/x)x$. In fact, no assignment mapping Z to a term having x as free variable is admissible. As another example, we consider the rewrite rule $\xi x(Z) \rightarrow f(Z)$. The assignment θ mapping Z to x is not admissible for this rewrite rule, since x is bound in $\xi x(x)$ but free in $f(x)$.

The definition of the rewrite relation of an ERS is the same as the one given for a CRS in Definition 4.5.8, with a restriction to admissible assignments. That is, we have $s \rightarrow_{\mathcal{R}} s'$ if there is a context $C\square$, a rewrite rule $\mathrm{l} \rightarrow \mathrm{r}$ and an assignment θ that is admissible for $\mathrm{l} \rightarrow \mathrm{r}$ such that $s = C[\mathrm{l}^\theta]$ and $s' = C[\mathrm{r}^\theta]$. This concludes the description of the syntax of ERSs.

In the sequel we will suppose that every assignment is admissible.

A Translation. We now discuss how to associate a higher-order rewriting system with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus to an ERS. This translation will be similar to the one for CRSs, however, there are some small differences. Again, since ERSs are untyped by nature, every term of an ERS will be translated into a term of type 0 in a higher-order order rewriting system. Since the binding power of a quantifier symbol is expressed by its arity, we won't need an operator to map terms of some arrow type to terms of type 0, as the operator lambda used in the translation of CRSs. First the rewrite alphabet of an ERS is translated as follows.

Definition 4.6.6. Let Σ be an alphabet.

1. A function symbol $f \in \Sigma$ of arity m is translated into a rewrite operator, denoted by $\iota(f)$, of type $0 \rightarrow \ldots \rightarrow 0 \rightarrow 0$ with $m+1$ times 0.
2. A quantifier symbol $\xi \in \Sigma$ of arity (m, n) is translated into a rewrite operator, denoted by $\iota(\xi)$, of type $A \rightarrow \ldots \rightarrow A \rightarrow 0$, with n times A, and A of the form $0 \rightarrow \ldots \rightarrow 0 \rightarrow 0$ with $m+1$ times 0.

The translation of terms is fairly straightforward. However, when trying to translate metaterms of an ERS we are faced with two problems, namely how to translate metavariables and how to translate metasubstitutions. In order to solve these difficulties we will make two assumptions. Below we will see that it is possible to transform a rewrite rule such that its left- and right-hand side satisfy the assumptions, without changing the rewrite relation induced by the rewrite rule.

A first problem is formed by the metasubstitutions, which may occur in right-hand sides of rewrite rules. They occur in the form $(s_1/x_1, \ldots, s_m/x_m)s$ with s an arbitrary metaterm. However, since the right-hand side of a rewrite rule is required to be closed, they can only act on the metavariables occurring in s. Therefore, we will suppose that metasubstitutions are distributed over metaterms, such that they occur only in the form $(s_1/x_1, \ldots, s_m/x_m)Z$. So for instance $(a/x, b/y)f(Z, Z')$ is transformed into $f((a/x, b/y)Z, (a/x, b/y)Z')$.

Second, it is not immediately clear how to translate metavariables. That is, they clearly must be translated into variables, but the question is of what type. Obviously a natural translation of $(s_1/x_1, \ldots, s_m/x_m)Z$ is $x_1. \ldots .x_m.Zs'_1 \ldots s'_m$ with $s'_1, \ldots, s'_m$ the translations of $s_1, \ldots, s_m$. However, the same metavariable may occur with a metasubstitution of a different arity, or without any metasubstitution at all. We will suppose that for every metaterm s there is for every metavariable Z occurring in s a list of variables, called the *substitution variables of Z in s*. Then, we suppose that if a metavariable Z occurs in s in a submetaterm of the form $(s_1/x_1, \ldots, s_m/x_m)Z$, the variables $x_1, \ldots, x_m$ all occur in the list of substitution variables of Z in s. If a metavariable Z has a list of substitution variables consisting of m variables, then Z is translated into a variable z of type $0 \rightarrow \ldots \rightarrow 0 \rightarrow 0$ with

$m+1$ times 0. We will discuss below how to find a list of substitution variables for a metavariable occurring in a rewrite rule.

The translation of metaterms is now as follows.

Definition 4.6.7. The translation of a term s, denoted by $\iota(s)$, is defined by induction on the structure of s as follows:

1. $\iota(x) = x$,
2. $\iota(Z) = zx_1 \ldots x_m$ if the substitution variables of Z in s are $x_1, \ldots, x_m$,
3. $\iota(f(s_1, \ldots, s_m)) = \iota(f)\iota(s_1) \ldots \iota(s_m)$,
4. $\iota(\xi x_1 \ldots x_m(s_1, \ldots, s_n)) =$ $\iota(\xi)(\iota(x_1). \ldots .\iota(x_m).\iota(s_1)) \ldots (\iota(x_1). \ldots .\iota(x_m).\iota(s_n))$,
5. if the substitution variables of Z in s are $y_1, \ldots, y_n$, then $\iota((s_1/x_1, \ldots, s_m/x_m)Z) = zt_1 \ldots t_n$ with $t_p = \begin{cases} \iota(s_q) & \text{if } y_p = x_q, \\ y_p & \text{otherwise.} \end{cases}$

We first comment on the part that is concerned with the translation of terms only. In the sequel we won't mention explicitly that a variable is the translation of itself. Note that $\iota(s) : 0$ for every term s. Further, the binding of variables by a quantifier is distributed over its arguments. For instance, $\xi x(s_1, s_2)$ is translated into $\iota(\xi)(x.\iota(s_1))(x.\iota(s_2))$.

Now we will discuss how to obtain a list of substitution variables for a rewrite rule. Let $\mathtt{l} \to \mathtt{r}$ be a rewrite rule in an ERS. Let Z be a metavariable occurring in $\mathtt{l}$. We will say that $x_1, \ldots, x_m$ is a list of substitution variables for $\mathtt{l} \to \mathtt{r}$ if it is a list of substitution variables both for $\mathtt{l}$ and for $\mathtt{r}$. A rewrite rule $\mathtt{l} \to \mathtt{r}$ is transformed as follows. First, metasubstitutions are made more economic by leaving out s_p/x_p from a metasubstitution $(s_1/x_1, \ldots, s_m/x_m)Z$ if there is an occurrence of Z in $\mathtt{l} \to \mathtt{r}$ where x_p is not bound. For instance, we transform the ERS rewrite rule $\xi xy(Z) \to (a/x, b/z)f(Z)$ into $\xi xy(Z) \to f((a/x)Z)$. Note that if θ is an admissible assignment for the rewrite rule $\xi xy(Z) \to (a/x, b/z)f(Z)$, then $\theta(Z)$ does not contain a free occurrence of the variable z. We claim that such a transformation does not change the rewrite relation induced by the rewrite rule, but we do not prove this in detail. It is essential that all assignments are supposed to be admissible. Now the list of substitution variables of Z in $\mathtt{l} \to \mathtt{r}$ consists of all variables that are bound at every occurrence of Z in $\mathtt{l}$ and $\mathtt{r}$. Using the list of substitution variables of a metavariable in a rewrite rule, the translation of a rewrite rule is defined as follows.

Definition 4.6.8. The translation of a rewrite rule $\mathtt{l} \to \mathtt{r}$ of an ERS is the closure of $\iota(\mathtt{l}) \to \iota(\mathtt{r})$, using the list of substitution variables for $\mathtt{l} \to \mathtt{r}$ for the translation of metaterms.

Example 4.6.9. As an example we consider the rewrite rule

$$\xi xy(Z, Z') \rightarrow_{\mathsf{R}} (a/x, b/y')f(Z).$$

First we slightly modify the rewrite rule which yields $\xi xy(Z, Z') \rightarrow f((a/x)Z)$. The list of substitution variables of Z for this rewrite rule is x and for Z' it is xy. The order in which they occur in the list is not important, but it is important that the order is fixed. The translation of R is

$$\xi(x.y.zx)(x.y.z'xy) \rightarrow f(za).$$

Note that for an admissible assignment θ the term $\theta(Z)$ does not contain a free occurrence of y'.

Again, we have that the translation preserves the rewrite relation in the sense that if $s \rightarrow_{\mathcal{R}} s'$ in an ERS $(\Sigma, \mathcal{R})$, then $\iota(s) \rightarrow_{\mathcal{R}} \iota(s')$ in its associated higher-order rewriting system $(\lambda^{\rightarrow}_{\overline{\eta}}, \iota(\Sigma), \iota(\mathcal{R}))$. To show this in full detail requires quite some more work.

4.7 Interaction Systems

This section is concerned with the class of Interaction Systems, in the sequel called ISs, that is defined by Asperti and Laneve in [Lan93] and [AL94]. ISs are introduced to study the theory of optimal reductions as defined by Lévy [Lév78] in a setting that is more general than λ-calculus. On the one hand, ISs from a generalisation of the Interaction Nets as defined by Lafont in [Laf90]. Interaction Nets generalise proof nets of linear logic by admitting arbitrary pairs of constructors and destructors. A rewrite rule of an Interaction Net should satisfy three requirements, one of which is the following: every variable occurring in a rewrite rule occurs exactly once in the left-hand side and exactly once in the right-hand side. ISs are obtained from Interaction Nets by dropping this constraint. On the other hand, ISs form a proper subclass of the class of CRSs, which will be clear from the presentation of the syntax of ISs below.

For results concerning optimal reductions in the setting of ISs we refer to [Lan93], [AL94] and [AL96].

In this section we first recall the syntax of ISs and then we explain how an IS can be represented as a higher-order rewriting system with $\lambda^{\rightarrow}_{\overline{\eta}}$ as substitution calculus. Since ISs form a subclass of CRSs, we can translate an IS into a higher-order rewriting system in the same way as we translate a CRS. However, as will be clear from the translation presented below, ISs are closer to higher-order rewriting systems than CRSs, and hence the natural translation of ISs into higher-order rewriting systems is not an instance of the translation of CRSs into higher-order rewriting systems with $\lambda^{\rightarrow}_{\overline{\eta}}$ as substitution calculus.

Interaction Systems. We suppose that there is a set Var consisting of infinitely many *variables* and a set MetaVar consisting of infinitely many *metavariables*. Variables are denoted by $x, y, z, \ldots$ and metavariables are denoted by $Z, Z', \ldots$. As in ERSs, all metavariables have arity 0. Moreover, we suppose that there is for every $m \geq 1$ an operator for metasubstitution, denoted by $\_[\_/\_, \ldots, \_/\_]$, with m times $\_/\_$. A *rewrite alphabet* for an IS is a set, denoted by Σ, consisting of *forms*. Every form is either a *constructor* or a *destructor*. Constructors are denoted by $c, c', \ldots$ and destructors are denoted by $d, d', \ldots$. Forms of which it is not specified whether they are constructors or destructors are denoted by $f, f', \ldots$. Every form is equipped with a fixed *arity*, which is a finite list of natural numbers. A form of arity $(m_1 \ldots m_n)$ should have n arguments, and binds m_p variables in the pth argument. The arity of a destructor must be of the form $(0m_2 \ldots m_n)$. For example, the operator λ of λ-calculus, which is a constructor, is a form of arity (1), and application of λ-calculus is a destructor of arity (00). The important difference between ISs on the one hand and CRSs and ERSs on the other hand is that ISs are constructor systems, following the principle of introduction and elimination rules in logical systems. Metaterms and terms are defined as follows.

Definition 4.7.1. Let Σ be a rewrite alphabet for an IS.

1. The set of *metaterms over* Σ, denoted by MetaTerms, is the smallest set that satisfies the following:

   (a) $x \in$ MetaTerms for every variable x,

   (b) $Z \in$ MetaTerms for every metavariable Z,

   (c) if $s_1, \ldots, s_n \in$ MetaTerms and f is a form of arity $(m_1 \ldots m_n)$, then $f(x_1^1. \ldots .x_{m_1}^1.s_1, \ldots, x_1^n. \ldots .x_{m_n}^n.s_n) \in$ MetaTerms,

   (d) if $s, s_1, \ldots, s_m \in$ MetaTerms and $x_1, \ldots, x_m$ are different variables, then $s[s_1/x_1, \ldots, s_m/x_m] \in$ MetaTerms.

2. A *term* is a metaterm without metavariables or metasubstitutions. The set of terms is denoted by Terms.

So terms and metaterms of ISs are very similar to terms and metaterms of ERSs. In the sequel, we abbreviate $x_1^p. \ldots .x_{m_p}^p.s$ by $\vec{x}_{m_p}^p.s$. Rewrite rules in ISs are more restricted than in ERSs. One of the reasons is that they obey a constructor-destructor discipline. Moreover there is a restriction on the use of metasubstitutions that makes the translation into higher-order rewriting systems easier, as we will see below.

Definition 4.7.2. Let Σ be a rewrite alphabet. A *rewrite rule over* Σ is a pair of closed metaterms, denoted by $\mathtt{l} \to \mathtt{r}$, such that

1. every metavariable in $\mathtt{r}$ occurs in $\mathtt{l}$,
2. $\mathtt{l}$ is of the form $d(c(\vec{x}^1_{m_1}.Z_1, \ldots, \vec{x}^n_{m_n}.Z_n), \vec{y}^2_{p_2}.Z'_2, \ldots, \vec{y}^q_{p_q}.Z'_q)$ with all metavariables different,
3. $\mathtt{r}$ is an element of the set $\mathtt{r}(\mathtt{l})$, which is defined as the smallest set that satisfies the following:
   (a) $x \in \mathtt{r}(\mathtt{l})$ for any variable x,
   (b) if $\mathtt{r}_1, \ldots, \mathtt{r}_n \in \mathtt{r}(\mathtt{l})$ and f is a form of arity $(m_1 \ldots m_n)$, then $f(\vec{x}^1_{m_1}.\mathtt{r}_1, \ldots, \vec{x}^n_{m_n}.\mathtt{r}_n) \in \mathtt{r}(\mathtt{l})$,
   (c) $Z_{n'}[\mathtt{r}_1/z^n_1, \ldots, \mathtt{r}_n/z^n_{m_n}] \in \mathtt{r}(\mathtt{l})$ if $\vec{z}^n_{m_n}.Z$ occurs in $\mathtt{l}$.

The restriction on the form of the right-hand side of a rewrite rules ensures that substitutions do not introduce bound variables. For instance, $d(c(Z)) \to Z[a/x]$ is not allowed as a rewrite rule, since x is not bound at the occurrence of Z in the left-hand side.

Definition 4.7.3. An *IS* is a pair $(\Sigma, \mathcal{R})$ consisting of a rewrite alphabet Σ and a set of rewrite rules over Σ such that there is at most one rewrite rule for every pair consisting of a constructor and a destructor.

Note that an IS is by definition orthogonal. In order to compare the syntax of ISs with the syntax of previously discussed rewriting systems, we present λ-calculus as an IS.

Example 4.7.4. The rewrite alphabet of the IS representing λ-calculus consists of two forms: a constructor abs of arity (1) and a destructor app of arity (00). The rewrite rule for β-reduction is written as follows:

$$\mathsf{app}(\mathsf{abs}(x.Z), Z') \to Z[Z'/x].$$

Another frequently occurring example of an IS is the one with a rewrite alphabet consisting of a constructor λ of arity (1) and a destructor μ of arity (0). Its rewrite rule is as follows:

$$\mu(\lambda(x.Z)) \to Z[\mu(\lambda(y.Z[y/x]))/x].$$

The rewrite relation of an IS is defined in the same way as the rewrite relation of an ERS, with the difference that the restriction to admissible assignments is not necessary, since the shape of the rewrite rules is so that every assignment is admissible.

A Translation. We now discuss how to associate a higher-order rewriting system with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus to an IS. Like CRSs and ERSs, ISs are untyped. Hence all terms of an IS will be translated into terms of type 0 of a higher-order rewriting system. The rewrite alphabet of an IS is translated as follows.

Definition 4.7.5. A form f of arity $(m_1 \dots m_n)$ is translated into a simply typed rewrite operator denoted by $\iota(f)$, of type $A_1 \to \dots \to A_n \to 0$, with $A_p = 0 \to \dots \to 0$ with $m_p + 1$ times 0.

We first present the translation of terms, which is quite straightforward.

Definition 4.7.6. Let Σ be a rewrite alphabet for an IS. The translation of a term s over Σ into a simply typed term, denoted by $\iota(s)$, is defined by induction on the structure of s as follows:

1. $\iota(x) = x$,
2. $\iota(f(\vec{x}^1_{m_1}.s_1, \dots, \vec{x}^n_{m_n}.s_n)) = f(\vec{x}^1_{m_1}.\iota(s_1)) \dots (\vec{x}^n_{m_n}.\iota(s_n))$.

The translation of a rewrite rule is obtained in two steps.

Definition 4.7.7. Let $(\Sigma, \mathcal{R})$ be an IS. Let $\mathrm{l} \to \mathrm{r}$ be a rewrite rule in $\mathcal{R}$ with

$$\mathrm{l} = d(c(\vec{x}^1_{m_1}.Z_1, \dots, \vec{x}^n_{m_n}.Z_n), \vec{y}^2_{p_2}.Z'_2, \dots, \vec{y}^q_{p_q}.Z'_q)$$

1. (a) The translation of the left-hand side, denoted by $\iota(\mathrm{l})$, is

   $$\iota(d)(\iota(c)(\vec{x}^1_{m_1}.z_1\vec{x}^1_{m_1}) \dots (\vec{x}^n_{m_n}.z_n\vec{x}^n_{m_n}))(\vec{y}^2_{p_2}.z'_2\vec{y}^2_{p_2}) \dots (\vec{y}^q_{p_q}.z'_q\vec{y}^q_{p_q})$$

   (b) The translation of the right-hand side, denoted by $\iota(\mathrm{r})$, is defined by induction on the structure of r as follows:

   i. $\iota(x) = x$,
   ii. $\iota(f(\vec{x}^1_{m_1}.\mathrm{r}_1, \dots, \vec{x}^n_{m_n}.\mathrm{r}_n)) = f(\vec{x}^1_{m_1}.\iota(\mathrm{r}_1)) \dots (\vec{x}^n_{m_n}.\iota(\mathrm{r}_n))$,
   iii. $\iota(Z_{n'}[\mathrm{r}_1/x^{n'}_1, \dots, \mathrm{r}_m/x^{n'}_{m_{n'}}]) = z_{n'}\,\iota(\mathrm{r}_1) \dots \iota(\mathrm{r}_m)$.

2. The translation of the rewrite rule $\mathrm{l} \to \mathrm{r}$ is obtained by taking the closure of $\iota(\mathrm{l}) \to \iota(\mathrm{r})$.

Example 4.7.8. Consider the rewrite rules given in Example 4.7.4. The translation of the rewrite rule for β-reduction is

$$z.z'.\mathsf{app}(\mathsf{abs}(x.zx), z') \to z.z'.zz'.$$

The translation of the rewrite rule for μ and λ is

$$z.\mu(\lambda(x.zx)) \to z.z(\mu(\lambda(y.zy))).$$

We have that $s \to s'$ in an IS $(\Sigma, \mathcal{R})$ implies that $\iota(s) \to \iota(s')$ in its associated higher-order rewriting system $(\lambda_{\overline{\eta}}^{\rightarrow}, \iota(\Sigma), \iota(\mathcal{R}))$. Again, we leave the details to the reader.

4.8 PCF

In this section we represent PCF as a higher-order rewriting system. PCF, which stands for Programming Language with Computable Functions, is introduced by Plotkin in [Plo77]. It is simply typed λ-calculus with a fixpoint operator and constants that are used to model elementary operations on booleans and natural numbers. We include the representation of PCF as a higher-order rewriting system to show that also typed rewriting systems can be represented as a higher-order rewriting system. As substitution calculus we will take system F, introduced by Girard in [Gir72] and independently by Reynolds in [Rey74]. First we recall the syntax of PCF and then we represent PCF as a higher-order rewriting system with system F as substitution calculus.

PCF. The *types* of PCF are the simple types, defined in Definition 2.5.1 of Chapter 2, built from two base types: $\mathtt{N}$ representing the natural numbers and $\mathtt{B}$ representing the booleans. The typing relation is denoted by :. We suppose that for every type A there is a set denoted by Var_A consisting of infinitely many *variables of type* A. The set consisting of all variables is denoted by Var. The *alphabet* of PCF consists of the following simply typed constants:

1. for every $m \geq 0$ there is a constant $\mathsf{m} : \mathtt{N}$,
2. $\mathsf{S} : \mathtt{N} \rightarrow \mathtt{N}$,
3. $\mathsf{P} : \mathtt{N} \rightarrow \mathtt{N}$,
4. $\mathsf{T} : \mathtt{B}$,
5. $\mathsf{F} : \mathtt{B}$,
6. $\mathsf{Z} : \mathtt{N} \rightarrow \mathtt{B}$,
7. $\mathsf{cond}_{\mathtt{B}} : \mathtt{B} \rightarrow \mathtt{B} \rightarrow \mathtt{B} \rightarrow \mathtt{B}$,
8. $\mathsf{cond}_{\mathtt{N}} : \mathtt{B} \rightarrow \mathtt{N} \rightarrow \mathtt{N} \rightarrow \mathtt{N}$,
9. for every type A there is a constant $\mathsf{Y}_A : (A \rightarrow A) \rightarrow A$.

The terms of type A of PCF are built over the alphabet given above as defined in Definition 2.5.3 of Chapter 2. We denote terms of PCF by $s, t, r, \ldots$.

The rewrite rules of PCF are the following:

$$\begin{array}{rcl}
\mathsf{cond_B T}st & \to & s \\
\mathsf{cond_B F}st & \to & t \\
\mathsf{cond_N T}st & \to & s \\
\mathsf{cond_N F}st & \to & t \\
\mathsf{Sm} & \to & \mathsf{m}+1 \\
\mathsf{P}(\mathsf{m}+1) & \to & \mathsf{m} \\
\mathsf{Z0} & \to & \mathsf{T} \\
\mathsf{Z}(\mathsf{m}+1) & \to & \mathsf{F} \\
\mathsf{Y}_A s & \to & s(\mathsf{Y}_A s) \\
(\lambda x.s)t & \to & s[x := t]
\end{array}$$

The Substitution Calculus System F. We briefly recall the syntax of System F from [GLT89]. We suppose that there are base types and infinitely many type variables.

Definition 4.8.1. The set of *types* of system F, denoted by Types, is defined as the smallest set that satisfies the following:

1. $\alpha \in$ Types if α is a type variable,
2. $0 \in$ Types if 0 is a base type,
3. if $A, B \in$ Types, then $A \to B \in$ Types,
4. if $A \in$ Types and α is a type variable, then $\Pi\alpha.A \in$ Types.

We suppose that for every type A there are infinitely many variables of type A, written as $x^A, y^A, z^A, \ldots$. The set of terms is defined as follows.

Definition 4.8.2. The set of *terms of type* A of system F, denoted by Terms^A, is defined as the smallest set that satisfies the following:

1. $x^A \in \mathsf{Terms}^A$ for every variable x^A of type A,
2. if $A = A_0 \to A_1$ and x^{A_0} is a variable of type A_0 and $s \in \mathsf{Terms}^{A_1}$, then $\lambda x^{A_0}.s \in \mathsf{Terms}^A$,
3. if $s \in \mathsf{Terms}^{B\to A}$ and $t \in \mathsf{Terms}^B$, then $st \in \mathsf{Terms}^A$,
4. if $A = \Pi\alpha.A_0$ and if $s \in \mathsf{Terms}^{A_0}$, then $\Lambda\alpha.s \in \mathsf{Terms}^A$, provided that α does not occur free in the type of a free variable of s,

5. if $A = A_0[\alpha := A_1]$ with A_0 and A_1 types, and if $s \in \mathsf{Terms}^{\Pi\alpha.A_0}$, then $sA_1 \in \mathsf{Terms}^{A}$.

The set of all terms is denoted by Terms.

The rewrite relation on the set of terms is generated by the following two rules for β-reduction:

$$\begin{array}{rcl} (\lambda x^A.s)t & \to & s[x := t], \\ (\Lambda\alpha.s)A & \to & s[\alpha := A]. \end{array}$$

We will suppose that all terms are in normal form with respect to $\to_{\overline{\eta}}$.

PCF as a Higher-Order Rewriting System. We represent PCF as a higher-order rewriting system with system F as substitution calculus. The alphabet of PCF is as given above, with the following changes:

1. $\mathsf{Y} : \Pi\alpha.(\alpha \to \alpha) \to \alpha$,
2. $\mathsf{abs} : \Pi\alpha.\beta.(\alpha \to \beta) \to \alpha \to \beta$,
3. $\mathsf{app} : \Pi\alpha.\beta.(\alpha \to \beta) \to \alpha \to \beta$.

So the fixpoint operator and the operators for abstraction and application are polymorphically typed. We will write Y_A, abs_{AB} and app_{AB} instead of $\mathsf{Y}A$, $\mathsf{abs}AB$ and $\mathsf{app}AB$. Further, we write $\mathsf{app}(\mathsf{app}\ \mathsf{cond}_{\mathbb{B}}st)r$ and $\mathsf{app}(\mathsf{app}\ \mathsf{cond}_{\mathbb{N}}st)r$ instead of explicitly mentioning the type of the applications. The rewrite rules of PCF now take the following form:

$$\begin{array}{rcl}
\lambda y^{\mathbb{B}}.\lambda z^{\mathbb{B}}.\mathsf{app}(\mathsf{app}\ \mathsf{cond}_{\mathbb{B}}\mathsf{T}y^{\mathbb{B}})z^{\mathbb{B}} & \to & \lambda y^{\mathbb{B}}.\lambda z^{\mathbb{B}}.y^{\mathbb{B}} \\
\lambda y^{\mathbb{B}}.\lambda z^{\mathbb{B}}.\mathsf{app}(\mathsf{app}\ \mathsf{cond}_{\mathbb{B}}\mathsf{F}y^{\mathbb{B}})z^{\mathbb{B}} & \to & \lambda y^{\mathbb{B}}.\lambda z^{\mathbb{B}}.z^{\mathbb{B}} \\
\lambda y^{\mathbb{N}}.\lambda z^{\mathbb{N}}.\mathsf{app}(\mathsf{app}\ \mathsf{cond}_{\mathbb{N}}\mathsf{T}y^{\mathbb{N}})z^{\mathbb{N}} & \to & \lambda y^{\mathbb{N}}.\lambda z^{\mathbb{N}}.y^{\mathbb{N}} \\
\lambda y^{\mathbb{N}}.\lambda z^{\mathbb{N}}.\mathsf{app}(\mathsf{app}\ \mathsf{cond}_{\mathbb{N}}\mathsf{F}y^{\mathbb{N}})z^{\mathbb{N}} & \to & \lambda y^{\mathbb{N}}.\lambda z^{\mathbb{N}}.z^{\mathbb{N}} \\
\mathsf{appSm} & \to & \mathsf{m}+1 \\
\mathsf{appP}(\mathsf{m}+1) & \to & \mathsf{m} \\
\mathsf{appZ0} & \to & \mathsf{T} \\
\mathsf{appZ}(\mathsf{m}+1) & \to & \mathsf{F} \\
\Lambda\alpha.\lambda y^{\alpha\to\alpha}.\mathsf{app}_{((\alpha\to\alpha)\to\alpha)(\alpha\to\alpha)}\mathsf{Y}_\alpha y^{\alpha\to\alpha} & \to & \\
\Lambda\alpha.\lambda y^{\alpha\to\alpha}.\mathsf{app}_{(\alpha\to\alpha)(\alpha)}y^{\alpha\to\alpha}(\mathsf{app}_{((\alpha\to\alpha)\to\alpha)(\alpha\to\alpha)}\mathsf{Y}_\alpha y^{\alpha\to\alpha}) & & \\
\Lambda\alpha.\Lambda\beta.\lambda y^{\alpha\to\beta}.\lambda z^{\alpha}.\mathsf{app}_{\alpha\beta}(\mathsf{abs}_{\alpha\beta}\lambda x^{\alpha}.y^{\alpha\to\beta}x^{\alpha})z^{\alpha} & \to & \\
\Lambda\alpha.\Lambda\beta.\lambda y^{\alpha\to\beta}.\lambda z^{\alpha}.y^{\alpha\to\beta}z^{\alpha} & &
\end{array}$$

In order to be able to define the rewrite relation, it must be possible to abstract over terms of the form $\Lambda\alpha.s$. Therefore yet another level of abstraction is necessary, which we will not elaborate here.

Chapter 5

Confluence for Higher-Order Rewriting

In this chapter we consider pattern higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus. We will say shortly 'higher-order rewriting system' instead of 'pattern higher-order rewriting system with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus'.

Confluence is an important property of rewriting systems, because it ensures the uniqueness of normal forms and it provides a method to establish consistency. In this chapter we prove that all weakly orthogonal higher-order rewriting systems are confluent. The proof makes use of the concept of parallel reduction. A parallel reduction step is obtained by contracting a set of redex occurrences in a term simultaneously. Confluence of weakly orthogonal higher-order rewriting systems is also proved by van Oostrom in [Oos94] but in a different way, namely by first proving that all developments are finite. Short versions of both proofs are also reported in [OR94b]. By admitting trivial critical pairs the result that weakly orthogonal higher-order rewriting systems are confluent extends earlier confluence results.

The organisation of this chapter is as follows. In Section 5.1 we discuss the two main methods of proving confluence of orthogonal rewriting systems: one using developments and one using parallel reduction. In this section we also discuss related work concerning confluence of higher-order rewriting. In Section 5.2 the relation $\multimap\!\!\!\rightarrow$ for parallel reduction is defined, and we prove some elementary properties of $\multimap\!\!\!\rightarrow$. In Section 5.3 we give a short proof of confluence of orthogonal higher-order rewriting systems using parallel reduction. Then, in Section 5.4 we extend this result by showing that weakly orthogonal higher-order rewriting systems are confluent. This result implies the result of Section 5.3. We have nevertheless included the proof for the orthogonal case since it clearly shows the essence of parallel reduction.

5.1 Confluence

There are two important methods to prove confluence of orthogonal rewriting systems: by developments and by parallel reduction. Most proofs of confluence of orthogonal rewriting systems known in the literature are by one of those two methods. At some places the requirement that the rewriting systems must be non-overlapping is relaxed. In this section we briefly discuss the proof method by developments, and next the proof method by parallel reduction. First we give two well-known examples to illustrate that the restriction to orthogonal systems is a reasonable one. The simple rewriting system defined by the rewrite rules $a \to b$ and $a \to c$ illustrates that an overlapping higher-order rewriting system is not necessarily confluent. The term rewriting system defined by the rewrite rules

$$\begin{aligned} x.fxx &\to x.a \\ x.f(gx)x &\to x.b \\ c &\to gc \end{aligned}$$

given by Huet in [Hue80] illustrates that a non-overlapping rewriting system is not necessarily confluent if it is not left-linear: we have $fcc \to f(gc)c \to b$ and $fcc \to a$, so the term fcc can be rewritten to two different normal forms. In [Klo80], Klop gives an example of a non-overlapping term rewriting system that is not confluent, defined by three rewrite rules of which only one is not left-linear. This rewriting system has moreover the property of uniqueness of normal forms.

As it turns out, the restriction to orthogonal higher-order rewriting systems can be relaxed to requiring them only to be weakly orthogonal.

Confluence via Developments. A development is a rewrite sequence in which only redex occurrences are contracted that are residuals of redex occurrences in the initial term. Developments are defined for functional rewriting systems in Section 1.2 of Chapter 1, and for λ-calculus with β-reduction in Section 2.3 of Chapter 2. In Section 1.2 of Chapter 1 it is explained that confluence can be established if we can construct the orthogonal projection of a rewrite sequence over some rewrite step. We recall also from the discussion in that section that it is possible to construct the orthogonal projection if all complete developments are finite and end in the same term. This is the principle of a proof of confluence via developments. The most difficult step of the proof is to show that all developments are finite.

The first proof of confluence by developments of λ-calculus with β-reduction is due to Church and Rosser [CR36]. It concerns λI-calculus. Thereafter, various proofs of finiteness of developments of the full λ-calculus with β-reduction have been given. In Section 2.3, some of these proofs are mentioned.

In [Ros73], Rosen proves confluence of orthogonal (although a different terminology is employed there) subtree replacement systems using developments. Every first-order term rewriting system is a subtree replacement system.

In [Klo80], Klop proves that orthogonal CRSs are confluent. This proof is structured as follows. First a weak normalisation result is obtained, stating that every set of redex occurrences can be developed in a finite way. Then local confluence is shown. This is the most complicated part of the proof. These two results entail confluence of orthogonal CRSs. The confluence result is then used to show that all developments are finite.

In [Kha92], Khasidashvili shows that orthogonal ERSs are confluent by first proving finiteness of developments. This result is close to the result by Klop, however it is not exactly the same, see Section 4.6. Moreover, the proof is different from the one by Klop, since for instance finiteness of developments is proved directly.

Van Oostrom proves in [Oos94] confluence of weakly orthogonal higher-order rewriting systems using developments. This extends the results mentioned above, since an orthogonal rewriting system is weakly orthogonal but not necessarily vice versa. Finiteness of developments is proved making use of the properties of the substitution calculus, in particular of the property of strong normalisation. As we already mentioned, a short version of both this proof and the one presented in Section 5.4 can be found in [OR94b].

Confluence via Parallel Reduction. A parallel reduction step $s \mathrel{-\!\!\circ\!\!\rightarrow} s'$ is obtained by contracting a number of simultaneous redex occurrences in s simultaneously. Since simultaneous redex occurrences can very well be nested, this notion of parallel reduction is essentially different from the one employed by Huet in [Hue80]. The proof method by parallel reduction consists of two steps:

1. show that $\mathrel{-\!\!\circ\!\!\rightarrow}^* = \twoheadrightarrow$,

2. show that the relation $\mathrel{-\!\!\circ\!\!\rightarrow}$ satisfies the diamond property.

The relation $\mathrel{-\!\!\circ\!\!\rightarrow}$ has the diamond property if for all terms s, s', s'' such that $s \mathrel{-\!\!\circ\!\!\rightarrow} s'$ and $s \mathrel{-\!\!\circ\!\!\rightarrow} s''$, there exists a term t with $s' \mathrel{-\!\!\circ\!\!\rightarrow} t$ and $s'' \mathrel{-\!\!\circ\!\!\rightarrow} t$. See Definition 1.1.20 of Chapter 1. That it indeed suffices to prove 1 and 2 above in order to infer confluence of the rewrite relation is expressed in Lemma 1.1.25 of Chapter 1.

The method of parallel reduction is due to Tait and Martin-Löf, and is also called the Tait and Martin-Löf method. Tait proved confluence of combinatory logic by parallel reduction, and Martin-Löf adapted the proof by Tait to the case of λ-calculus with β-reduction. The proofs by Tait and Martin-Löf are unpublished. The proof of confluence of λ-calculus with β-reduction by parallel reduction can be found in [Ste72], in [Bar84] and in [HS86]. A proof of confluence using parallel reductions and comments about its history can be found in [Ros82].

The result by Tait and Martin-Löf is extended to various larger classes of rewriting systems. The first such extension is due to Aczel. In [Acz78], Aczel introduces the class of contraction schemes, and proves that orthogonal contraction schemes are confluent by parallel reduction. The class of contraction schemes contains λ-calculus with β-reduction and a subclass of the class of first-order term rewriting systems. See Section 4.5 of Chapter 4 for a short discussion on the expressivity of contraction schemes. It is moreover worth mentioning that the relation for parallel reduction used by Aczel is not the same as the one used by Tait and Martin-Löf. We illustrate the difference for the case of λ-calculus with β-reduction. The essential clause in the definition of parallel reduction as employed by Tait and Martin-Löf, and as considered in this thesis, is the following: if $M \multimap\!\!\to M'$ and $N \multimap\!\!\to N'$, then $(\lambda x.M)N \multimap\!\!\to M'[x := N']$. In the definition given by Aczel, the essential clause is: if $M \multimap\!\!\to \lambda x.M'$ and $N \multimap\!\!\to N'$, then $MN \multimap\!\!\to M'[x := N']$. A parallel reduction step in λ-calculus corresponds to a β-development, whereas a parallel reduction step à la Aczel corresponds to a β-superdevelopment. In Section 2.4 of Chapter 2 superdevelopments are defined and proved to be strongly normalising. So the relation used by Aczel is stronger than the relation for parallel reduction.

Nipkow proves in [Nip93b] that orthogonal HRSs are confluent, using parallel reduction. This proof is also reported in [MN94], which contains more results, like for instance a critical pair lemma, and a proof of confluence of orthogonal HRSs using a relation for parallel reduction à la Aczel and a variation on the proof method due to Takahashi (which is discussed below). In [Raa93], a proof of confluence of orthogonal CRSs using parallel reduction à la Aczel is given. From the discussion in Section 4.5 and Section 4.4 of Chapter 4, it is clear that the result that orthogonal HRSs are confluent is strongly related to the result that orthogonal CRSs are confluent. Moreover, both in [Nip93b] and in [Raa93] the proof makes use of parallel reduction, the only difference being that in [Raa93] the relation for parallel reduction à la Aczel is employed. Clearly, this situation is unsatisfactory. In [OR94a, OR93] these two confluence results are related: it is shown that confluence of a HRS implies confluence of its associated CRS and vice versa.

Takahashi proves in [Tak93] confluence of so-called semi-orthogonal λ-calculi with conditional rules using parallel reduction. Semi-orthogonal rewriting systems are left-linear. Moreover, a semi-orthogonal rewriting system should satisfy the following requirement. Let $u, u_1 \ldots, u_m$ be redex occurrences in a term such that $u \preceq u_1 \preceq \ldots \preceq u_m$, further $u \sharp u_1, \ldots, u \sharp u_m$ and moreover $u_p \parallel u_q$ for all $p, q \in \{1, \ldots, m\}$. Then the result of contracting u must be the same a the result of contracting $u_m, \ldots, u_1$ in that order. The notion of semi-orthogonality is weaker than the one of orthogonality, but stronger than the one of weak orthogonality. The first-order term rewriting system defined by the rewrite rules

$$f(g(a)) \rightarrow f(g(h(a)))$$

$$\begin{array}{rcl} g(x) & \rightarrow & g(h(a)) \\ a & \rightarrow & h(a) \end{array}$$

is an example of a rewriting system that is weakly orthogonal but not semi-orthogonal. In [Tak95], Takahashi proves various classical results in λ-calculus using parallel reduction. She gives for instance proofs of confluence of system F, of the standardisation theorem for λ-calculus with β-reduction and of the post-ponement theorem for η-reduction. Moreover, in the same paper [Tak95] she gives a very short and elegant proof of confluence of λ-calculus with β-reduction. It differs from the other proofs mentioned above in the following sense. In most proofs of the diamond property of $\mathrel{-\!\!\circ\!\!\!\rightarrow}$, given three terms s, s' and s'' such that $s \mathrel{-\!\!\circ\!\!\!\rightarrow} s'$ and $s \mathrel{-\!\!\circ\!\!\!\rightarrow} s''$, one shows that a term t exists such that $s' \mathrel{-\!\!\circ\!\!\!\rightarrow} t$ and $s'' \mathrel{-\!\!\circ\!\!\!\rightarrow} t$. The term t depends on s' and on s''. Takahashi defines for every term s a term s^* such that $s' \mathrel{-\!\!\circ\!\!\!\rightarrow} s^*$ whenever $s \mathrel{-\!\!\circ\!\!\!\rightarrow} s'$. We adapt this proof to the case of orthogonal higher-order rewriting systems in Section 5.3.

Huet gives in [Hue94] a proof of confluence of λ-calculus with β-reduction using parallel reduction, that is fully formalised using the Coq proof assistant Gallina. Nipkow describes in [Nip96] a formalisation of a proof of confluence of λ-calculus with β- and η-reduction in an implementation of higher-order logic in the theorem prover Isabelle. An interesting observation in this paper is that although the proof method due to Takahashi gives rise to a shorter proof than the usual one using parallel reduction, it is not easier to formalise.

Comparison. The relationship between the proof method by developments and the proof method by parallel reduction is as follows. A parallel reduction step is a complete development of a set of redex occurrences, in which contractions are performed in an inside-out way.

Both proof methods have their particular properties that can be considered as advantages or disadvantages, depending on the taste and the needs of the reader. Proofs by parallel reduction are short, because besides the notion of parallel reduction hardly any additional concepts are needed, and moreover the proof that parallel reduction satisfies the diamond property makes use of just a few auxiliary lemmata. Proofs by developments are usually longer, since they need the concept of residual and the result that all developments are finite. However, the auxiliary results, and especially the result that all developments are finite, can be used for various other purposes.

5.2 Parallel Reduction

In the remainder of this chapter we adopt the following conventions. We consider left-linear pattern higher-order rewriting systems. All preterms are supposed to be *in $\overline{\eta}$-normal form. We use the formalisation of redex and redex occurrence as given in Definition 4.2.25 of Chapter 4, that is, a redex is a pair consisting of a context and a rewrite rule.*

In this section we define the relation $-\!\!\circ\!\!\!\rightarrow$ for parallel reduction and we prove some elementary properties of $-\!\!\circ\!\!\!\rightarrow$.

Definition 5.2.1. Let $\mathcal{H}$ be a left-linear higher-order rewriting system. The binary relation for *parallel reduction*, denoted by $-\!\!\circ\!\!\!\rightarrow$, is defined inductively as follows:

1. if $s_1 -\!\!\circ\!\!\!\rightarrow s'_1, \ldots, s_m -\!\!\circ\!\!\!\rightarrow s'_m$ for some $m \geq 0$, then $x s_1 \ldots s_m -\!\!\circ\!\!\!\rightarrow x s'_1 \ldots s'_m$,
2. if $s_1 -\!\!\circ\!\!\!\rightarrow s'_1, \ldots, s_m -\!\!\circ\!\!\!\rightarrow s'_m$ for some $m \geq 0$, then $f s_1 \ldots s_m -\!\!\circ\!\!\!\rightarrow f s'_1 \ldots s'_m$,
3. if $s -\!\!\circ\!\!\!\rightarrow s'$, then $x.s -\!\!\circ\!\!\!\rightarrow x.s'$,
4. if $s_1 -\!\!\circ\!\!\!\rightarrow s'_1, \ldots, s_m -\!\!\circ\!\!\!\rightarrow s'_m$ for some $m \geq 0$, and $\mathrm{l} \to \mathrm{r}$ is a rewrite rule, then $(\mathrm{l} s_1 \ldots s_m)\downarrow_\beta -\!\!\circ\!\!\!\rightarrow (\mathrm{r} s'_1 \ldots s'_m)\downarrow_\beta$.

By convention $\mathrm{l} s_1 \ldots s_m$ and $\mathrm{r} s_1 \ldots s_m$ are in $\overline{\eta}$-normal form. It is easy to see that the relation $-\!\!\circ\!\!\!\rightarrow$ is reflexive. We make the following observations.

Remark 5.2.2. Let $\mathcal{H}$ be a left-linear higher-order rewriting system.

1. If $s = x s_1 \ldots s_m$ for some $m \geq 0$ and $s -\!\!\circ\!\!\!\rightarrow s'$, then $s -\!\!\circ\!\!\!\rightarrow s'$ is of the form $x s_1 \ldots s_m -\!\!\circ\!\!\!\rightarrow x s'_1 \ldots s'_m$ with $s_1 -\!\!\circ\!\!\!\rightarrow s'_1, \ldots, s_m -\!\!\circ\!\!\!\rightarrow s'_m$.
2. If $s = x.s_0$ and $s -\!\!\circ\!\!\!\rightarrow s'$, then $s -\!\!\circ\!\!\!\rightarrow s'$ is $x.s_0 -\!\!\circ\!\!\!\rightarrow x.s'_0$ with $s_0 -\!\!\circ\!\!\!\rightarrow s'_0$.

Remark 5.2.3. Let $\mathcal{H}$ be a left-linear higher-order rewriting system. Let s be a term. If $s = (\mathrm{l} s_1 \ldots s_m)\downarrow_\beta$ with $s_1 = x_1 \ldots . x_{n_1}.s_{10}, \ldots, s_m = x_1. \ldots . x_{n_m}.s_{m0}$, then s_{p0} is, up to a renaming of bound variables, a subterm of s for every $p \in \{1, \ldots, m\}$. This is a consequence of the restriction to pattern higher-order rewriting systems.

The previous remark does not hold if l is not a rule-pattern. Consider for instance $\mathrm{l} = x.f(xa)$, which is not a rule-pattern since $f(xa)$ is not a pattern, and the term $s_1 = y.y$. Then $(\mathrm{l} s_1)\downarrow_\beta = fa$ which does not contain y as a subterm.

We show that the set of terms is closed under the relation $-\!\!\circ\!\!\!\rightarrow$.

Proposition 5.2.4. *Let $\mathcal{H}$ be a left-linear higher-order rewriting system. If $s \in$ Terms and $s -\!\!\circ\!\!\!\rightarrow s'$, then $s' \in$ Terms.*

Proof. Suppose $s \in \mathsf{Terms}$ and $s -\!\!\circ\!\!\!\rightarrow s'$. We prove that $s' \in \mathsf{Terms}$. The proof proceeds by induction on the structure of s.

1. Suppose that $s \mathrel{\circ\!\!\!\!\longrightarrow} s'$ is $xs_1 \dots s_m \mathrel{\circ\!\!\!\!\longrightarrow} xs'_1 \dots s'_m$ with $s_1 \mathrel{\circ\!\!\!\!\longrightarrow} s'_1, \dots, s_m \mathrel{\circ\!\!\!\!\longrightarrow} s'_m$ for some $m \geq 0$. We have $s_1, \dots, s_m \in \mathsf{Terms}$. By the induction hypothesis, $s'_1, \dots, s'_m \in \mathsf{Terms}$. Hence $s' \in \mathsf{Terms}$.

2. Suppose $s \mathrel{\circ\!\!\!\!\longrightarrow} s'$ is $fs_1 \dots s_m \mathrel{\circ\!\!\!\!\longrightarrow} fs'_1 \dots s'_m$ with $s_1 \mathrel{\circ\!\!\!\!\longrightarrow} s'_1, \dots, s_m \mathrel{\circ\!\!\!\!\longrightarrow} s'_m$ for some $m \geq 0$. This case is similar to the previous one.

3. Suppose $s \mathrel{\circ\!\!\!\!\longrightarrow} s'$ is $x.s_0 \mathrel{\circ\!\!\!\!\longrightarrow} x.s'_0$ with $s_0 \mathrel{\circ\!\!\!\!\longrightarrow} s'_0$. We have $s_0 \in \mathsf{Terms}$, and hence we have by the induction hypothesis $s'_0 \in \mathsf{Terms}$. This yields that $s' \in \mathsf{Terms}$.

4. Finally, we suppose that $s \mathrel{\circ\!\!\!\!\longrightarrow} s'$ is $(\mathbf{l}s_0 \dots s_m){\downarrow_\beta} \mathrel{\circ\!\!\!\!\longrightarrow} (\mathbf{r}s'_0 \dots s'_m){\downarrow_\beta}$ with $s_0 \mathrel{\circ\!\!\!\!\longrightarrow} s'_0, \dots, s_m \mathrel{\circ\!\!\!\!\longrightarrow} s'_m$ for some $m \geq 0$. Then $s' \in \mathsf{Terms}$. □

We will show that the relation $\mathrel{\circ\!\!\!\!\longrightarrow}^*$ coincides with the relation $\twoheadrightarrow_{\mathcal{R}}$. To that end the following lemma is needed. Recall from Notation 1.2.5 of Chapter 1 that $|\sigma|$ denotes the length of the rewrite sequence σ. In the proof of the following lemma, we denote by $|s \twoheadrightarrow_{\mathcal{R}} s'|$ the length of the rewrite sequence $s \twoheadrightarrow_{\mathcal{R}} s'$.

Lemma 5.2.5. *Let $\mathcal{H}$ be a left-linear higher-order rewriting system. Let s be a closed linear pattern. Suppose $t_1 \twoheadrightarrow_{\mathcal{R}} t'_1, \dots, t_m \twoheadrightarrow_{\mathcal{R}} t'_m$. Then $(st_1 \dots t_m){\downarrow_\beta} \twoheadrightarrow_{\mathcal{R}} (st'_1 \dots t'_m){\downarrow_\beta}$.*

Proof. The proof proceeds by induction on $|t_1 \twoheadrightarrow_{\mathcal{R}} t'_1| + \dots + |t_m \twoheadrightarrow_{\mathcal{R}} t'_m|$.

Suppose $|t_1 \twoheadrightarrow_{\mathcal{R}} t'_1| + \dots + |t_m \twoheadrightarrow_{\mathcal{R}} t'_m| = 0$. Then the statement clearly holds.

Suppose $|t_1 \twoheadrightarrow_{\mathcal{R}} t'_1| + \dots + |t_m \twoheadrightarrow_{\mathcal{R}} t'_m| > 0$. Let $t_n \rightarrow_{\mathcal{R}} t''_n \twoheadrightarrow_{\mathcal{R}} t'_n$ for some $n \in \{1, \dots, m\}$. By the induction hypothesis, we have that $(st_1 \dots t''_n \dots t_m){\downarrow_\beta} \twoheadrightarrow_{\mathcal{R}} (st'_1 \dots t'_n \dots t'_m){\downarrow_\beta}$. Then, the proof is completed by showing that we have that $(st_1 \dots t_n \dots t_m){\downarrow_\beta} \twoheadrightarrow_{\mathcal{R}} (st_1 \dots t''_n \dots t_m){\downarrow_\beta}$. The rewrite step $t_n \rightarrow_{\mathcal{R}} t''_n$ can be decomposed into $t_n \;{}_{\beta\bar{\eta}}\!\twoheadleftarrow C_0[\mathbf{l}] \rightsquigarrow C_0[\mathbf{r}] \twoheadrightarrow_{\beta\bar{\eta}} t''_n$ for some context $C_0\Box$ and some rewrite rule $\mathbf{l} \rightarrow \mathbf{r}$. Then there is a precontext $C\Box$ such that $st_1 \dots t_n \dots t_m \;{}_{\beta\bar{\eta}}\!\twoheadleftarrow C[\mathbf{l}] \rightsquigarrow C[\mathbf{r}] \twoheadrightarrow_{\beta\bar{\eta}} st_1 \dots t''_n \dots t_m$. The proof is now suggested by the following diagram, where $C'\Box = C\Box{\downarrow_\beta}$:

$$\begin{tikzcd}
st_1 \dots t_n \dots t_m \arrow[dd, two heads, "\beta"'] & C[\mathbf{l}] \arrow[l, two heads, "\beta"] \arrow[d, two heads, "\beta"] & C[\mathbf{r}] \arrow[r, two heads, "\beta"] \arrow[d, two heads, "\beta"'] & st_1 \dots t''_n \dots t_m \arrow[dd, two heads, "\beta"] \\
 & C'[\mathbf{l}] \arrow[dl, two heads, "\beta"] & C'[\mathbf{r}] \arrow[dr, two heads, "\beta"'] & \\
(st_1 \dots t_n \dots t_m){\downarrow_\beta} & & & (st_1 \dots t''_n \dots t_m){\downarrow_\beta}
\end{tikzcd}$$

□

In the following theorem the first step of the proof of confluence using parallel reduction is established: we show that the relations $\twoheadrightarrow_{\mathcal{R}}$ and $\mathrel{\circ\!\!\!\!\longrightarrow}^*$ coincide. In fact, it is shown that $\rightarrow_{\mathcal{R}} \subseteq \mathrel{\circ\!\!\!\!\longrightarrow} \subseteq \twoheadrightarrow_{\mathcal{R}}$.

Theorem 5.2.6. *Let $\mathcal{H}$ be a left-linear higher-order rewriting system and let s be a term. Then $s \mathrel{-\!\!\ominus\!\!\!\rightarrow}^* s'$ if and only if $s \twoheadrightarrow_{\mathcal{R}} s'$.*

Proof. Suppose $(C\Box, \mathrm{l} \to \mathrm{r}) : s \to_{\mathcal{R}} s'$. Then we have $s = C[\mathrm{l}]\!\downarrow_\beta$ and $s' = C[\mathrm{r}]\!\downarrow_\beta$. We prove by induction on the structure of $C\Box$ that $s \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'$.

1. Suppose $C\Box = x_1. \ldots x_m.\Box t_1 \ldots t_n$ for some $m, n \geq 0$. By the definition of $\mathrel{-\!\!\ominus\!\!\!\rightarrow}$, we have that $(\mathrm{l}t_1 \ldots t_n)\!\downarrow_\beta \mathrel{-\!\!\ominus\!\!\!\rightarrow} (\mathrm{r}t_1 \ldots t_n)\!\downarrow_\beta$. Hence
$$\begin{array}{rcl} s & = & x_1. \ldots .x_m.(\mathrm{l}t_1 \ldots t_n)\!\downarrow_\beta \\ & \mathrel{-\!\!\ominus\!\!\!\rightarrow} & x_1. \ldots .x_m.(\mathrm{r}t_1 \ldots t_n)\!\downarrow_\beta \\ & = & s'. \end{array}$$

2. Suppose $C\Box = x_1. \ldots x_m.yt_1 \ldots t_{p-1}(C'\Box)t_{p+1} \ldots t_n$ for some $m, n \geq 0$. By the induction hypothesis, we have that $C'[\mathrm{l}]\!\downarrow_\beta \mathrel{-\!\!\ominus\!\!\!\rightarrow} C'[\mathrm{r}]\!\downarrow_\beta$. Hence
$$\begin{array}{rcl} s & = & x_1. \ldots .x_m.yt_1 \ldots t_{p-1}(C'[\mathrm{l}]\!\downarrow_\beta)t_{p+1} \ldots t_n \\ & \mathrel{-\!\!\ominus\!\!\!\rightarrow} & x_1. \ldots .x_m.yt_1 \ldots t_{p-1}(C'[\mathrm{r}]\!\downarrow_\beta)t_{p+1} \ldots t_n \\ & = & s'. \end{array}$$

3. Suppose $C\Box = x_1. \ldots x_m.ft_1 \ldots t_{p-1}(C'\Box)t_{p+1} \ldots t_n$ for some $m, n \geq 0$. This case is similar to the previous one.

Suppose $s \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'$. We prove by induction on the structure of s, using the observations of Remark 5.2.2 and Remark 5.2.3, that $s \twoheadrightarrow_{\mathcal{R}} s'$ if $s \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'$.

1. Suppose $s \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'$ is $xs_1 \ldots s_m \mathrel{-\!\!\ominus\!\!\!\rightarrow} xs'_1 \ldots s'_m$ with $s_1 \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'_1, \ldots, s_m \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'_m$ for some $m \geq 0$. By the induction hypothesis, we have that $s_1 \twoheadrightarrow_{\mathcal{R}} s'_1, \ldots, s_m \twoheadrightarrow_{\mathcal{R}} s'_m$. This yields $s = xs_1 \ldots s_m \twoheadrightarrow_{\mathcal{R}} xs'_1 \ldots s'_m = s'$.

2. Suppose $s \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'$ is $fs_1 \ldots s_m \mathrel{-\!\!\ominus\!\!\!\rightarrow} fs'_1 \ldots s'_m$ with $s_1 \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'_1, \ldots, s_m \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'_m$ for some $m \geq 0$. This case is similar to the previous one.

3. Suppose $s \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'$ is $x.s_0 \mathrel{-\!\!\ominus\!\!\!\rightarrow} x.s'_0$ with $s_0 \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'_0$. By the induction hypothesis, we have that $s_0 \twoheadrightarrow_{\mathcal{R}} s'_0$. Hence $s = x.s_0 \twoheadrightarrow_{\mathcal{R}} x.s'_0 = s'$.

4. Finally, we suppose that $s \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'$ is $(\mathrm{l}s_1 \ldots s_m)\!\downarrow_\beta \mathrel{-\!\!\ominus\!\!\!\rightarrow} (\mathrm{r}s'_1 \ldots s'_m)\!\downarrow_\beta$ with $s_1 \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'_1, \ldots, s_m \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'_m$ for some $m \geq 0$. By the induction hypothesis, we have that $s_1 \twoheadrightarrow_{\mathcal{R}} s'_1, \ldots, s_m \twoheadrightarrow_{\mathcal{R}} s'_m$. Moreover, by Lemma 5.2.5, we have $(\mathrm{l}s_1 \ldots s_m)\!\downarrow_\beta \twoheadrightarrow_{\mathcal{R}} (\mathrm{l}s'_1 \ldots s'_m)\!\downarrow_\beta$. Finally, since the rewriting system is left-linear, we have that $(\mathrm{l}s'_1 \ldots s'_m)\!\downarrow_\beta \to_{\mathcal{R}} (\mathrm{r}s'_1 \ldots s'_m)\!\downarrow_\beta$. We conclude that $s = (\mathrm{l}s_1 \ldots s_m)\!\downarrow_\beta \twoheadrightarrow_{\mathcal{R}} (\mathrm{r}s'_1 \ldots s'_m)\!\downarrow_\beta = s'$. □

Note that Proposition 5.2.4 is a corollary of Theorem 5.2.6. We preferred to give also a direct proof.

In the proof of the diamond property of the relation $\mathrel{-\!\!\circ\!\!\!\to}$, given in Section 5.3 for orthogonal systems and in Section 5.4 for weakly orthogonal systems, we need that the relation $\mathrel{-\!\!\circ\!\!\!\to}$ is closed under substitution. In the following lemma a stronger property is shown. It will be used only for the case that $s_0 = s'_0$ in the last case of the proof of Lemma 5.3.3, in the proof of Lemma 5.4.3 and in the proof of Theorem 5.4.6.

Lemma 5.2.7. *Let $\mathcal{H}$ be a left-linear higher-order rewriting system. Suppose that $s_0 \mathrel{-\!\!\circ\!\!\!\to} s'_0$ and $t_1 \mathrel{-\!\!\circ\!\!\!\to} t'_1, \ldots, t_m \mathrel{-\!\!\circ\!\!\!\to} t'_m$ for some $m \geq 0$. Then we have $(s_0 t_1 \ldots t_m)\!\downarrow_\beta \mathrel{-\!\!\circ\!\!\!\to} (s'_0 t'_1 \ldots t'_m)\!\downarrow_\beta$.*

Proof. Note that s_0 is by the convention of this section of the form $x_1. \ldots .x_m.s$. So $(s_0 t_1 \ldots t_m)\!\downarrow_\beta = (s[x_1 := t_1 \ldots x_m := t_m])\!\downarrow_\beta$. We prove that we have that $(s[x_1 := t_1 \ldots x_m := t_m])\!\downarrow_\beta \mathrel{-\!\!\circ\!\!\!\to} (s'[x_1 := t'_1 \ldots x_m := t'_m])\!\downarrow_\beta$. The proof proceeds by noetherian induction on the length of a rewrite sequence of $s[x_1 := t_1 \ldots x_m := t_m]$ to $\beta\overline{\eta}$-normal form.

I. Suppose that $s[x_1 := t_1 \ldots x_m := t_m]$ is in $\beta\overline{\eta}$-normal form. The proof of this part proceeds by induction on the structure of s, using the observations of Remark 5.2.2 and Remark 5.2.3.

1. Suppose that $s \mathrel{-\!\!\circ\!\!\!\to} s'$ is $y s_1 \ldots s_n \mathrel{-\!\!\circ\!\!\!\to} y s'_1 \ldots s'_n$ with $s_1 \mathrel{-\!\!\circ\!\!\!\to} s'_1, \ldots, s_n \mathrel{-\!\!\circ\!\!\!\to} s'_n$ for some $n \geq 0$. If $n = 0$, then the statement clearly holds, both if $x = y$ and if $x \neq y$. Otherwise, we have by the induction hypothesis that

   $$s_p[x_1 := t_1 \ldots x_m := t_m] \mathrel{-\!\!\circ\!\!\!\to} s'_p[x_1 := t'_1 \ldots x_m := t'_m].$$

   for every $p \in \{1, \ldots, n\}$. Since $s[x_1 := t_1 \ldots x_m := t_m]$ is in $\beta\overline{\eta}$-normal form, we have $y \notin \{x_1, \ldots, x_m\}$. We abbreviate $s_p[x_1 := t_1 \ldots x_m := t_m]$ by $\hat{s}_p$ and $s'_p[x_1 := t'_1 \ldots x_m := t'_m]$ by $\check{s}_p$ for every $p \in \{1, \ldots, n\}$. We have:

   $$\begin{array}{rl} s[x_1 := t_1 \ldots x_m := t_m] & = \\ y \hat{s}_1 \ldots \hat{s}_n & \mathrel{-\!\!\circ\!\!\!\to} \\ y \check{s}_1 \ldots \check{s}_n & = \\ s'[x_1 := t'_1 \ldots x_m := t'_m]. & \end{array}$$

2. Suppose that $s \mathrel{-\!\!\circ\!\!\!\to} s'$ is $f s_1 \ldots s_n \mathrel{-\!\!\circ\!\!\!\to} f s'_1 \ldots s'_n$ with $s_1 \mathrel{-\!\!\circ\!\!\!\to} s'_1, \ldots, s_n \mathrel{-\!\!\circ\!\!\!\to} s'_n$ for some $n \geq 0$. This case is similar to the previous one.

3. Suppose that $s \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'$ is $y.s_0 \mathrel{-\!\!\ominus\!\!\!\rightarrow} y.s_0'$ with $s_0 \mathrel{-\!\!\ominus\!\!\!\rightarrow} s_0'$. By the induction hypothesis we have:

$$s_0[x_1 := t_1 \ldots x_m := t_m] \mathrel{-\!\!\ominus\!\!\!\rightarrow} s_0'[x_1 := t_1' \ldots x_m := t_m'].$$

This yields that $s[x_1 := t_1 \ldots x_m := t_m] \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'[x_1 := t_1' \ldots x_m := t_m']$.

4. Suppose that we have that $s \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'$ is $(\mathtt{l}s_1 \ldots s_n){\downarrow_\beta} \mathrel{-\!\!\ominus\!\!\!\rightarrow} (\mathtt{r}s_1' \ldots s_n'){\downarrow_\beta}$ with $s_0 \mathrel{-\!\!\ominus\!\!\!\rightarrow} s_1', \ldots, s_n \mathrel{-\!\!\ominus\!\!\!\rightarrow} s_n'$ for some $n \geq 0$. If $n = 0$, then the statement clearly holds since $\mathtt{l}$ and $\mathtt{r}$ are closed. Otherwise, we have by the induction hypothesis that

$$s_p[x_1 := t_1 \ldots x_m := t_m] \mathrel{-\!\!\ominus\!\!\!\rightarrow} s_p'[x_1 := t_1' \ldots x_m := t_m']$$

for every $p \in \{1, \ldots, n\}$. The term $s_p[x_1 := t_1 \ldots x_m := t_m]$ is in β-normal form since it is a subterm of a term in β-normal form.

We abbreviate $s_p[x_1 := t_1 \ldots x_m := t_m]$ by $\hat{s}_p$ and $s_p'[x_1 := t_1' \ldots x_m := t_m']$ by $\check{s}_p$ for every $p \in \{1, \ldots, n\}$. Since $\mathtt{l}$ and $\mathtt{r}$ are closed, we have that $(\mathtt{l}s_1 \ldots s_n){\downarrow_\beta}[x_1 := t_1 \ldots x_m := t_m] = (\mathtt{l}\hat{s}_1 \ldots \hat{s}_n){\downarrow_\beta}$, and similarly for $\mathtt{r}$. Hence we have:

$$\begin{array}{rl} s[x_1 := t_1 \ldots x_m := t_m] & = \\ (\mathtt{l}s_1 \ldots s_n){\downarrow_\beta}[x_1 := t_1 \ldots x_m := t_m] & = \\ (\mathtt{l}\hat{s}_1 \ldots \hat{s}_n){\downarrow_\beta} & \mathrel{-\!\!\ominus\!\!\!\rightarrow} \\ (\mathtt{r}\check{s}_1 \ldots \check{s}_n){\downarrow_\beta} & = \\ (\mathtt{r}s_1' \ldots s_n'){\downarrow_\beta}[x_1 := t_1' \ldots x_m := t_m'] & = \\ s'[x_1 := t_1' \ldots x_m := t_m']. & \end{array}$$

II. Suppose $s[x_1 := t_1 \ldots x_m := t_m]$ is not in $\beta\overline{\eta}$-normal form. The proof of this part proceeds by induction on the structure of s.

1. Suppose $s \mathrel{-\!\!\ominus\!\!\!\rightarrow} s'$ is $ys_1 \ldots s_n \mathrel{-\!\!\ominus\!\!\!\rightarrow} ys_1' \ldots s_n'$ with $s_1 \mathrel{-\!\!\ominus\!\!\!\rightarrow} s_1', \ldots, s_n \mathrel{-\!\!\ominus\!\!\!\rightarrow} s_n'$ for some $n \geq 0$. Note that $m > 0$. By the induction hypothesis of the induction we have:

$$(s_p[x_1 := t_1 \ldots x_m := t_m]){\downarrow_\beta} \mathrel{-\!\!\ominus\!\!\!\rightarrow} (s_p'[x_1 := t_1' \ldots x_m := t_m']){\downarrow_\beta}$$

for every $p \in \{1, \ldots, n\}$. This is the case since $(s_p[x_1 := t_1 \ldots x_m := t_m]){\downarrow_\beta}$ is a subterm of $(s[x_1 := t_1 \ldots x_m := t_m]){\downarrow_\beta}$ for every $p \in \{1, \ldots, n\}$. We abbreviate $(s_p[x_1 := t_1 \ldots x_m := t_m]){\downarrow_\beta}$ by $\hat{s}_p$ and $(s_p'[x_1 := t_1' \ldots x_m := t_m']){\downarrow_\beta}$ by $\check{s}_p$. Now two cases are distinguished.

(a) Suppose that $y \notin \{x_1, \ldots, x_m\}$. Then we have:

$$\begin{aligned}
(s[x_1 := t_1 \ldots x_m := t_m])\downarrow_\beta & = \\
y\hat{s}_1 \ldots \hat{s}_n & \mathrel{-\!\!\ominus\!\!\rightarrow} \\
y\check{s}_1 \ldots \check{s}_n & = \\
(s'[x_1 := t'_1 \ldots x_m := t'_m])\downarrow_\beta . &
\end{aligned}$$

(b) Suppose that $y = x_q$ for some $q \in \{1, \ldots, m\}$. In that case, we have $t_q = y_1. \ldots .y_n.t_{q0}$ and $t'_q = y_1. \ldots .y_n.t'_{q0}$ with $t_{q0} \mathrel{-\!\!\ominus\!\!\rightarrow} t'_{q0}$. If $n = 0$, then the statement clearly holds. If $n > 0$, then we have:

$$\begin{aligned}
s[x_1 := t_1 \ldots x_m := t_m] & = \\
(t_q s_1 \ldots s_n)[x_1 := t_1 \ldots x_m := t_m] & \twoheadrightarrow_{\beta\overline{\eta}} \\
t_q \hat{s}_1 \ldots \hat{s_n} & \rightarrow^+_{\beta\overline{\eta}} \\
t_{q0}[y_1 := \hat{s_1} \ldots y_n := \hat{s_n}]. &
\end{aligned}$$

By the induction hypothesis we have that $\hat{s_p} \mathrel{-\!\!\ominus\!\!\rightarrow} \check{s_p}$ for every $p \in \{1, \ldots, n\}$. By the main induction hypothesis, we have that

$$\begin{aligned}
(t_{q0}[y_1 := \hat{s_1} \ldots y_n := \hat{s_n}])\downarrow_\beta & \mathrel{-\!\!\ominus\!\!\rightarrow} \\
(t'_{q0}[y_1 := \check{s_1} \ldots y_n := \check{s_n}])\downarrow_\beta . &
\end{aligned}$$

Hence

$$\begin{aligned}
(s[x_1 := t_1 \ldots x_m := t_m])\downarrow_\beta & = \\
(t_q \hat{s_1} \ldots \hat{s_n})\downarrow_\beta & = \\
(t_{q0}[y_1 := \hat{s_1} \ldots y_n := \hat{s_n}])\downarrow_\beta & \mathrel{-\!\!\ominus\!\!\rightarrow} \\
(t'_{q0}[y_1 := \check{s_1} \ldots y_n := \check{s_n}])\downarrow_\beta & = \\
(t'_q \check{s_1} \ldots \check{s_n})\downarrow_\beta & = \\
(s'[x_1 := t'_1 \ldots x_m := t'_m])\downarrow_\beta . &
\end{aligned}$$

2. Suppose that $s \mathrel{-\!\!\ominus\!\!\rightarrow} s'$ is $fs_1 \ldots s_n \mathrel{-\!\!\ominus\!\!\rightarrow} fs'_1 \ldots s'_n$ with $s_1 \mathrel{-\!\!\ominus\!\!\rightarrow} s'_1, \ldots, s_n \mathrel{-\!\!\ominus\!\!\rightarrow} s'_n$, for some $n \geq 0$. This case is entirely similar to the previous case with $y \notin \{x_1, \ldots, x_m\}$.

3. Suppose $s \mathrel{-\!\!\ominus\!\!\rightarrow} s'$ is $y.s_0 \mathrel{-\!\!\ominus\!\!\rightarrow} y.s'_0$ with $s_0 \mathrel{-\!\!\ominus\!\!\rightarrow} s'_0$. By the induction hypothesis we have:

$$(s_0[x_1 := t_1 \ldots x_m := t_m])\downarrow_\beta \mathrel{-\!\!\ominus\!\!\rightarrow} (s'_0[x_1 := t'_1 \ldots x_m := t'_m])\downarrow_\beta .$$

Hence we have $(s[x_1 := t_1 \ldots x_m := t_m])\downarrow_\beta \mathrel{-\!\!\ominus\!\!\rightarrow} (s'[x_1 := t'_1 \ldots x_m := t'_m])\downarrow_\beta$.

4. Suppose that $s \mathrel{-\!\!\circ\!\!\rightarrow} s'$ is of the form $(\mathtt{l}s_1 \ldots s_n)\!\downarrow_\beta \mathrel{-\!\!\circ\!\!\rightarrow} (\mathtt{r}s'_1 \ldots s'_n)\!\downarrow_\beta$ with $s_1 \mathrel{-\!\!\circ\!\!\rightarrow} s'_1, \ldots, s_n \mathrel{-\!\!\circ\!\!\rightarrow} s'_n$. If $n = 0$, then the statement holds since $\mathtt{l}$ and $\mathtt{r}$ are closed. Otherwise, we have by the induction hypothesis that

$$(s_p[x_1 := t_1 \ldots x_m := t_m])\!\downarrow_\beta \mathrel{-\!\!\circ\!\!\rightarrow} (s'_p[x_1 := t'_1 \ldots x_m := t'_m])\!\downarrow_\beta$$

for every $p \in \{1, \ldots, n\}$. We have, using the same abbreviations as above,

$$\begin{array}{rl}
(s[x_1 := t_1 \ldots x_m := t_m])\!\downarrow_\beta & = \\
((\mathtt{l}s_1 \ldots s_n)\!\downarrow_\beta[x_1 := t_1 \ldots x_m := t_m])\!\downarrow_\beta & = \\
(\mathtt{l}\hat{s}_1 \ldots \hat{s}_n)\!\downarrow_\beta & \mathrel{-\!\!\circ\!\!\rightarrow} \\
(\mathtt{r}\check{s}_1 \ldots \check{s}_n)\!\downarrow_\beta & = \\
((\mathtt{r}s'_1 \ldots s'_n)\!\downarrow_\beta[x_1 := t'_1 \ldots x_m := t'_m])\!\downarrow_\beta & = \\
(s'[x_1 := t'_1 \ldots x_m := t'_m])\!\downarrow_\beta. &
\end{array}$$

□

5.3 Orthogonal Systems

In this section we will consider pattern higher-order rewriting systems of the form $\mathcal{H} = (\lambda^{\rightarrow}_{\overline{\eta}}, \mathcal{A}, \mathcal{R})$ that are orthogonal, as defined in Definition 4.2.49 of Chapter 4.

In [Tak95], Takahashi gives a short and elegant proof of confluence for λ-calculus with β-reduction using parallel reduction. In this section, we show that orthogonal higher-order rewriting systems are confluent by adapting the proof by Takahashi. That is, we will show that the relation $\mathrel{-\!\!\circ\!\!\rightarrow}$ satisfies the diamond property by defining for every term s a term s^* such that $s' \mathrel{-\!\!\circ\!\!\rightarrow} s^*$ whenever $s \mathrel{-\!\!\circ\!\!\rightarrow} s'$. The term s^* is defined by induction on the structure of s, which is possible by Remark 5.2.3.

Definition 5.3.1. Let $\mathcal{H}$ be an orthogonal higher-order rewriting system. Let s be a term. We define s^* by induction on the structure of s as follows:

1. if $s = xs_1 \ldots s_m$ for some $m \geq 0$, then $s^* = xs^*_1 \ldots s^*_m$,

2. if $s = fs_1 \ldots s_m$ for some $m \geq 0$ and s is not of the form $(\mathtt{l}t_1 \ldots t_n)\!\downarrow_\beta$, then $s^* = fs^*_1 \ldots s^*_m$,

3. if $s = x.s_0$, then $s^* = x.s^*_0$,

4. if $s = (\mathtt{l}s_1 \ldots s_m)\!\downarrow_\beta$ with $s_1 = x_1.\ldots.x_{n_1}.s_{10}, \ldots, s_m = x_1.\ldots.x_{n_m}.s_{m0}$, then $s^* = (\mathtt{r}s^*_1 \ldots s^*_m)\!\downarrow_\beta$ with $s^*_1 = x_1.\ldots.x_{n_1}.s^*_{10}, \ldots, s_m = x_1.\ldots.x_{n_m}.s^*_{m0}$.

The link with a proof by developments is clearly present: the term s^* is obtained from s by performing a complete developments of all redex occurrences in s.

Lemma 5.3.2. *Let $\mathcal{H}$ be an orthogonal higher-order rewriting system. Let $s = fs_1 \dots s_m = (\mathtt{l}t_1 \dots t_n)\downarrow_\beta$ for some $m, n \geq 0$. Suppose that $s \mathrel{-\!\!\circ\!\!\!\rightarrow} s'$ with $fs_1 \dots s_m \mathrel{-\!\!\circ\!\!\!\rightarrow} fs'_1 \dots s'_m$ with $s_1 \mathrel{-\!\!\circ\!\!\!\rightarrow} s'_1, \dots, s_m \mathrel{-\!\!\circ\!\!\!\rightarrow} s'_m$. Then there exist terms $t'_1, \dots, t'_n$ such that*

1. $t_1 \mathrel{-\!\!\circ\!\!\!\rightarrow} t'_1, \dots, t_n \mathrel{-\!\!\circ\!\!\!\rightarrow} t'_n$,
2. $fs'_1 \dots s'_m = (\mathtt{l}t'_1 \dots t'_n)\downarrow_\beta$.

Proof. The left-hand side $\mathtt{l}$ is of the form $z_1. \dots .z_n.f\mathtt{l}_1 \dots \mathtt{l}_m$ with $\mathtt{l}_1, \dots, \mathtt{l}_m$ patterns. For every $p \in \{1, \dots, m\}$, we have that $(\mathtt{l}_p[z_1 := t_1 \dots z_n := t_n])\downarrow_\beta = s_p$.

Let $p \in \{1, \dots, n\}$. We have that z_p occurs exactly once in $f\mathtt{l}_1 \dots \mathtt{l}_m$, since the rewriting system is left-linear. Suppose $z_p : A$ with $\mathsf{ar}(A) = q_p$. There is a position $\phi \in \mathsf{Pos}(f\mathtt{l}_1 \dots \mathtt{l}_m)$ such that $f\mathtt{l}_1 \dots \mathtt{l}_m|_\phi = z_p y_1 \dots y_{q_p}$. We have that $t_p = y_1. \dots .y_{p_q}.t_{p0}$. Let $t'_{p0} = s'|_\phi$ and let $t'_p = y_1. \dots .y_{p_q}.t'_{p0}$.

In this way we obtain terms $t'_1, \dots, t'_n$ such that $t_1 \mathrel{-\!\!\circ\!\!\!\rightarrow} t'_1, \dots, t_n \mathrel{-\!\!\circ\!\!\!\rightarrow} t'_n$ and $(\mathtt{l}t'_1 \dots t'_n)\downarrow_\beta = s'$. □

Lemma 5.3.3. *Let $\mathcal{H}$ be an orthogonal higher-order rewriting system. If $s \mathrel{-\!\!\circ\!\!\!\rightarrow} s'$, then $s' \mathrel{-\!\!\circ\!\!\!\rightarrow} s^*$.*

Proof. The proof proceeds by induction on the structure of s.

1. Suppose $s \mathrel{-\!\!\circ\!\!\!\rightarrow} s'$ is $xs_1 \dots s_m \mathrel{-\!\!\circ\!\!\!\rightarrow} xs'_1 \dots s'_m$ with $s_1 \mathrel{-\!\!\circ\!\!\!\rightarrow} s'_1, \dots, s_m \mathrel{-\!\!\circ\!\!\!\rightarrow} s'_m$ for some $m \geq 0$. By the induction hypothesis, we have $s'_1 \mathrel{-\!\!\circ\!\!\!\rightarrow} s^*_1, \dots .s'_m \mathrel{-\!\!\circ\!\!\!\rightarrow} s^*_m$. Hence $s' = xs'_1 \dots s'_m \mathrel{-\!\!\circ\!\!\!\rightarrow} xs^*_1 \dots s^*_m = s^*$.

2. Suppose $s \mathrel{-\!\!\circ\!\!\!\rightarrow} s'$ is $fs_1 \dots s_m \mathrel{-\!\!\circ\!\!\!\rightarrow} fs'_1 \dots s'_m$ with $s_1 \mathrel{-\!\!\circ\!\!\!\rightarrow} s'_1, \dots, s_m \mathrel{-\!\!\circ\!\!\!\rightarrow} s'_m$ for some $m \geq 0$. Two cases are distinguished.

   (a) Suppose that s is not of the form $(\mathtt{l}t_1 \dots t_n)\downarrow_\beta$. Then $s^* = fs^*_1 \dots s^*_m$. By the induction hypothesis, we $s'_1 \mathrel{-\!\!\circ\!\!\!\rightarrow} s^*_1, \dots .s'_m \mathrel{-\!\!\circ\!\!\!\rightarrow} s^*_m$. This yields that $s' = fs'_1 \dots s'_m \mathrel{-\!\!\circ\!\!\!\rightarrow} fs^*_1 \dots s^*_m = s^*$.

   (b) Suppose that $s = (\mathtt{l}t_1 \dots t_n)\downarrow_\beta$. Then $s^* = (\mathtt{r}t^*_1 \dots t^*_n)\downarrow_\beta$. By Lemma 5.3.2, there exist terms $t'_1, \dots, t'_n$ such that $t_1 \mathrel{-\!\!\circ\!\!\!\rightarrow} t'_1, \dots, t_n \mathrel{-\!\!\circ\!\!\!\rightarrow} t'_n$ and $s' = (\mathtt{l}t'_1 \dots t'_n)\downarrow_\beta$. By the induction hypothesis, which can be applied by the observation made in Remark 5.2.3, we have $t_1 \mathrel{-\!\!\circ\!\!\!\rightarrow} t^*_1, \dots, t_n \mathrel{-\!\!\circ\!\!\!\rightarrow} t^*_n$. This yields that $s' = (\mathtt{l}t'_1 \dots t'_n)\downarrow_\beta \mathrel{-\!\!\circ\!\!\!\rightarrow} (\mathtt{r}t^*_1 \dots t^*_n)\downarrow_\beta = s^*$.

3. Suppose $s \mathrel{-\!\!\circ\!\!\!\rightarrow} s'$ is $x.s_0 \mathrel{-\!\!\circ\!\!\!\rightarrow} x.s'_0$. We have that $s^* = x.s^*_0$. By the induction hypothesis, we have that $s'_0 \mathrel{-\!\!\circ\!\!\!\rightarrow} s^*_0$. This yields that $s' = x.s'_0 \mathrel{-\!\!\circ\!\!\!\rightarrow} x.s^*_0 = s^*$.

4. Suppose that $s \mathrel{-\!\!\circ\!\!\rightarrow} s'$ is $(\mathrm{l}s_1 \ldots s_m){\downarrow_\beta} \mathrel{-\!\!\circ\!\!\rightarrow} (\mathrm{r}s'_1 \ldots s'_m){\downarrow_\beta}$. We have that $s^* = (\mathrm{r}s^*_1 \ldots s^*_m){\downarrow_\beta}$. By the induction hypothesis, we have $s'_1 \mathrel{-\!\!\circ\!\!\rightarrow} s^*_1, \ldots, s'_m \mathrel{-\!\!\circ\!\!\rightarrow} s^*_m$. By Lemma 5.2.7, this yields $s' = (\mathrm{r}s'_1 \ldots s'_m){\downarrow_\beta} \mathrel{-\!\!\circ\!\!\rightarrow} (\mathrm{r}s^*_1 \ldots s^*_m){\downarrow_\beta} = s^*$. □

Theorem 5.3.4. *Orthogonal higher-order rewriting systems are confluent.*

Proof. Let s be a term in an orthogonal higher-order rewriting system and suppose that $s \mathrel{-\!\!\circ\!\!\rightarrow} s'$ and $s \mathrel{-\!\!\circ\!\!\rightarrow} s''$. By Lemma 5.3.3, we have that $s' \mathrel{-\!\!\circ\!\!\rightarrow} s^*$ and $s'' \mathrel{-\!\!\circ\!\!\rightarrow} s^*$. Then, Theorem 5.2.6 in Section 5.2 and Lemma 1.1.25 in Chapter 1 yield confluence. □

5.4 Weakly Orthogonal Systems

In this section we consider higher-order rewriting systems that are weakly orthogonal, as defined in Definition 4.2.49 of Chapter 4.

In this section we show that weakly orthogonal higher-order rewriting systems are confluent. First we define the set of redex occurrences that are contracted in a parallel reduction step $s \mathrel{-\!\!\circ\!\!\rightarrow} s'$.

Definition 5.4.1. The set of *redex occurrences that are contracted in* $s \mathrel{-\!\!\circ\!\!\rightarrow} s'$, denoted by $\mathsf{c}(s \mathrel{-\!\!\circ\!\!\rightarrow} s')$, is defined by induction on the definition of $\mathrel{-\!\!\circ\!\!\rightarrow}$.

1. Suppose $s \mathrel{-\!\!\circ\!\!\rightarrow} s'$ is $xs_1 \ldots s_m \mathrel{-\!\!\circ\!\!\rightarrow} xs'_1 \ldots s'_m$ with $s_1 \mathrel{-\!\!\circ\!\!\rightarrow} s'_1, \ldots, s_m \mathrel{-\!\!\circ\!\!\rightarrow} s'_m$ for some $m \geq 0$. If $m = 0$ then $\mathsf{c}(s \mathrel{-\!\!\circ\!\!\rightarrow} s') = \emptyset$, otherwise $\mathsf{c}(s \mathrel{-\!\!\circ\!\!\rightarrow} s')$ consists of all redex occurrences $(C\Box, \mathrm{l} \rightarrow \mathrm{r})$ such that

$$C\Box = xs_1 \ldots s_{p-1}(C'\Box)s_{p+1} \ldots s_m$$

   and

$$(C'\Box, \mathrm{l} \rightarrow \mathrm{r}) \in \mathsf{c}(s_p \mathrel{-\!\!\circ\!\!\rightarrow} s'_p)$$

   for some $p \in \{1, \ldots, m\}$.

2. Suppose that $s \mathrel{-\!\!\circ\!\!\rightarrow} s'$ is $fs_1 \ldots s_m \mathrel{-\!\!\circ\!\!\rightarrow} fs'_1 \ldots s'_m$ with $s_1 \mathrel{-\!\!\circ\!\!\rightarrow} s'_1, \ldots, s_m \mathrel{-\!\!\circ\!\!\rightarrow} s'_m$ for some $m \geq 0$. If $m = 0$ then $\mathsf{c}(s \mathrel{-\!\!\circ\!\!\rightarrow} s') = \emptyset$, otherwise $\mathsf{c}(s \mathrel{-\!\!\circ\!\!\rightarrow} s')$ consists of all redex occurrences $(C\Box, \mathrm{l} \rightarrow \mathrm{r})$ such that

$$C\Box = fs_1 \ldots s_{p-1}(C'\Box)s_{p+1} \ldots s_m$$

   and

$$(C'\Box, \mathrm{l} \rightarrow \mathrm{r}) \in \mathsf{c}(s_p \mathrel{-\!\!\circ\!\!\rightarrow} s'_p)$$

   for some $p \in \{1, \ldots, m\}$.

3. Suppose $s \mathrel{-\!\!\circ\!\!\to} s'$ is $x.s_0 \mathrel{-\!\!\circ\!\!\to} x.s_0'$ with $s_0 \mathrel{-\!\!\circ\!\!\to} s_0'$. Then $\mathsf{c}(s \mathrel{-\!\!\circ\!\!\to} s')$ consists of all redex occurrences $(C\Box, \mathsf{l} \to \mathsf{r})$ such that
$$C\Box = x.C'\Box$$
and
$$(C'\Box, \mathsf{l} \to \mathsf{r}) \in \mathsf{c}(s_0 \mathrel{-\!\!\circ\!\!\to} s_0').$$

4. Finally, suppose that $s \mathrel{-\!\!\circ\!\!\to} s'$ is of the form $(\mathsf{g}s_1 \ldots s_m){\downarrow_\beta} \mathrel{-\!\!\circ\!\!\to} (\mathsf{d}s_1' \ldots s_m'){\downarrow_\beta}$ with $s_1 \mathrel{-\!\!\circ\!\!\to} s_1', \ldots, s_m \mathrel{-\!\!\circ\!\!\to} s_m'$ for some $m \geq 0$. Then $\mathsf{c}(s \mathrel{-\!\!\circ\!\!\to} s')$ consists of
$$(\Box s_1 \ldots s_m, \mathsf{g} \to \mathsf{d})$$
and moreover, if $m > 0$, of all redex occurrences $(C\Box, \mathsf{l} \to \mathsf{r})$ such that
$$C\Box = (\mathsf{g}s_1 \ldots s_{p-1}(C'\Box)s_{p+1} \ldots s_m){\downarrow_\beta}$$
and
$$(C'\Box, \mathsf{l} \to \mathsf{r}) \in \mathsf{c}(s_p \mathrel{-\!\!\circ\!\!\to} s_p')$$
for some $p \in \{1, \ldots, m\}$.

Note that since the higher-order rewriting system is by assumption left-linear, in the last clause of the previous definition $(\mathsf{g}s_1 \ldots s_{p-1}C\Box s_{p+1} \ldots s_m){\downarrow_\beta}$ contains indeed exactly one occurrence of the distinguished variable $\Box$.

We will show in Lemma 5.4.3 that if $s \mathrel{-\!\!\circ\!\!\to} s'$, and the redex occurrence $(C\Box, \mathsf{l} \to \mathsf{r})$ is contracted in the parallel reduction step $s \mathrel{-\!\!\circ\!\!\to} s'$, then the result of contracting the redex occurrence $(C\Box, \mathsf{l} \to \mathsf{r})$ can perform a parallel reduction step to s'. This is an essential ingredient of the proof of the key result of this section, presented as Theorem 5.4.6. In the proof of Lemma 5.4.3 we make use of the following observation.

Remark 5.4.2. Suppose $s \mathrel{-\!\!\circ\!\!\to} s'$.

1. If $s = xs_1 \ldots s_m$ then $s' = xs_1' \ldots s_m'$ with $s_1 \mathrel{-\!\!\circ\!\!\to} s_1', \ldots, s_m \mathrel{-\!\!\circ\!\!\to} s_m'$ (see Remark 5.2.2). Moreover, if $(C\Box, \mathsf{l} \to \mathsf{r}) \in \mathsf{c}(s \mathrel{-\!\!\circ\!\!\to} s')$, then $m > 0$ and $C\Box = xs_1 \ldots s_{p-1}(C'\Box)s_{p+1} \ldots s_m$ with $(C'\Box, \mathsf{l} \to \mathsf{r}) \in \mathsf{c}(s_p \mathrel{-\!\!\circ\!\!\to} s_p')$ for some $p \in \{1, \ldots, m\}$.

2. If $s = x.s_0$, then $s' = x.s_0'$ with $s_0 \mathrel{-\!\!\circ\!\!\to} s_0'$ (see Remark 5.2.2). Moreover, if $(C\Box, \mathsf{l} \to \mathsf{r}) \in \mathsf{c}(s \mathrel{-\!\!\circ\!\!\to} s')$, then $C\Box = x.C'\Box$ and $(C'\Box, \mathsf{l} \to \mathsf{r}) \in \mathsf{c}(s_0 \mathrel{-\!\!\circ\!\!\to} s_0')$.

Lemma 5.4.3. *Let $\mathcal{H}$ be a left-linear higher-order rewriting system. Let $s \mathrel{-\!\!\circ\!\!\to} s'$ and $(C\Box, \mathsf{l} \to \mathsf{r}) \in \mathsf{c}(s \mathrel{-\!\!\circ\!\!\to} s')$. Then $C[\mathsf{r}]{\downarrow_\beta} \mathrel{-\!\!\circ\!\!\to} s'$.*

Proof. The proof proceeds by induction on the derivation of $s \mathrel{\circ\!\!\!\!\longrightarrow} s'$.

1. Suppose $s \mathrel{\circ\!\!\!\!\longrightarrow} s'$ is $xs_1 \ldots s_m \mathrel{\circ\!\!\!\!\longrightarrow} xs'_1 \ldots s'_m$ with $s_1 \mathrel{\circ\!\!\!\!\longrightarrow} s'_1, \ldots, s_m \mathrel{\circ\!\!\!\!\longrightarrow} s'_m$ for some $m \geq 0$. If $(C\Box, \mathsf{l} \rightarrow \mathsf{r}) \in \mathsf{c}(s \mathrel{\circ\!\!\!\!\longrightarrow} s')$, then $m > 0$ and we have that $C\Box = xs_1 \ldots s_{p-1}(C'\Box)s_{p+1} \ldots s_m$ with $(C'\Box, \mathsf{l} \rightarrow \mathsf{r}) \in \mathsf{c}(s_p \mathrel{\circ\!\!\!\!\longrightarrow} s'_p)$ for some $p \in \{1, \ldots, m\}$. By the induction hypothesis, we have $C'[\mathsf{r}]\!\downarrow_\beta \mathrel{\circ\!\!\!\!\longrightarrow} s'_p$. Hence we have
$$\begin{array}{rl} C[\mathsf{r}]\!\downarrow_\beta & = \\ xs_1 \ldots s_{p-1}(C'[\mathsf{r}]\!\downarrow_\beta)s_{p+1} \ldots s_m & \mathrel{\circ\!\!\!\!\longrightarrow} \\ xs'_1 \ldots s'_{p-1}(C'[\mathsf{r}]\!\downarrow_\beta)s'_{p+1} \ldots s'_m & = \\ s'. & \end{array}$$

2. Suppose that $s \mathrel{\circ\!\!\!\!\longrightarrow} s'$ is $fs_1 \ldots s_m \mathrel{\circ\!\!\!\!\longrightarrow} fs'_1 \ldots s'_m$ with $s_1 \mathrel{\circ\!\!\!\!\longrightarrow} s'_1, \ldots, s_m \mathrel{\circ\!\!\!\!\longrightarrow} s'_m$ for some $m \geq 0$. This case is similar to the previous one.

3. Suppose $s \mathrel{\circ\!\!\!\!\longrightarrow} s'$ is $x.s_0 \mathrel{\circ\!\!\!\!\longrightarrow} x.s'_0$ with $s_0 \mathrel{\circ\!\!\!\!\longrightarrow} s'_0$. If $(C\Box, \mathsf{l} \rightarrow \mathsf{r}) \in \mathsf{c}(s \mathrel{\circ\!\!\!\!\longrightarrow} s')$, then $C\Box = x.C'\Box$ with $(C'\Box, \mathsf{l} \rightarrow \mathsf{r}) \in \mathsf{c}(s \mathrel{\circ\!\!\!\!\longrightarrow} s')$. By the induction hypothesis, we have that $C'[\mathsf{r}]\!\downarrow_\beta \mathrel{\circ\!\!\!\!\longrightarrow} s'_0$. Hence we have
$$\begin{array}{rl} C[\mathsf{r}]\!\downarrow_\beta & = \\ x.C'[\mathsf{r}]\!\downarrow_\beta & \mathrel{\circ\!\!\!\!\longrightarrow} \\ x.s'_0 & = \\ s'. & \end{array}$$

4. Suppose that $s \mathrel{\circ\!\!\!\!\longrightarrow} s'$ is of the form $(\mathsf{g}s_1 \ldots s_m)\!\downarrow_\beta \mathrel{\circ\!\!\!\!\longrightarrow} (\mathsf{d}s'_1 \ldots s'_m)\!\downarrow_\beta$, with $s_1 \mathrel{\circ\!\!\!\!\longrightarrow} s'_1, \ldots, s_m \mathrel{\circ\!\!\!\!\longrightarrow} s'_m$ for some $m \geq 0$. Two cases are distinguished.

   (a) $C\Box = \Box s_1 \ldots s_m$ and $\mathsf{l} \rightarrow \mathsf{r} = \mathsf{g} \rightarrow \mathsf{d}$. In that case, we have that $(\mathsf{d}s_1 \ldots s_m)\!\downarrow_\beta \mathrel{\circ\!\!\!\!\longrightarrow} (\mathsf{d}s'_1 \ldots s'_m)\!\downarrow_\beta$ by Lemma 5.2.7. Hence we have that $C[\mathsf{r}]\!\downarrow_\beta \mathrel{\circ\!\!\!\!\longrightarrow} s'$.

   (b) Otherwise, we have $m > 0$ and $C\Box = (\mathsf{g}s_1 \ldots s_{p-1}(C'\Box)s_{p+1} \ldots s_m)\!\downarrow_\beta$ with $(C'\Box, \mathsf{l} \rightarrow \mathsf{r}) \in \mathsf{c}(s_p \mathrel{\circ\!\!\!\!\longrightarrow} s'_p)$ for some $p \in \{1, \ldots, m\}$. By the induction hypothesis, we have that $C'[\mathsf{r}] \mathrel{\circ\!\!\!\!\longrightarrow} s'_p$. Hence we have
$$\begin{array}{rl} C[\mathsf{r}]\!\downarrow_\beta & = \\ (\mathsf{l}s_1 \ldots s_{p-1}(C'[\mathsf{r}]\!\downarrow_\beta)s_{p+1} \ldots s_m)\!\downarrow_\beta & \mathrel{\circ\!\!\!\!\longrightarrow} \\ (\mathsf{d}s'_1 \ldots s'_{p-1}s'_p s'_{p+1} \ldots s_m)\!\downarrow_\beta & = \\ s'. & \end{array}$$

□

In Theorem 5.4.6 it is shown that the relation $\xrightarrow{\circ}$ satisfies the diamond property. This is more difficult as there are more redex occurrences that are overlapping, that are contracted in a divergence $t' \xleftarrow{\circ} s \xrightarrow{\circ} t''$. The proof of Theorem 5.4.6 proceeds by induction on a measure expressing a degree of difficulty which is defined as follows.

Definition 5.4.4. Let $s \xrightarrow{\circ} t$ and $s \xrightarrow{\circ} t'$. The *complexity of the divergence* $t \xleftarrow{\circ} s \xrightarrow{\circ} t'$, denoted by $\mathsf{compl}(t \xleftarrow{\circ} s \xrightarrow{\circ} t')$, is defined as the cardinality of the set consisting of pairs

$$((C\Box, \mathrm{l} \to \mathrm{r}), (C'\Box, \mathrm{l}' \to \mathrm{r}'))$$

with

1. $(C\Box, \mathrm{l} \to \mathrm{r}) \in \mathsf{c}(s \xrightarrow{\circ} t)$,
2. $(C'\Box, \mathrm{l}' \to \mathrm{r}') \in \mathsf{c}(s \xrightarrow{\circ} t')$,
3. $(C\Box, \mathrm{l} \to \mathrm{r})$ and $(C'\Box, \mathrm{l}' \to \mathrm{r}')$ are overlapping redex occurrences in s.

In [Oos96], van Oostrom gives a condition on critical pairs that implies confluence of higher-order rewriting systems. He shows that if for every critical pair (s, t) we have that $s \xrightarrow{\circ} t$, that is, s can be rewritten to t by performing a complete development, then the higher-order rewriting system is confluent. The proof of this result makes use of a measure that is a refinement of the notion of complexity of a divergence defined above.

Lemma 5.4.5. *Let $\mathcal{H}$ be a weakly orthogonal higher-order rewriting system. Let $s = fs_1 \ldots s_m = (\mathrm{l}t_1 \ldots t_n)\downarrow_\beta$ for some $m, n \geq 0$. Suppose that $s \xrightarrow{\circ} s'$ is $fs_1 \ldots s_m \xrightarrow{\circ} fs'_1 \ldots s'_m$ with $s_1 \xrightarrow{\circ} s'_1, \ldots, s_m \xrightarrow{\circ} s'_m$, and that $s \xrightarrow{\circ} s''$ is $(\mathrm{l}t_1 \ldots t_n)\downarrow_\beta \xrightarrow{\circ} (\mathrm{r}t_1 \ldots t_n)\downarrow_\beta$ with $\mathsf{compl}(s' \xleftarrow{\circ} s \xrightarrow{\circ} s'') = 0$. Then, if $n > 0$, there exist terms $t'_1, \ldots, t'_n$ such that*

1. $t_1 \xrightarrow{\circ} t'_1, \ldots, t_n \xrightarrow{\circ} t'_n$,
2. $s' = (\mathrm{l}t'_1 \ldots t'_n)\downarrow_\beta$.

Proof. As the proof of Lemma 5.3.2 in Section 5.3. □

Theorem 5.4.6. *Let $\mathcal{H}$ be a weakly orthogonal higher-order rewriting system. Then the relation $\xrightarrow{\circ}$ satisfies the diamond property.*

Proof. Suppose $s \xrightarrow{\circ} s'$ and $s \xrightarrow{\circ} s''$. We prove that a term t exists such that $s' \xrightarrow{\circ} t$ and $s'' \xrightarrow{\circ} t$. The proof proceeds by induction on $\mathsf{compl}(s' \xleftarrow{\circ} s \xrightarrow{\circ} s'')$.

I. Suppose $\mathsf{compl}(s' \mathrel{\leftarrow\!\!\circ\!\!-} s \mathrel{-\!\!\circ\!\!\to} s'') = 0$. The proof of this part proceeds by induction on the structure of s.

1. Suppose $s \mathrel{-\!\!\circ\!\!\to} s'$ is $xs_1 \dots s_m \mathrel{-\!\!\circ\!\!\to} xs'_1 \dots s'_m$ with $s_1 \mathrel{-\!\!\circ\!\!\to} s'_1, \dots, s_m \mathrel{-\!\!\circ\!\!\to} s'_m$ for some $m \geq 0$. Then $s \mathrel{-\!\!\circ\!\!\to} s''$ is of the form $xs_1 \dots s_m \mathrel{-\!\!\circ\!\!\to} xs''_1 \dots s''_m$ with $s_1 \mathrel{-\!\!\circ\!\!\to} s''_1, \dots, s_m \mathrel{-\!\!\circ\!\!\to} s''_m$. By the induction hypothesis, if $m > 0$ then terms $t_1, \dots, t_m$ exist such that $s'_p \mathrel{-\!\!\circ\!\!\to} t_p$ and $s''_p \mathrel{-\!\!\circ\!\!\to} t_p$ for every $p \in \{1, \dots, m\}$. Take $t = xt_1 \dots t_m$. Then we have both $s' = xs'_1 \dots s'_m \mathrel{-\!\!\circ\!\!\to} xt_1 \dots t_m = t$ and $s'' = xs''_1 \dots s''_m \mathrel{-\!\!\circ\!\!\to} xt_1 \dots t_m = t$.

2. Suppose $s \mathrel{-\!\!\circ\!\!\to} s'$ is $fs_1 \dots s_m \mathrel{-\!\!\circ\!\!\to} fs'_1 \dots s'_m$ with $s_1 \mathrel{-\!\!\circ\!\!\to} s'_1, \dots, s_m \mathrel{-\!\!\circ\!\!\to} s'_m$ for some $m \geq 0$. Two cases are distinguished.

   (a) Suppose $s \mathrel{-\!\!\circ\!\!\to} s''$ is $fs_1 \dots s_m \mathrel{-\!\!\circ\!\!\to} fs''_1 \dots s''_m$ with $s_1 \mathrel{-\!\!\circ\!\!\to} s''_1, \dots, s_m \mathrel{-\!\!\circ\!\!\to} s''_m$. This case is similar to the previous case.

   (b) Suppose that $s \mathrel{-\!\!\circ\!\!\to} s''$ is of the form $(\mathrm{l}t_1 \dots t_n){\downarrow_\beta} \mathrel{-\!\!\circ\!\!\to} (\mathrm{r}t''_1 \dots t''_n){\downarrow_\beta}$ with $t_1 \mathrel{-\!\!\circ\!\!\to} t''_1, \dots, t_n \mathrel{-\!\!\circ\!\!\to} t''_n$ for some $n \geq 0$. By Lemma 5.4.5, if $n > 0$ then there exist terms $t'_1, \dots, t'_n$ such that $t_1 \mathrel{-\!\!\circ\!\!\to} t'_1, \dots, t_n \mathrel{-\!\!\circ\!\!\to} t'_n$ and $s' = (\mathrm{l}t'_1 \dots t'_n){\downarrow_\beta}$. By the induction hypothesis, there exist terms $t_1, \dots, t_n$ such that $t'_p \mathrel{-\!\!\circ\!\!\to} t_p$ and $t''_p \mathrel{-\!\!\circ\!\!\to} t_p$ for every $p \in \{1, \dots, n\}$. Take $t = (\mathrm{r}t_1 \dots t_n){\downarrow_\beta}$. Then we have $s' = (\mathrm{l}t'_1 \dots t'_n){\downarrow_\beta} \mathrel{-\!\!\circ\!\!\to} (\mathrm{r}t_1 \dots t_n){\downarrow_\beta} = t$ and, by Lemma 5.2.7, $s'' = (\mathrm{r}t''_1 \dots t''_n){\downarrow_\beta} \mathrel{-\!\!\circ\!\!\to} (\mathrm{r}t_1 \dots t_n){\downarrow_\beta} = t$.

3. Suppose $s \mathrel{-\!\!\circ\!\!\to} s'$ is $x.s_0 \mathrel{-\!\!\circ\!\!\to} x.s'_0$ with $s_0 \mathrel{-\!\!\circ\!\!\to} s'_0$. Then $s \mathrel{-\!\!\circ\!\!\to} s''$ is $x.s_0 \mathrel{-\!\!\circ\!\!\to} x.s''_0$ with $s_0 \mathrel{-\!\!\circ\!\!\to} s''_0$. By the induction hypothesis, there exists a term t_0 such that $s'_0 \mathrel{-\!\!\circ\!\!\to} t_0$ and $s''_0 \mathrel{-\!\!\circ\!\!\to} t_0$. Take $t = x.t_0$. Then we have both $s' = x.s'_0 \mathrel{-\!\!\circ\!\!\to} x.t_0 = t$ and $s'' = x.s''_0 \mathrel{-\!\!\circ\!\!\to} x.t_0 = t$.

4. Suppose that $s \mathrel{-\!\!\circ\!\!\to} s'$ is of the form $(\mathrm{l}s_1 \dots s_m){\downarrow_\beta} \mathrel{-\!\!\circ\!\!\to} (\mathrm{r}s'_1 \dots s'_m){\downarrow_\beta}$ with $s_1 \mathrel{-\!\!\circ\!\!\to} s'_1, \dots, s_m \mathrel{-\!\!\circ\!\!\to} s'_m$ for some $m \geq 0$. Two cases are distinguished.

   (a) Suppose $s \mathrel{-\!\!\circ\!\!\to} s''$ is $ft_1 \dots t_n \mathrel{-\!\!\circ\!\!\to} ft''_1 \dots t''_n$ with $t_1 \mathrel{-\!\!\circ\!\!\to} t''_1, \dots, t_n \mathrel{-\!\!\circ\!\!\to} t''_n$ for some $n \geq 0$. This is case 2(b) considered above.

   (b) Suppose $s \mathrel{-\!\!\circ\!\!\to} s''$ is $(\mathrm{g}\tilde{s}_1 \dots \tilde{s}_n){\downarrow_\beta} \mathrel{-\!\!\circ\!\!\to} (\mathrm{d}s''_1 \dots s''_n){\downarrow_\beta}$. Since we are in the case that $\mathsf{compl}(s' \mathrel{\leftarrow\!\!\circ\!\!-} s \mathrel{-\!\!\circ\!\!\to} s') = 0$, we have that $\mathrm{l} \to \mathrm{r} = \mathrm{g} \to \mathrm{d}$, $m = n$ and $s_1 = \tilde{s}_1, \dots, s_m = \tilde{s}_m$. By the induction hypothesis, if $m > 0$ then terms $t_1, \dots, t_m$ exist such that $s'_p \mathrel{-\!\!\circ\!\!\to} t_p$ and $s''_p \mathrel{-\!\!\circ\!\!\to} t_p$ for every $p \in \{1, \dots, m\}$. Take $t = (\mathrm{r}t_1 \dots t_m){\downarrow_\beta}$. Then we have $s' = (\mathrm{r}s'_1 \dots s'_m){\downarrow_\beta} \mathrel{-\!\!\circ\!\!\to} (\mathrm{r}t_1 \dots t_m){\downarrow_\beta} = t$ and moreover $s'' = (\mathrm{r}s''_1 \dots s''_m){\downarrow_\beta} \mathrel{-\!\!\circ\!\!\to} (\mathrm{r}t_1 \dots t_m){\downarrow_\beta} = t$, both by Lemma 5.2.7.

II. Suppose $\mathsf{compl}(s' \mathrel{\leftarrow\!\!\circ\!\!-} s \mathrel{-\!\!\circ\!\!\rightarrow} s'') > 0$. Then the set consisting of all pairs

$$((C\Box, \mathtt{l} \to \mathtt{r}), (C'\Box, \mathtt{g} \to \mathtt{d}))$$

with

1. $(C\Box, \mathtt{l} \to \mathtt{r}) \in \mathsf{c}(s \mathrel{-\!\!\circ\!\!\rightarrow} s')$,
2. $(C'\Box, \mathtt{g} \to \mathtt{d}) \in \mathsf{c}(s \mathrel{-\!\!\circ\!\!\rightarrow} s'')$,
3. $(C\Box, \mathtt{l} \to \mathtt{r})$ and $(C'\Box, \mathtt{g} \to \mathtt{d})$ are overlapping redex occurrences in s,

is non-empty. There are two overlapping redex occurrences in the above set that are minimal in the sense that they do not contain other redex occurrences that contribute to $\mathsf{compl}(s' \mathrel{\leftarrow\!\!\circ\!\!-} s \mathrel{-\!\!\circ\!\!\rightarrow} s'')$. Let $(C\Box, \mathtt{l} \to \mathtt{r})$ and $(C'\Box, \mathtt{g} \to \mathtt{d})$ be such a minimal pair of overlapping redex occurrences with $(C\Box, \mathtt{l} \to \mathtt{r}) \in \mathsf{c}(s \mathrel{-\!\!\circ\!\!\rightarrow} s')$ and $(C'\Box, \mathtt{g} \to \mathtt{d}) \in \mathsf{c}(s \mathrel{-\!\!\circ\!\!\rightarrow} s'')$.

Since the higher-order rewriting system is weakly orthogonal, we have that $C[\mathtt{r}]\downarrow_\beta = C'[\mathtt{d}]\downarrow_\beta$. Call this term s_0. By Lemma 5.4.3, we have that $C[\mathtt{r}]\downarrow_\beta \mathrel{-\!\!\circ\!\!\rightarrow} s'$ and $C'[\mathtt{d}]\downarrow_\beta \mathrel{-\!\!\circ\!\!\rightarrow} s''$. Since we have chosen $(C\Box, \mathtt{l} \to \mathtt{r})$ and $(C'\Box, \mathtt{g} \to \mathtt{d})$ to be a minimal pair of overlapping redex occurrences, we have $\mathsf{compl}(s' \mathrel{\leftarrow\!\!\circ\!\!-} s_0 \mathrel{-\!\!\circ\!\!\rightarrow} s'') < \mathsf{compl}(s' \mathrel{\leftarrow\!\!\circ\!\!-} s \mathrel{-\!\!\circ\!\!\rightarrow} s'')$. Hence by the induction hypothesis applied to $s_0 \mathrel{-\!\!\circ\!\!\rightarrow} s'$ and $s_0 \mathrel{-\!\!\circ\!\!\rightarrow} s''$, a term t exists such that $s' \mathrel{-\!\!\circ\!\!\rightarrow} t$ and $s'' \mathrel{-\!\!\circ\!\!\rightarrow} t$. $\Box$

Theorem 5.4.7. *Weakly orthogonal higher-order rewriting systems are confluent.*

Proof. By Theorem 5.4.6, Lemma 5.2.7 in Section 5.2 and Lemma 1.1.25 in Chapter 1. $\Box$

Chapter 6

Normalisation in Higher-Order Rewriting

This chapter is concerned with pattern higher-order rewriting systems with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus. We will say simply 'higher-order rewriting system' instead of 'pattern higher-order rewriting system with $\lambda_{\overline{\eta}}^{\rightarrow}$ as substitution calculus'.

If a term can be rewritten to a normal form but admits an infinite rewrite sequence as well, then it is important to know how to rewrite this term in order to reach a normal form. For λ-calculus with β-reduction, there is a classical result stating that the leftmost rewrite strategy is normalising. This result is mentioned in Corollary 2.1.19 of Chapter 2. For first-order term rewriting, O'Donnell introduces in [O'D77] the notion of outermost-fair rewrite sequence. A rewrite sequence is said to be outermost-fair if every outermost redex occurrence is eventually eliminated. O'Donnell shows that outermost-fair rewriting is normalising for almost orthogonal first-order term rewriting systems.

In this chapter we will show that outermost-fair rewriting is normalising for higher-order rewriting systems that are almost orthogonal and fully extended as defined in Definition 4.2.49 and Definition 4.2.51 of Chapter 4. This extends the result by O'Donnell to the higher-order case. It corrects (and extends) a result presented by Bergstra and Klop in [BK86], stating that outermost-fair rewriting is normalising for orthogonal CRSs.

The structure of this chapter is as follows. In Section 6.1 we define the notion of outermost-fair rewriting. Section 6.2 is concerned with the main result of this chapter: we prove that outermost-fair rewriting is normalising for almost orthogonal and fully extended higher-order rewriting systems. We discuss related work and the possibility to extend the result to the class of weakly orthogonal and fully extended higher-order rewriting systems.

6.1 Outermost-Fair Rewriting

In this section we consider left-linear higher-order rewriting systems.

The notion of outermost-fair rewriting is due to O'Donnell [O'D77]. A rewrite sequence is said to be outermost-fair, or in the terminology of [O'D77], eventually outermost, if every outermost redex occurrence is eventually eliminated. In this section we define outermost-fair rewriting. First the notion of outermost is defined.

Definition 6.1.1. Let $\mathcal{H}$ be a higher-order rewriting system. Let s be a term and let $\mathcal{U}$ be a set of redex occurrences in s. The set of *outermost redex occurrences of* $\mathcal{U}$, denoted by $\mathcal{OM}(\mathcal{U})$, is defined as the set consisting of all redex occurrences $(\phi, \mathrm{l} \to \mathrm{r}) \in \mathcal{U}$ that satisfy the following:
if $(\phi', \mathrm{l}' \to \mathrm{r}') \in \mathcal{U}$ and $(\phi', \mathrm{l}' \to \mathrm{r}') \preceq (\phi, \mathrm{l} \to \mathrm{r})$, then $\phi' = \phi$.

The sets $\mathcal{U}$ and $\mathcal{OM}(\mathcal{U})$ in the previous definition may contain overlapping redex occurrences.

Outermost-fair rewriting is defined using the outermost rewrite strategy. Recall from Section 1.2 of Chapter 1 that a strategy selects a set of redex occurrences in a term. The outermost rewrite strategy selects all redex occurrences in a term that are outermost. It is defined as follows. Recall that $\mathfrak{U}(s)$ denotes the set of redex occurrences in s.

Definition 6.1.2. Let $\mathcal{H}$ be a higher-order rewriting system. The *outermost rewrite strategy*, denoted by F_{om}, is a mapping

$$\mathsf{F}_{\mathsf{om}} : \mathsf{Terms} \to \mathfrak{U}$$

that is defined by

$$\mathsf{F}_{\mathsf{om}}(s) = \mathcal{OM}(\mathfrak{U}_{\mathcal{R}}(s)).$$

If $u \in \mathsf{F}_{\mathsf{om}}(s)$ then u is said to be an *outermost redex occurrence in* s and (s, u) is said to be an *outermost redex*.

The outermost rewrite strategy is non-deterministic since it may select more than one redex occurrence in a term. It can happen that the set $\mathsf{F}_{\mathsf{om}}(s)$ contains redex occurrences that are overlapping. This is illustrated in the following example.

Example 6.1.3. We consider the higher-order rewriting system for parallel-or, that is also given in Example 4.2.45 of Chapter 4. Its rewrite rules are the following:

$$\begin{array}{rcl} x.\mathsf{por}\mathsf{T}x & \to_{\mathrm{R}_1} & x.\mathsf{T} \\ x.\mathsf{por}x\mathsf{T} & \to_{\mathrm{R}_2} & x.\mathsf{T} \\ \mathsf{por}\mathsf{F}\mathsf{F} & \to_{\mathrm{R}_3} & \mathsf{F} \end{array}$$

We consider the term $\mathsf{por}(\mathsf{por}\mathsf{T}\mathsf{T})(\mathsf{por}\mathsf{T}\mathsf{T})$. The set $\mathsf{F}_{\mathsf{om}}(\mathsf{por}(\mathsf{por}\mathsf{T}\mathsf{T})(\mathsf{por}\mathsf{T}\mathsf{T}))$ consists of the following redex occurrences:

$$(0100, \mathrm{R}_1),$$
$$(0100, \mathrm{R}_2),$$
$$(100, \mathrm{R}_1),$$
$$(100, \mathrm{R}_2).$$

We proceed by defining the notion of outermost-fair rewriting for higher-order rewriting systems. Recall from Chapter 1 that a rewrite sequence is said to be fair with respect to a certain predicate on redexes, if for every term in the rewrite sequence, every redex in the term satisfying the predicate is eventually eliminated, in the sense that the redex eventually doesn't leave a residual that satisfies the predicate. The general definition of fairness with respect to a certain predicate is given in Definition 1.2.28 of Chapter 1. In this chapter we consider fairness in higher-order rewriting systems with respect to the predicate P defined by $P(s, w) \Leftrightarrow w \in \mathsf{F}_{\mathsf{om}}(s)$. Instead of saying 'fair with respect to P' we will say 'outermost-fair'. In an outermost-fair rewrite sequence every outermost redex occurrence is eliminated eventually. An outermost redex occurrence w is eliminated in a rewrite step $u : s \to s'$ if there is no $w' \in \mathsf{Res}(s, u)(w)$ such that $w' \in \mathsf{F}_{\mathsf{om}}(s')$. This can happen in three different ways:

1. by contracting w,
2. by contracting a redex occurrence that is overlapping with w,
3. by contracting a redex occurrence that creates a new outermost redex occurrence such that the residual of w is not outermost anymore.

We define an outermost-fair rewrite sequence as a rewrite sequence that either ends in a normal form or does not contain an infinite outermost chain. An infinite outermost chain is defined using the notion of an infinite residual chain, given in Definition 1.2.26.

Definition 6.1.4. Let $\mathcal{H}$ be a higher-order rewriting system. Let $\sigma : s_0 \xrightarrow{u_0} s_1 \xrightarrow{u_1} s_2 \xrightarrow{u_2} \ldots$ be an infinite rewrite sequence. An *infinite outermost chain in σ* is an infinite residual chain in σ of the form $w_m, w_{m+1}, w_{m+2}, \ldots$ with $w_m \in \mathfrak{U}_{\mathcal{R}}(s_m)$ for some m, such that moreover $w_n \in \mathsf{F}_{\mathsf{om}}(s_n)$ for every $n \geq m$.

The definition of an outermost-fair rewrite sequence is now given as follows.

Definition 6.1.5. Let $\mathcal{H}$ be a higher-order rewriting system. Let $\sigma : s_0 \xrightarrow{u_0} s_1 \xrightarrow{u_1} s_2 \xrightarrow{u_2} \ldots$ be a rewrite sequence. The rewrite sequence σ is said to be *outermost-fair* either if it ends in a normal form, or if it is infinite and it does not contain an infinite outermost chain.

Example 6.1.6. Consider the higher-order rewriting system with rewrite alphabet consisting of $f : 0 \to 0 \to 0, a : 0, b : 0, c : 0$ and defined by the following rewrite rules:

$$\begin{array}{rcl} x.fcx & \to_{R_1} & x.fbx \\ b & \to_{R_2} & c \\ a & \to_{R_3} & a \end{array}$$

1. The rewrite sequence
$$\sigma : fca \xrightarrow{u_0} fba \xrightarrow{u_1} fca \xrightarrow{u_2} \ldots$$
with for every $m \geq 0$
$$\begin{array}{rcl} u_{2m} & = & (00, R_1), \\ u_{2m+1} & = & (01, R_2), \end{array}$$
is outermost-fair. Note that there is an infinite residual chain of the form $(1, R_3), (1, R_3), (1, R_3), \ldots$ starting in the first term of σ. This infinite residual chain is not an infinite outermost chain, although infinitely many residuals in it are outermost.

2. The rewrite sequence
$$\sigma : faa \xrightarrow{u_0} faa \xrightarrow{u_1} faa \xrightarrow{u_2} \ldots$$
with for every $m \geq 0$
$$u_m = (01, R_3)$$
is not outermost-fair since we have an infinite sequence of residuals of outermost redex occurrences, namely
$$(1, R_3), (1, R_3), (1, R_3), \ldots.$$

6.2 Outermost-Fair Rewriting is Normalising

In this section we prove that outermost-fair rewriting is normalising for higher-order rewriting systems that are almost orthogonal and fully extended. The structure of the proof is basically the same as in [O'D77] and in [BK86]. First, we prove that a projection, called the weakly orthogonal projection, of an outermost-fair rewrite sequence over some rewrite step is again an outermost-fair rewrite sequence. Second, we prove that if the weakly orthogonal projection of an outermost-fair rewrite sequence over some rewrite step ends in a normal form, then the original outermost-fair rewrite sequence ends in a normal form. Finally, we prove that if a term s is weakly normalising, then all outermost-fair rewrite sequences starting in s end in a normal form. This means that an outermost-fair rewrite sequence eventually ends in a normal form whenever possible.

We first explain the construction of the weakly orthogonal projection, then we outline the structure of the proof in somewhat more detail.

The Weakly Orthogonal Projection. The weakly orthogonal projection is defined by van Oostrom in [Oos94]. It is used to prove confluence of weakly orthogonal higher-order rewriting systems by developments. Here we recall the construction of the weakly orthogonal projection which, as the terminology indicates, is defined for all weakly orthogonal higher-order rewriting system. We will use it for the smaller class consisting of all almost orthogonal and fully extended higher-order rewriting systems.

Let $\mathcal{H}$ be a weakly orthogonal higher-order rewriting system. Consider a finite or infinite rewrite sequence

$$\tilde{\sigma} : s_0 \xrightarrow{\tilde{u}_0} s_1 \xrightarrow{\tilde{u}_1} s_2 \xrightarrow{\tilde{u}_2} s_3 \xrightarrow{\tilde{u}_3} \ldots$$

and a rewrite step

$$v : s_0 \rightarrow t_0.$$

Let $\mathcal{V}_0 = \{v\}$ and define for $m \geq 0$ the following:

$$\begin{aligned} u_m &= \begin{cases} \tilde{u}_m & \text{if } \tilde{u}_m \parallel \mathcal{V}_m, \\ v_m & \text{if } \tilde{u}_m \sharp v_m \text{ for some } v_m \in \mathcal{V}_m, \end{cases} \\ \mathcal{V}_{m+1} &= \mathsf{Res}(s_m, u_m)(\mathcal{V}_m). \end{aligned}$$

Since the rewriting system is weakly orthogonal, we have for every $m \geq 0$ that $u_m : s_m \rightarrow s_{m+1}$ if $\tilde{u}_m : s_m \rightarrow s_{m+1}$. Let σ be the rewrite sequence

$$\sigma : s_0 \xrightarrow{u_0} s_1 \xrightarrow{u_1} s_2 \xrightarrow{u_2} s_3 \xrightarrow{u_3} \ldots.$$

By construction, we have for every $m \geq 0$ that $u_m \parallel \mathcal{V}_m$. Note that $\mathcal{V}_m$ is the set of residuals of v in s_m. For every $m \geq 0$ we define

$$\mathcal{U}_m = \mathsf{Res}(s_m, \mathcal{V}_m)(u_m).$$

Let τ be the development rewrite sequence

$$\tau : t_0 \xrightarrow{\mathcal{U}_0}\!\!\!\circ\; t_1 \xrightarrow{\mathcal{U}_1}\!\!\!\circ\; t_2 \xrightarrow{\mathcal{U}_2}\!\!\!\circ\; t_3 \xrightarrow{\mathcal{U}_3}\!\!\!\circ\; \ldots.$$

In a diagram, the situation is depicted as follows:

$$\begin{tikzcd}
\tilde{\sigma}: & s_0 \arrow[r, "\tilde{u}_0"] & s_1 \arrow[r, "\tilde{u}_1"] & s_2 \arrow[r, "\tilde{u}_2"] & s_3 \arrow[r, "\tilde{u}_3"] & \cdots \\
\sigma: & s_0 \arrow[r, "u_0"] \arrow[d, "\mathcal{V}_0"'] & s_1 \arrow[r, "u_1"] \arrow[d, "\mathcal{V}_1"'] & s_2 \arrow[r, "u_2"] \arrow[d, "\mathcal{V}_2"'] & s_3 \arrow[r, "u_3"] \arrow[d, "\mathcal{V}_3"'] & \cdots \\
\tau: & t_0 \arrow[r, "\mathcal{U}_0"] & t_1 \arrow[r, "\mathcal{U}_1"] & t_2 \arrow[r, "\mathcal{U}_2"] & t_3 \arrow[r, "\mathcal{U}_3"] & \cdots
\end{tikzcd}$$

We use the following terminology. The rewrite sequence σ is said to be a *simulation* of the rewrite sequence $\tilde{\sigma}$. The development rewrite sequence τ is said to be the *orthogonal projection* of the rewrite sequence σ over the rewrite step $v : s_0 \to t_0$, as in Section 1.2 of Chapter 1. The development rewrite sequence τ is said to be a *weakly orthogonal projection* of the rewrite sequence $\tilde{\sigma}$ over the rewrite step $v : s_0 \to t_0$.

If we consider weakly orthogonal higher-order rewriting systems, then a simulation σ of the rewrite sequence $\tilde{\sigma}$ as constructed for a weakly orthogonal projection of $\tilde{\sigma}$ over a rewrite step $v : s_0 \to t_0$ is not necessarily unique. The reason is that in the case that $\tilde{u}_m \sharp \mathcal{V}_m$ we define $u_m = v_m$ for a not uniquely determined $v_m \in \mathcal{V}_m$ such that $\tilde{u}_m \sharp v_m$. However, in the case we consider almost orthogonal higher-order rewriting systems, there is at most one redex occurrence $v_m \in \mathcal{V}_m$ such that $\tilde{u}_m \sharp v_m$, since any other one would also be overlapping with v_m, which is impossible, since the set $\mathcal{V}_m$ consists of redex occurrences that are non-overlapping. The non-determinism in the construction in the weakly orthogonal case is illustrated in the following example.

Example 6.2.1. Consider the higher-order rewriting system defined by the following rewrite rules:

$$\begin{aligned}
x.fx &\to_{\mathrm{R}_1} x.gxx \\
gaa &\to_{\mathrm{R}_2} gaa \\
a &\to_{\mathrm{R}_3} a
\end{aligned}$$

It is weakly orthogonal and represents a first-order term rewriting system. Let $\tilde{\sigma}$ be the rewrite sequence

$$\tilde{\sigma} : fa \xrightarrow{\tilde{u}_0} gaa \xrightarrow{\tilde{u}_1} gaa$$

with

$$\begin{aligned}
\tilde{u}_0 &= (0, \mathrm{R}_1), \\
\tilde{u}_1 &= (00, \mathrm{R}_2).
\end{aligned}$$

Consider the rewrite step

$$v : fa \to fa$$

with $v = (1, a \to a)$. We construct a weakly orthogonal projection of $\tilde{\sigma}$ over $v : fa \to fa$.

A first possibility is that we construct the weakly orthogonal projection of $\tilde{\sigma}$ over v as follows:

$$\begin{tikzcd}
\tilde{\sigma} : & fa \arrow[r, "\tilde{u}_0"] & gaa \arrow[r, "\tilde{u}_1"] & gaa \\
\sigma : & fa \arrow[r, "u_0"] \arrow[d, "\mathcal{V}_0"', "\circ" marking] & gaa \arrow[r, "u_1"] \arrow[d, "\mathcal{V}_1"', "\circ" marking] & gaa \arrow[d, "\mathcal{V}_2"', "\circ" marking] \\
\tau : & fa \arrow[r, "\mathcal{U}_0", "\circ" marking] & gaa \arrow[r, "\mathcal{U}_1", "\circ" marking] & gaa
\end{tikzcd}$$

with

$$\begin{array}{rclrcl}
u_0 & = & (0, \mathrm{R}_1), & \mathcal{U}_0 & = & \{(0, \mathrm{R}_1)\}, \\
u_1 & = & (01, \mathrm{R}_3), & \mathcal{U}_1 & = & \emptyset,
\end{array}$$

and

$$\mathcal{V}_0 = \{(1, \mathrm{R}_3)\}, \mathcal{V}_1 = \{(01, \mathrm{R}_3), (1, a \to a)\}, \mathcal{V}_2 = \{(1, \mathrm{R}_3)\}.$$

Another possibility is that we construct the following weakly orthogonal projection:

$$\begin{tikzcd}
\tilde{\sigma} : & fa \arrow[r, "\tilde{u}_0"] & gaa \arrow[r, "\tilde{u}_1"] & gaa \\
\sigma' : & fa \arrow[r, "u'_0"] \arrow[d, "\mathcal{V}'_0"', "\circ" marking] & gaa \arrow[r, "u'_1"] \arrow[d, "\mathcal{V}'_1"', "\circ" marking] & gaa \arrow[d, "\mathcal{V}'_2"', "\circ" marking] \\
\tau' : & fa \arrow[r, "\mathcal{U}'_0", "\circ" marking] & gaa \arrow[r, "\mathcal{U}'_1", "\circ" marking] & gaa
\end{tikzcd}$$

with

$$\begin{array}{rclrcl}
u'_0 & = & (0, \mathrm{R}_1), & \mathcal{U}'_0 & = & \{(0, \mathrm{R}_1)\}, \\
u'_1 & = & (1, \mathrm{R}_3), & \mathcal{U}'_1 & = & \emptyset,
\end{array}$$

and

$$\mathcal{V}'_0 = \{(0, \mathrm{R}_3)\}, \mathcal{V}'_1 = \{(01, \mathrm{R}_3), (1, \mathrm{R}_3)\}, \mathcal{V}'_2 = \{(01, \mathrm{R}_3)\}.$$

The Structure of the Proof. We will prove that outermost-fair rewriting is normalising for higher-order rewriting systems that are almost orthogonal and fully extended. The structure of the proof is as follows. Suppose that the finite or infinite rewrite sequence

$$\tilde{\sigma} : s_0 \xrightarrow{\tilde{u}_0} s_1 \xrightarrow{\tilde{u}_1} s_2 \xrightarrow{\tilde{u}_2} s_3 \xrightarrow{\tilde{u}_3} \ldots$$

is outermost-fair. Let $v : s_0 \to t_0$ be a rewrite step and construct the weakly orthogonal projection of $\tilde{\sigma}$ over v as above. In a diagram:

$$\begin{tikzcd}
\tilde{\sigma}: & s_0 \arrow[r, "\tilde{u}_0"] & s_1 \arrow[r, "\tilde{u}_1"] & s_2 \arrow[r, "\tilde{u}_2"] & s_3 \arrow[r, "\tilde{u}_3"] & \cdots
\end{tikzcd}$$

$$\begin{tikzcd}
\sigma: & s_0 \arrow[r, "u_0"] \arrow[d, "\mathcal{V}_0"', "\circ" marking] & s_1 \arrow[r, "u_1"] \arrow[d, "\mathcal{V}_1"', "\circ" marking] & s_2 \arrow[r, "u_2"] \arrow[d, "\mathcal{V}_2"', "\circ" marking] & s_3 \arrow[r, "u_3"] \arrow[d, "\mathcal{V}_3"', "\circ" marking] & \cdots \\
\tau: & t_0 \arrow[r, "\mathcal{U}_0", "\circ" marking] & t_1 \arrow[r, "\mathcal{U}_1", "\circ" marking] & t_2 \arrow[r, "\mathcal{U}_2", "\circ" marking] & t_3 \arrow[r, "\mathcal{U}_3", "\circ" marking] & \cdots
\end{tikzcd}$$

The proof consists of the following steps:

1. If $\tilde{\sigma}$ is outermost-fair, then σ is outermost-fair (Proposition 6.2.2).

2. If σ is outermost-fair, then τ is outermost-fair (Proposition 6.2.11).

3. If σ is outermost-fair and τ ends in a normal form, then σ ends in a normal form (Proposition 6.2.12).

4. If a term s has a normal form then any outermost-fair rewrite sequence starting in s ends in a normal form (Theorem 6.2.13).

The remainder of this section is devoted to the proof. The notation will be such that the diagram above applies.

The Proof.

In the remainder of this section we consider higher-order rewriting systems that are

1. *almost orthogonal as defined in Definition 4.2.49 of Chapter 4,*

2. *fully extended as defined in Definition 4.2.51 of Chapter 4.*

To start with, we make some observations concerning the nature of outermost redex occurrences in almost orthogonal and fully extended higher-order rewriting systems.

An important property of an outermost redex occurrence in a fully extended and almost orthogonal higher-order rewriting system is that it can only be eliminated by contracting an outermost redex occurrence. This property fails if the higher-order rewriting system contains also trivial critical pairs where the overlap is not at the root. Then an outermost redex occurrence w can be eliminated by contracting a redex occurrence that is overlapping with w but strictly below w. Consider for instance the higher-order rewriting system defined by the rewrite rules

$$\begin{aligned} f(ga) &\rightarrow_{R_1} f(gb) \\ ga &\rightarrow_{R_2} gb \\ gb &\rightarrow_{R_3} ga \\ b &\rightarrow_{R_4} a \end{aligned}$$

It is fully extended and weakly orthogonal but not almost orthogonal. The outermost redex occurrence $(0, R_1)$ in the term $f(ga)$ is eliminated in the rewrite step $(10, R_2) : f(ga) \rightarrow f(gb)$ because we have $(0, R_1) \sharp (10, R_2)$. However, the redex occurrence $(10, R_2)$ is not outermost itself.

Another important property of an outermost redex occurrence is that it only can be created by contracting an outermost redex occurrence. This property fails if the higher-order rewriting system is either not fully extended or not almost orthogonal. Consider for instance again the rewriting system above. In the rewrite step $(11, R_4) : f(gb) \rightarrow f(ga)$ the outermost redex occurrence $(0, R_1)$ is created. However, the redex occurrence $(11, R_4)$ is not outermost, since $(10, R_3) \prec (11, R_4)$. Note that both in this and in the previous example, the redex that is contracted is indeed not outermost itself, but is overlapping with an outermost redex. In a higher-order rewriting system that is orthogonal but not fully extended, it can also happen that an outermost redex occurrence is created by contracting a redex occurrence that is not outermost. Consider for instance the following higher-order rewriting system, due to Vincent van Oostrom:

$$\begin{aligned} z.f(x.z) &\rightarrow_{R_1} z.a \\ z.gz &\rightarrow_{R_2} z.a \\ z.hz &\rightarrow_{R_3} z.hz \end{aligned}$$

It is orthogonal but not fully extended because of the rewrite rule R_1. In the rewrite step $(1010, R_2) : f(x.h(gx)) \rightarrow f(x.ha)$ the outermost redex occurrence $(0, R_1)$ is created. However, the redex occurrence $(1010, R_2)$ is not outermost, since $(100, R_3) \prec (1010, R_2)$, and it is not overlapping with an outermost redex occurrence either. This example illustrates moreover that outermost-fair rewriting is not necessarily normalising for a higher-order rewriting system that is orthogonal but not fully extended.

In an almost orthogonal system every redex that is overlapping with an outermost redex is outermost itself. Finally, we remark that it is important to keep in mind that in almost orthogonal systems the relation $\sharp$ is transitive.

Now we perform the first step of the proof: we show that if a rewrite sequence is outermost-fair, then its simulation as constructed for the weakly orthogonal projection is outermost-fair.

Proposition 6.2.2. *Let $\tilde{\sigma}$ be an outermost-fair rewrite sequence issuing from s_0 and let $v : s_0 \to t_0$ be a rewrite step. Let σ be the simulation of $\tilde{\sigma}$ constructed for the weakly orthogonal projection of $\tilde{\sigma}$ over $v : s_0 \to t_0$. Then σ is outermost-fair.*

Proof. If $\tilde{\sigma}$ ends in a normal form, then σ ends in a normal form, so we restrict attention to the case that $\tilde{\sigma}$ is infinite. Let $\tilde{\sigma} : s_0 \xrightarrow{\tilde{u}_0} s_1 \xrightarrow{\tilde{u}_1} s_2 \xrightarrow{\tilde{u}_2} s_3 \xrightarrow{\tilde{u}_3} \dots$ be an infinite outermost-fair rewrite sequence. Let σ be the simulation of $\tilde{\sigma}$ as constructed for the weakly orthogonal projection. We prove that σ is outermost-fair. It suffices to prove the following: if an outermost redex $w_m \in \mathsf{F}_{\mathsf{om}}(s_m)$ is eliminated in the rewrite step $\tilde{u}_m : s_m \to s_{m+1}$, then it is also eliminated in the rewrite step $u_m : s_m \to s_{m+1}$. This is clearly the case if $u_m = \tilde{u}_m$. We show that it is also the case if $u_m \neq \tilde{u}_m$; then we have $u_m \sharp \tilde{u}_m$. Three possibilities are distinguished, that correspond to the three possible ways of eliminating an outermost redex occurrence.

1. Suppose the outermost redex occurrence w_m is eliminated in the rewrite step $\tilde{u}_m : s_m \to s_{m+1}$ because $w_m = \tilde{u}_m$. Then we have $u_m \sharp w_m$. Hence the outermost redex occurrence w_m is also eliminated in the rewrite step $u_m : s_m \to s_{m+1}$.

2. Suppose the outermost redex occurrence w_m is eliminated in the rewrite step $\tilde{u}_m : s_m \to s_{m+1}$ because $w_m \sharp \tilde{u}_m$. There are two possibilities. It can be the case that $u_m = w_m$. Then w_m is clearly eliminated in the rewrite step $u_m : s_m \to s_{m+1}$. Otherwise, we have $u_m \sharp w_m$, since the rewriting system is almost orthogonal, and also in that case w_m is eliminated in the rewrite step $u_m : s_m \to s_{m+1}$.

3. Finally, suppose the outermost redex occurrence w_m is eliminated in the rewrite step $\tilde{u}_m : s_m \to s_{m+1}$ because s_{m+1} contains an outermost redex occurrence w'_{m+1} that nests the residual of w_m in s_{m+1}. Then, the outermost redex occurrence w_m is also eliminated in the rewrite step $u_m : s_m \to s_{m+1}$.

□

In order to perform the second and third step of the proof we need to collect some more material. First we prove an auxiliary lemma, Lemma 6.2.4, which is needed for the proof of Proposition 6.2.11 (the second step of the proof). Then we will discuss a construction that we will use both in the proof of Proposition 6.2.11 and in the proof of Proposition 6.2.12 (the third step of the proof). In the proof of Lemma 6.2.4 we will make use of the following observations.

Lemma 6.2.3. *Let $\mathcal{H}$ be a higher-order rewriting system and let s be a term. Let $u \in \mathfrak{U}_{\mathcal{R}}(s)$ and $u' \in \mathfrak{U}_{\mathcal{R}}(s)$ and let $\mathcal{V}$ be a set of simultaneous redex occurrences in s that contains neither u nor u'. Suppose moreover that $u \parallel \mathcal{V}$ and $u' \parallel \mathcal{V}$.*

1. *If $u \sharp u'$, then after performing a development of $\mathcal{V}$ there is for every residual w of u a residual w' of u' such that $w \sharp w'$.*

2. *If the contraction of u erases u', then after performing a development of $\mathcal{V}$ there is for every residual w' of u' a residual w of u such that the contraction of w erases w'.*

Proof. Let s be a term and let $u \in \mathfrak{U}_{\mathcal{R}}(s)$ and $u' \in \mathfrak{U}_{\mathcal{R}}(s)$. Let $v \in \mathfrak{U}_{\mathcal{R}}(s)$ such that $v \neq u$, $v \neq u'$, $v \parallel u$ and $v \parallel u'$. Consider the rewrite step $v : s \to s'$. If $u \sharp u'$, then for every residual w of u in s', there is a residual w' of u' such that $w \sharp w'$. If contraction of u erases u', then for every residual w' of u' in s' there is a residual w of u such that contraction of w erases w'. The statement follows from the fact that residuals of simultaneous redexes are again simultaneous. $\Box$

Lemma 6.2.4. *Let $\mathcal{H}$ be a higher-order rewriting system. Let s_0 be a term. Let $\mathcal{U}_1$ and $\mathcal{U}_2$ be sets of simultaneous redex occurrences in the term s_0 such that $\mathcal{U}_1 \parallel \mathcal{U}_2$. Let $\mathcal{U}_1 : s_0 \mathrel{-\!\!\circ\!\!\!\twoheadrightarrow} s_1$ and $\mathcal{U}_2 : s_0 \mathrel{-\!\!\circ\!\!\!\twoheadrightarrow} s_2$. Let $\mathcal{U}_1' = \mathsf{Res}(s_0, \mathcal{U}_2)(\mathcal{U}_1)$ and $\mathcal{U}_2' = \mathsf{Res}(s_0, \mathcal{U}_1)(\mathcal{U}_2)$. In a diagram:*

$$\begin{tikzcd}
s_0 \arrow[r, "\mathcal{U}_1"] \arrow[d, "\mathcal{U}_2"'] & s_1 \arrow[d, "\mathcal{U}_2'"] \\
s_2 \arrow[r, "\mathcal{U}_1'"] & s_3
\end{tikzcd}$$

Let w_0 be a redex occurrence in s_0, let w_2 be a redex occurrence in s_2 and let w_3 be a redex occurrence in s_3 such that

$$\begin{aligned}
w_2 &\in \mathsf{Res}(s_0, \mathcal{U}_2)(w_0), \\
w_3 &\in \mathsf{Res}(s_2, \mathcal{U}_1')(w_2).
\end{aligned}$$

1. *Then there exists a unique redex occurrence w_1 in s_1 such that*

$$\begin{aligned}
w_1 &\in \mathsf{Res}(s_0, \mathcal{U}_1)(w_0), \\
w_3 &\in \mathsf{Res}(s_1, \mathcal{U}_2')(w_1).
\end{aligned}$$

2. *If moreover $w_0 \in \mathsf{F}_{\mathsf{om}}(s_0)$, $w_2 \in \mathsf{F}_{\mathsf{om}}(s_2)$ and $w_3 \in \mathsf{F}_{\mathsf{om}}(s_3)$, then $w_1 \in \mathsf{F}_{\mathsf{om}}(s_1)$.*

Proof.

1. First we prove that the redex occurrence w_0 has a residual in s_1. Suppose towards a contradiction that $\mathsf{Res}(s_0, \mathcal{U}_1)(w_0) = \emptyset$. That is, the redex w_0 is eliminated in the complete development $\mathcal{U}_1 : s_0 \mathrel{-\!\!\circ\!\!\!\twoheadrightarrow} s_1$. This can happen in one of the following three ways:

(a) $w_0 \in \mathcal{U}_1$,

(b) there is a redex occurrence $u_1 \in \mathcal{U}_1$ such that $u_1 \sharp w_0$,

(c) there is a redex occurrence $u_1 \in \mathcal{U}_1$ such that contraction of u_1 in the complete development of $\mathcal{U}_1$ erases the redex w_0.

We show that in all three cases $\mathsf{Res}(s_2, \mathcal{U}_1')(w_2) = \emptyset$, which contradicts the hypothesis, since $w_3 \in \mathsf{Res}(s_2, \mathcal{U}_1')(w_2)$.

In the first case, $\mathsf{Res}(s_0, \mathcal{U}_2)(w_0) \subseteq \mathcal{U}_1'$. That is, all residuals of the redex occurrence w_0 are contracted in the complete development $\mathcal{U}_1' : s_2 \multimap\!\!\!\rightarrow s_3$. Since $w_2 \in \mathsf{Res}(s_0, \mathcal{U}_2)(w_0)$, we have $\mathsf{Res}(s_2, \mathcal{U}_1')(w_2) = \emptyset$.

Suppose that there is a redex occurrence $u_1 \in \mathcal{U}_1$ such that $u_1 \sharp w_0$. By the hypothesis, we have that $u_1 \parallel \mathcal{U}_2$. Since the redex occurrence w_0 has a residual w_2 in s_2, we have moreover that $w_0 \parallel \mathcal{U}_2$ and $w_0 \notin \mathcal{U}_2$. Then, there exists for every redex occurrence $w_2' \in \mathsf{Res}(s_0, \mathcal{U}_2)(w_0)$ a redex occurrence $u_1' \in \mathcal{U}_1'$ such that $u_1' \sharp w_2'$. Since $w_2 \in \mathsf{Res}(s_0, \mathcal{U}_2)(w_0)$, we have $\mathsf{Res}(s_2, \mathcal{U}_1')(w_2) = \emptyset$.

In the third case, there exists for every redex occurrence $w_2' \in \mathsf{Res}(s_0, \mathcal{U}_2)(w_0)$ a redex occurrence $u_1' \in \mathcal{U}_1'$ such that contraction of u_1' erases w_2'. Since $w_2 \in \mathsf{Res}(s_0, \mathcal{U}_2)(w_0)$, we have that $\mathsf{Res}(s_2, \mathcal{U}_1')(w_2) = \emptyset$.

Second, it needs to be shown that a redex occurrence $w_1 \in \mathsf{Res}(s_0, \mathcal{U}_1)(w_0)$ such that $w_3 \in \mathsf{Res}(s_1, \mathcal{U}_2')(w_1)$ is unique. This is the case since a redex occurrence can be the residual of at most one redex occurrence.

2. Suppose moreover $w_0 \in \mathsf{F_{om}}(s_0)$, $w_2 \in \mathsf{F_{om}}(s_2)$ and $w_3 \in \mathsf{F_{om}}(s_3)$. Then

$$\begin{aligned}\mathsf{Res}(s_0, \mathcal{U}_2)(w_0) &= \{w_2\},\\ \mathsf{Res}(s_2, \mathcal{U}_1')(w_2) &= \{w_3\}.\end{aligned}$$

Suppose towards a contradiction that $w_1 \notin \mathsf{F_{om}}(s_1)$. This means that performing a complete development of the set of redex occurrences $\mathcal{U}_1$ creates a redex occurrence v_1 above the redex occurrence w_1. We have that $v_1 \notin \mathcal{U}_2'$ since the redex occurrence v_1 is created by performing the complete development of $\mathcal{U}_1$. Moreover, $v_1 \parallel \mathcal{U}_2'$ since if there is a redex occurrence $u_2' \in \mathcal{U}_2'$ such that $v_1 \sharp u_2'$, then u_2' is also an outermost redex occurrence, since the rewriting system is almost orthogonal. So if there would be a $u_2' \in \mathcal{U}_2'$ such that $v_1 \sharp u_2'$, then w_0 would not be an outermost redex occurrence in s_0. Therefore it must be the case that $v_1 \parallel \mathcal{U}_2'$, and hence v_1 has a unique residual in s_3 which is above the redex occurrence w_3. □

We will consider the orthogonal projection of a development rewrite sequence over a development rewrite sequence. Its (standard) construction is as follows. Let

$$\sigma^0 : t_0^0 \xrightarrow{\mathcal{U}_0} t_1^0 \xrightarrow{\mathcal{U}_1} t_2^0 \xrightarrow{\mathcal{U}_2} \ldots$$

be a development rewrite sequence and let

$$\tau_0 : t_0^0 \xrightarrow{\mathcal{V}_0^0} \ldots t_0^{p-1} \xrightarrow{\mathcal{V}_0^{p-1}} t_0^p$$

be a development rewrite sequence. We construct the projection of σ^0 over τ_0 as follows.

For every n with $0 \le n \le p$ the set of redex occurrences $\mathcal{U}_0^n$ is defined by induction on n:

1. $\mathcal{U}_0^0 = \mathcal{U}_0$,
2. $\mathcal{U}_0^{n+1} = \mathsf{Res}(t_0^n, \mathcal{V}_0^n)(\mathcal{U}_0^n)$ for n with $0 \le n \le p-1$.

So $\mathcal{U}_0^n$ consists of the residuals of $\mathcal{U}_0$ in t_0^n.

Suppose for every n with $0 \le n \le p$ the terms t_q^n and the sets of redex occurrences $\mathcal{U}_q^n$ have been defined, and for $0 \le n \le p-1$ the sets $\mathcal{V}_q^n$ have been defined. We define for n with $0 \le n \le p$ terms t_{q+1}^n and sets of redex occurrences $\mathcal{U}_{q+1}^n$, and for n with $0 \le n \le p-1$ sets of redex occurrences $\mathcal{V}_{q+1}^n$ as follows:

1. $\mathcal{V}_{q+1}^n = \mathsf{Res}(t_q^n, \mathcal{U}_q^n)(\mathcal{V}_q^n)$ for n with $0 \le n \le p-1$,
2. for $0 < n \le p$ the term t_{q+1}^n is obtained by performing a complete development of the set of redex occurrences $\mathcal{V}_{q+1}^{n-1}$ in t_{q+1}^{n-1},
3. the set $\mathcal{U}_{q+1}^n$ is defined by induction on n for $0 \le n \le p$:
   (a) $\mathcal{U}_{q+1}^0 = \mathcal{U}_{q+1}$,
   (b) $\mathcal{U}_{q+1}^{n+1} = \mathsf{Res}(t_{q+1}^n, \mathcal{V}_{q+1}^n)(\mathcal{U}_{q+1}^n)$ for n with $0 \le n \le p-1$.

For m with $m \ge 0$ the development rewrite sequence τ_m is defined as

$$\tau_m : t_m^0 \xrightarrow{\mathcal{V}_m^0} t_m^1 \xrightarrow{\mathcal{V}_m^1} \ldots t_m^{p-1} \xrightarrow{\mathcal{V}_m^{p-1}} t_m^p.$$

The situation is depicted as follows:

$$\begin{tikzcd}
t_0^0 \arrow[r, "\mathcal{U}_0^0"] \arrow[d, "\mathcal{V}_0^0"'] & t_1^0 \arrow[r, "\mathcal{U}_1^0"] \arrow[d, "\mathcal{V}_1^0"'] & t_2^0 \arrow[r, "\mathcal{U}_2^0"] \arrow[d, "\mathcal{V}_2^0"'] & \cdots \\
t_0^1 \arrow[r, "\mathcal{U}_0^1"] \arrow[d, "\mathcal{V}_0^1"'] & t_1^1 \arrow[r, "\mathcal{U}_1^1"] \arrow[d, "\mathcal{V}_1^1"'] & t_2^1 \arrow[r, "\mathcal{U}_2^1"] \arrow[d, "\mathcal{V}_2^1"'] & \cdots \\
\vdots & \vdots & \vdots & \\
t_0^{p-1} \arrow[r, "\mathcal{U}_0^{p-1}"] \arrow[d, "\mathcal{V}_0^{p-1}"'] & t_1^{p-1} \arrow[r, "\mathcal{U}_1^{p-1}"] \arrow[d, "\mathcal{V}_1^{p-1}"'] & t_2^{p-1} \arrow[r, "\mathcal{U}_2^{p-1}"] \arrow[d, "\mathcal{V}_2^{p-1}"'] & \cdots \\
t_0^p \arrow[r, "\mathcal{U}_0^p"'] & t_1^p \arrow[r, "\mathcal{U}_1^p"'] & t_2^p \arrow[r, "\mathcal{U}_2^p"'] & \cdots
\end{tikzcd}$$

We now discuss a construction that will be used in the proofs of Proposition 6.2.11 and of Proposition 6.2.12. It is way of performing a complete development of a set of simultaneous redex occurrences $\mathcal{V}$ in phases. The idea is that, starting in the initial term, a complete development of the redex occurrences in $\mathcal{V}$ that are outermost is performed. This yields a second term, in which some of the residuals of redex occurrences in $\mathcal{V}$ may have become outermost. These redex occurrences are contracted in a second complete development. We repeat this process until no residual of $\mathcal{V}$ is an outermost redex occurrence. Then, a complete development of all residuals of $\mathcal{V}$ is performed. We will call such a development rewrite sequence a complete outermost development rewrite sequence. It is defined as follows.

Definition 6.2.5. Let $\mathcal{H}$ be a higher-order rewriting system. Let s be a term and let $\mathcal{V}$ be a set of simultaneous redex occurrences in s. Let $\mathcal{V} : s \mathrel{-\!\!\circ\!\!\!\to} t$.

We define a natural number p and we define for every n with $0 \leq n \leq p$ a term s^0_n and sets of redex occurrences $\mathcal{W}^0_n$ and $\mathcal{V}^0_n$ by induction on n.

1. $n = 0$. The term s^0_0 and the sets of redex occurrences $\mathcal{W}^0_0$ and $\mathcal{V}^0_0$ are defined as follows:

   (a) $s^0_0 = s$,

   (b) $\mathcal{W}^0_0 = \mathcal{V}$,

   (c) $\mathcal{V}^0_0 = \mathcal{W}^0_0 \cap \mathsf{F_{om}}(s^0_0)$.

2. $n > 0$. Suppose the term s^0_n and the sets of redex occurrences $\mathcal{W}^0_n$ and $\mathcal{V}^0_n$ are defined.

   If $\mathcal{V}^0_n = \emptyset$, then we put $p = n$ and we are done.

   Otherwise, we define the term s^0_{n+1} and the sets of redex occurrences $\mathcal{W}^0_{n+1}$ and $\mathcal{V}^0_{n+1}$ as follows:

   (a) s^0_{n+1} is the term obtained by performing a complete development of the set of redex occurrences $\mathcal{V}^0_n$, that is, $\mathcal{V}^0_n : s^0_n \mathrel{-\!\!\circ\!\!\!\to} s^0_{n+1}$,

   (b) $\mathcal{W}^0_{n+1} = \mathsf{Res}(s^0_n, \mathcal{V}^0_n)(\mathcal{W}^0_n)$,

   (c) $\mathcal{V}^0_{n+1} = \mathcal{W}^0_{n+1} \cap \mathsf{F_{om}}(s^0_{n+1})$.

Now the following development rewrite sequences are defined.

1. The *complete outermost development rewrite sequence of* $\mathcal{V}$ is the development rewrite sequence

$$\sigma; \mathcal{W}^0_p : s^0_0 \overset{\mathcal{V}^0_0}{\mathrel{-\!\!\circ\!\!\!\to}} s^0_1 \overset{\mathcal{V}^0_1}{\mathrel{-\!\!\circ\!\!\!\to}} \ldots s^0_{p-1} \overset{\mathcal{V}^0_{p-1}}{\mathrel{-\!\!\circ\!\!\!\to}} s^0_p \overset{\mathcal{W}^0_p}{\mathrel{-\!\!\circ\!\!\!\to}} t$$

   consisting of p complete developments of sets of outermost redex occurrences, followed by a complete development of redex occurrences that are not outermost.

2. An *outermost development rewrite sequence of* $\mathcal{V}$ is a development rewrite sequence

$$\tau;\mathcal{W}_0^p : s_0^0 \xrightarrow{\mathcal{U}_0^0} s_0^1 \xrightarrow{\mathcal{U}_0^1} \ldots s_0^{p-1} \xrightarrow{\mathcal{U}_0^{p-1}} s_p^0 \xrightarrow{\mathcal{W}_0^p} t$$

such that for the orthogonal projection of σ over τ

$$\begin{tikzcd}
s_0^0 \arrow[r, "\mathcal{V}_0^0"] \arrow[d, "\mathcal{U}_0^0"'] & s_1^0 \arrow[r, "\mathcal{V}_1^0"] \arrow[d, "\mathcal{U}_1^0"'] & \cdots \arrow[r, "\mathcal{V}_{p-1}^0"] & s_p^0 \arrow[d, "\mathcal{U}_p^0"'] \\
s_0^1 \arrow[r, "\mathcal{V}_0^1"] \arrow[d, "\mathcal{U}_0^1"'] & s_1^1 \arrow[r, "\mathcal{V}_1^1"] \arrow[d, "\mathcal{U}_1^1"'] & \cdots \arrow[r, "\mathcal{V}_{p-1}^1"] & {} \arrow[d, "\mathcal{U}_p^1"'] \\
\vdots & \vdots & & \vdots \\
s_0^{p-1} \arrow[r, "\mathcal{V}_0^{p-1}"] \arrow[d, "\mathcal{U}_0^{p-1}"'] & s_1^{p-1} \arrow[r, "\mathcal{V}_1^{p-1}"] \arrow[d, "\mathcal{U}_1^{p-1}"'] & \cdots \arrow[r, "\mathcal{V}_{p-1}^{p-1}"] & {} \arrow[d, "\mathcal{U}_p^{p-1}"'] \\
s_0^p \arrow[r, "\mathcal{V}_0^p"'] & s_1^p \arrow[r, "\mathcal{V}_1^p"'] & \cdots \arrow[r, "\mathcal{V}_{p-1}^p"'] & {}
\end{tikzcd}$$

we have:

(a) $\mathcal{U}_n^m = \emptyset$ for $n > 0$ and $m < n$,

(b) $\mathcal{U}_0^n \subseteq \mathsf{F}_{\mathsf{om}}(s_0^n)$,

(c) $\mathsf{Res}(s_0^n, \mathcal{V}_0^1; \ldots; \mathcal{V}_0^n)(\mathcal{U}_0^n) \subseteq \mathcal{V}_n^0$.

Moreover, $\mathcal{W}_0^p$ is the set of residuals of $\mathcal{V}$ in s_0^p.

In a complete outermost development rewrite sequence of a set $\mathcal{V}$, the last complete development consists of contractions of redex occurrences that are not outermost. In an arbitrary outermost development rewrite sequence however, the last complete development may contain contractions of redex occurrences that are outermost.

Example 6.2.6. Consider the higher-order rewriting system defined by the rewrite rules

$$\begin{aligned}
x.y.fbxy &\to_{\mathrm{R}_1} x.y.c \\
x.gx &\to_{\mathrm{R}_2} x.x \\
a &\to_{\mathrm{R}_3} b
\end{aligned}$$

We consider the term $f(ga)(ga)(ga)$ and the set of redex occurrences

$$\begin{aligned}\mathcal{V} &= \{(0010,\mathrm{R}_2),(010,\mathrm{R}_2),\\ &\quad (0011,\mathrm{R}_3),(011,\mathrm{R}_3),(11,\mathrm{R}_3)\}.\end{aligned}$$

The complete outermost development rewrite sequence of $\mathcal{V}$ issuing from the term $f(ga)(ga)(ga)$ is

$$f(ga)(ga)(ga) \xrightarrow{\mathcal{V}_0} faa(ga) \xrightarrow{\mathcal{V}_1} fbb(ga) \xrightarrow{\mathcal{W}_2} fbb(gb)$$

with

$$\begin{aligned}\mathcal{V}_0 &= \{(0010,\mathrm{R}_2),(010,\mathrm{R}_2)\},\\ \mathcal{V}_1 &= \{(0011,\mathrm{R}_3),(011,\mathrm{R}_3)\},\\ \mathcal{W}_2 &= \{(11,\mathrm{R}_3)\}.\end{aligned}$$

An example of an outermost development rewrite sequence of $\mathcal{V}$ issuing from $f(ga)(ga)(ga)$ is

$$f(ga)(ga)(ga) \xrightarrow{\mathcal{V}_0'} fa(ga)(ga) \xrightarrow{\mathcal{V}_1'} fb(ga)(ga) \xrightarrow{\mathcal{W}_2'} fbb(gb)$$

with

$$\begin{aligned}\mathcal{V}_0' &= \{(0010,\mathrm{R}_2)\},\\ \mathcal{V}_1' &= \{(0011,\mathrm{R}_3)\},\\ \mathcal{W}_2' &= \{(010,\mathrm{R}_2),(011,\mathrm{R}_3),(11,\mathrm{R}_3)\}.\end{aligned}$$

In the following definition we define what it means that a complete development creates an outermost redex.

Definition 6.2.7. Let s be a term and let $\mathcal{V}$ be a set of simultaneous redex occurrences in s. Let $\mathcal{V} : s \multimap\!\!\rightarrow t$ and let $w \in \mathsf{F}_{\mathsf{om}}(t)$. The complete development $\mathcal{V} : s \multimap\!\!\rightarrow t$ is said to *create the outermost redex occurrence* w if there is no redex occurrence $w' \in \mathsf{F}_{\mathsf{om}}(s)$ such that $w \in \mathsf{Res}(s,\mathcal{V})(w')$.

Note that there are two ways in which an outermost redex occurrence w in a term t can be created by a complete development $\mathcal{V} : s \multimap\!\!\rightarrow t$: either if there is no redex occurrence w' in s such that $w \in \mathsf{Res}(s,\mathcal{V})(w')$, or if there is w' in s such that $w \in \mathsf{Res}(s,\mathcal{V})(w')$, but such that $w' \notin \mathsf{F}_{\mathsf{om}}(s)$.

We will be interested in the most economic way to create an outermost redex occurrence. This is formalised by the notion of a minimal outermost development rewrite sequence that creates some outermost redex occurrence. As the terminology indicates, we make use of the definition of an outermost development rewrite sequence. An easy observation is that if a complete development $\mathcal{V} : s \twoheadrightarrow t$ creates an outermost redex occurrence w in t, and if

$$\sigma : s_0 \xrightarrow{\mathcal{V}_0} s_1 \xrightarrow{\mathcal{V}_1} \ldots \xrightarrow{\mathcal{V}_{p-1}} s_p \xrightarrow{\mathcal{W}_p} t$$

is the complete outermost development rewrite sequence of $\mathcal{V}$, then there is a redex occurrence $w_p \in \mathsf{F_{om}}(s_p)$ such that $w \in \mathsf{Res}(s_p, \mathcal{W}_p)(w_p)$. This is the case since by definition in the complete development of $\mathcal{W}_p$ only redex occurrences are contracted that are not outermost. Such a development cannot create an outermost redex occurrence, since an outermost redex occurrence can only be created by the contraction of an outermost redex occurrence.

Definition 6.2.8. Let s_0 be a term and let $\mathcal{V}$ be a set of simultaneous redex occurrences in s_0 with $\mathcal{V} : s_0 \twoheadrightarrow t$. Let $w \in \mathsf{F_{om}}(t)$ be an outermost redex occurrence that is created by the complete development of $\mathcal{V}$.

Let

$$\sigma : s_0 \xrightarrow{\mathcal{V}_0} s_1 \xrightarrow{\mathcal{V}_1} \ldots \xrightarrow{\mathcal{V}_{p-1}} s_p \xrightarrow{\mathcal{W}_p} t$$

be an outermost development rewrite sequence of $\mathcal{V}$. Then σ is said to *create* the outermost redex occurrence w if there is a redex occurrence $w_p \in \mathsf{F_{om}}(s_p)$ such that $w \in \mathsf{Res}(s_p, \mathcal{W}_p)(w_p)$.

Moreover, σ is said to be a *minimal outermost development rewrite sequence of* $\mathcal{V}$ *creating* w, if whenever a set $\mathcal{V}_n$ for some n with $0 \leq n \leq p-1$ is replaced by a proper subset, the so-obtained development rewrite sequence doesn't create the outermost redex occurrence w anymore, or it is not an outermost development rewrite sequence of $\mathcal{V}$ anymore.

Example 6.2.9. The second outermost development rewrite sequence in Example 6.2.6 is an outermost development rewrite sequence creating the outermost redex occurrence $(000, \mathtt{R}_1)$. In fact, it is a minimal outermost development rewrite sequence creating that outermost redex occurrence.

In the following lemma we show that an outermost redex occurrence can be created in a unique most economic way.

Lemma 6.2.10. *Let s_0 be a term and let $\mathcal{V}$ be a set of simultaneous redex occurrences in s_0 with $\mathcal{V} : s_0 \twoheadrightarrow t$. Suppose the complete development of $\mathcal{V}$ creates an outermost redex occurrence $w \in \mathsf{F_{om}}(t)$. There is a unique minimal outermost development rewrite sequence of $\mathcal{V}$ creating the outermost redex occurrence w.*

Proof. Let

$$\sigma; \mathcal{W}_p^0 : s_0^0 \xrightarrow{\mathcal{V}_0^0} s_1^0 \xrightarrow{\mathcal{V}_1^0} \ldots s_{p-1}^0 \xrightarrow{\mathcal{V}_{p-1}^0} s_p^0 \xrightarrow{\mathcal{W}_p^0} t$$

and

$$\tau; \mathcal{W}_0^p : s_m^0 \xrightarrow{\mathcal{U}_m^0} s_m^1 \xrightarrow{\mathcal{U}_m^1} \ldots s_m^{p-1} \xrightarrow{\mathcal{U}_m^{p-1}} s_m^p \xrightarrow{\mathcal{W}_0^p} t$$

be two outermost development rewrite sequences of $\mathcal{V}$ that create the outermost redex occurrence $w \in \mathsf{F_{om}}(t)$. Construct the orthogonal projection of σ over τ:

$$\begin{tikzcd}
s_0^0 \arrow[r, "\mathcal{V}_0^0"] \arrow[d, "\mathcal{U}_0^0"'] & s_1^0 \arrow[r, "\mathcal{V}_1^0"] \arrow[d, "\mathcal{U}_1^0"'] & {} \arrow[r, phantom, "\cdots"] & {} \arrow[r, "\mathcal{V}_{p-1}^0"] & s_p^0 \arrow[d, "\mathcal{U}_p^0"'] \\
s_0^1 \arrow[r, "\mathcal{V}_0^1"] \arrow[d, "\mathcal{U}_0^1"'] & s_1^1 \arrow[r, "\mathcal{V}_1^1"] \arrow[d, "\mathcal{U}_1^1"'] & {} \arrow[r, phantom, "\cdots"] & {} \arrow[r, "\mathcal{V}_{p-1}^1"] & {} \arrow[d, "\mathcal{U}_p^1"'] \\
\vdots & \vdots & & & \vdots \\
s_0^{p-1} \arrow[r, "\mathcal{V}_0^{p-1}"] \arrow[d, "\mathcal{U}_0^{p-1}"'] & s_1^{p-1} \arrow[r, "\mathcal{V}_1^{p-1}"] \arrow[d, "\mathcal{U}_1^{p-1}"'] & {} \arrow[r, phantom, "\cdots"] & {} \arrow[r, "\mathcal{V}_{p-1}^{p-1}"] & {} \arrow[d, "\mathcal{U}_p^{p-1}"'] \\
s_0^p \arrow[r, "\mathcal{V}_0^p"'] & s_1^p \arrow[r, "\mathcal{V}_1^p"'] & {} \arrow[r, phantom, "\cdots"] & {} \arrow[r, "\mathcal{V}_{p-1}^p"'] & {}
\end{tikzcd}$$

Define for m with $0 \leq m \leq p-1$ the set of redex occurrences $\mathcal{U}_m = \mathcal{U}_m^m \cap \mathcal{V}_m^m$. This defines a development rewrite sequence $s_0^0 \xrightarrow{\mathcal{U}_0} s_1^1 \xrightarrow{\mathcal{U}_1} \ldots \xrightarrow{\mathcal{U}_{p-1}} s_p^p$. Define $\mathcal{W}_p$ to be the set of residuals of $\mathcal{V}$ in s_p^p. Then the development rewrite sequence

$$s_0^0 \xrightarrow{\mathcal{U}_0} s_1^1 \xrightarrow{\mathcal{U}_1} \ldots \xrightarrow{\mathcal{U}_{p-1}} s_p^p \xrightarrow{\mathcal{U}_p} t$$

is also an outermost development rewrite sequence of $\mathcal{V}$, and creates moreover the outermost redex occurrence w. □

By now we have collected all the material needed to perform the second step of the proof: in the following proposition we prove that the orthogonal projection of an outermost-fair rewrite sequence over some rewrite step is outermost-fair.

Proposition 6.2.11. *Let $\sigma : s_0 \xrightarrow{u_0} s_1 \xrightarrow{u_1} s_2 \xrightarrow{u_2} \ldots$ be an infinite rewrite sequence. Let $v : s_0 \to t_0$ be a rewrite step. Let $\tau : t_0 \xrightarrow{\mathcal{U}_0} t_1 \xrightarrow{\mathcal{U}_1} t_2 \xrightarrow{\mathcal{U}_2} \ldots$ be the orthogonal projection of the rewrite sequence σ over the rewrite step $v : s_0 \to t_0$. If τ contains an infinite outermost chain, then σ contains an infinite outermost chain.*

Proof. Let $\sigma : t_0^0 \xrightarrow{u_0} t_1^0 \xrightarrow{u_1} t_2^0 \xrightarrow{u_2} \ldots$ be an infinite rewrite sequence. Let $\tau : t_0 \xrightarrow{\mathcal{U}_0} t_1 \xrightarrow{\mathcal{U}_1} t_2 \xrightarrow{\mathcal{U}_2} \ldots$ be the orthogonal projection of the rewrite sequence σ over the rewrite step $v : t_0^0 \to t_0$. We recall that we are in the following situation:

$$\begin{tikzcd}
\sigma : & s_0 \arrow[r, "u_0"] \arrow[d, "\mathcal{V}_0"'] & s_1 \arrow[r, "u_1"] \arrow[d, "\mathcal{V}_1"'] & s_2 \arrow[r, "u_2"] \arrow[d, "\mathcal{V}_2"'] & \cdots \\
\tau : & t_0 \arrow[r, "\mathcal{U}_0"] & t_1 \arrow[r, "\mathcal{U}_1"] & t_2 \arrow[r, "\mathcal{U}_2"] & \cdots
\end{tikzcd}$$

Suppose that the rewrite sequence τ contains an infinite outermost chain $\vec{w}$ starting with the redex occurrence $w_m \in \mathsf{Fom}(t_m)$. Let this sequence contain further redex occurrences $w_{m+1} \in \mathsf{Fom}(t_{m+1}), w_{m+2} \in \mathsf{Fom}(t_{m+2}), \dots$. We prove that the rewrite sequence σ contains an infinite outermost chain.

By Lemma 6.2.10 there is a unique minimal outermost development rewrite sequence of $\mathcal{V}_m$ creating the outermost redex occurrence w_m. Let this outermost development rewrite sequence be

$$\rho_m : s_m = t^0_m \xrightarrow{\mathcal{V}^0_m} t^1_m \xrightarrow{\mathcal{V}^1_m} \dots t^{p-1}_m \xrightarrow{\mathcal{V}^{p-1}_m} t^p_m \xrightarrow{\mathcal{W}_m} t_m.$$

We construct the projection of the rewrite sequence $\sigma^0_m : s_m \xrightarrow{u_m} s_{m+1} \xrightarrow{u_{m+1}} s_{m+2} \xrightarrow{u_{m+2}} \dots$ over the development rewrite sequence ρ_m. This yields the following:

$$\begin{tikzcd}
\sigma^0_m : & t^0_m \arrow[r, "\mathcal{U}^0_m"] \arrow[d, "\mathcal{V}^0_m"] & t^0_{m+1} \arrow[r, "\mathcal{U}^0_{m+1}"] \arrow[d, "\mathcal{V}^0_{m+1}"] & t^0_{m+2} \arrow[r, "\mathcal{U}^0_{m+2}"] \arrow[d, "\mathcal{V}^0_{m+2}"] & \cdots \\
\sigma^1_m : & t^1_m \arrow[r, "\mathcal{U}^1_m"] \arrow[d, "\mathcal{V}^1_m"] & t^1_{m+1} \arrow[r, "\mathcal{U}^1_{m+1}"] \arrow[d, "\mathcal{V}^1_{m+1}"] & t^1_{m+2} \arrow[r, "\mathcal{U}^1_{m+2}"] \arrow[d, "\mathcal{V}^1_{m+2}"] & \cdots \\
 & \vdots \arrow[d] & \vdots \arrow[d] & \vdots \arrow[d] & \\
\sigma^p_m : & t^p_m \arrow[r, "\mathcal{U}^p_m"] \arrow[d, "\mathcal{W}_m"] & t^p_{m+1} \arrow[r, "\mathcal{U}^p_{m+1}"] \arrow[d, "\mathcal{W}_{m+1}"] & t^p_{m+2} \arrow[r, "\mathcal{U}^p_{m+2}"] \arrow[d, "\mathcal{W}_{m+2}"] & \cdots \\
\tau_m : & t_m \arrow[r, "\mathcal{U}_m"] & t_{m+1} \arrow[r, "\mathcal{U}_{m+1}"] & t_{m+2} \arrow[r, "\mathcal{U}_{m+2}"] & \cdots
\end{tikzcd}$$

with $t^0_n = s_n$ for $n \geq m$. First we show that the development rewrite sequence σ^p_m contains an infinite sequence of residuals of outermost redex occurrences. This is a consequence of Lemma 6.2.4, as follows. We show by induction on q that the term t^p_{m+q} contains an outermost redex occurrence w^p_{m+q} such that $w_{m+q} \in \mathsf{Res}(t^p_{m+q}, \mathcal{W}_{m+q})(w^p_{m+q})$ and, if $q > 0$, $w^p_{m+q} \in \mathsf{Res}(t^p_{m+q-1}, \mathcal{U}^p_{m+q-1})(w^p_{m+q-1})$.

Let $q = 0$. By definition of the minimal outermost development rewrite sequence creating w_m, there is a redex occurrence $w^p_m \in \mathsf{Fom}(t^p_m)$ such that $w_m \in \mathsf{Res}(t^p_m, \mathcal{W}_m)(w^p_m)$.

Let $q > 0$. By the induction hypothesis, there is an outermost redex occurrence w^p_{q-1} in t^p_{q-1} such that $w_{q-1} \in \mathsf{Res}(t^p_{q-1}, \mathcal{W}_{q-1})(u^p_{q-1})$. Further, we have by hypothesis that t_q contains an outermost redex occurrence w_q such that $w_q \in \mathsf{Res}(t_{q-1}, \mathcal{U}_{q-1})(w_{q-1})$. By Lemma 6.2.4, there is an outermost redex occurrence w^p_q in t^p_q such that $w^p_q \in \mathsf{Res}(t^p_{q-1}, \mathcal{U}^p_{q-1})(w^p_{q-1})$ and $w_q \in \mathsf{Res}(t^p_q, \mathcal{W}_q)(w^p_q)$. This yields that the development rewrite sequence $t^p_m \overset{\mathcal{U}^p_m}{-\!\!\circ\!\!\rightarrow} t^p_{m+1} \overset{\mathcal{U}^p_{m+1}}{-\!\!\circ\!\!\rightarrow} t^p_{m+1} \overset{\mathcal{U}^p_{m+2}}{-\!\!\circ\!\!\rightarrow} \ldots$ contains an infinite outermost chain.

For $n \geq m$, we define $q(n)$ to be the smallest number such that $\mathcal{V}^{q(n)}_n \neq \emptyset$.

The redex occurrences in $\mathcal{V}^{q(n)}_n$ are all outermost. There are two ways to eliminate a redex occurrence $w^{q(n)}_n$ in $\mathcal{V}^{q(n)}_n$:

1. by contracting $w^{q(n)}_n$,

2. by contracting a redex occurrence that is overlapping with $w^{q(n)}_n$.

It cannot be the case that an outermost redex occurrence $w^{q(n)}_n$ in the set $\mathcal{V}^{q(n)}_n$ is eliminated by creating a new redex occurrence above it, since otherwise $w^{q(n)}_n$ would not have been needed for creating w_m. Hence for every $n \geq m$ there exists n' such that $\mathcal{V}^{q(n)}_{n'} = \emptyset$. We conclude that there is a q such that $\mathcal{V}^0_q = \ldots = \mathcal{V}^{p-1}_q = \emptyset$. Hence the rewrite sequence σ contains an infinite outermost chain, since eventually the rewrite sequence τ_m coincides with the rewrite sequence σ^p_m. □

Since a rewrite sequence ending in a normal form is always outermost-fair, we now have by the previous proposition that if a rewrite sequence σ is outermost-fair, then the orthogonal projection of σ over some rewrite step is outermost-fair as well. This completes the second step of the proof.

Now we come to the third step of the proof: we show that if a rewrite sequence σ is outermost-fair, and its orthogonal projection τ ends in a normal form, then σ ends in a normal form as well.

Proposition 6.2.12. *Let $\sigma : s_0 \xrightarrow{u_0} s_1 \xrightarrow{u_1} s_2 \xrightarrow{u_2} \ldots$ be an outermost-fair rewrite sequence. Let $\tau : t_0 \overset{\mathcal{U}_0}{-\!\!\circ\!\!\rightarrow} t_1 \overset{\mathcal{U}_1}{-\!\!\circ\!\!\rightarrow} t_2 \overset{\mathcal{U}_2}{-\!\!\circ\!\!\rightarrow} \ldots$ be the orthogonal projection of σ over the rewrite step $v : s_0 \rightarrow t_0$. If τ ends in a normal form then σ ends in a normal form.*

Proof. Let $\sigma : t^0_0 \xrightarrow{u_0} t^0_1 \xrightarrow{u_1} t^0_2 \xrightarrow{u_2} \ldots$ be an outermost-fair rewrite sequence. Let $\tau : t_0 \overset{\mathcal{U}_0}{-\!\!\circ\!\!\rightarrow} t_1 \overset{\mathcal{U}_1}{-\!\!\circ\!\!\rightarrow} t_2 \overset{\mathcal{U}_2}{-\!\!\circ\!\!\rightarrow} \ldots$ be the orthogonal projection of σ over the rewrite step $v : t^0_0 \rightarrow t_0$. Suppose the rewrite sequence τ ends in normal form t_m. We prove that the rewrite sequence σ ends in the normal form t_m as well. The situation is depicted as follows:

$$\begin{tikzcd}
\sigma : & s_0 \arrow[r, "u_0"] \arrow[d, "\mathcal{V}_0"'] & s_1 \arrow[r, dotted] \arrow[d, "\mathcal{V}_1"'] & {} \arrow[r, "u_{m-1}"] & s_m \arrow[r, "u_m"] \arrow[d, "\mathcal{V}_m"'] & s_{m+1} \arrow[r, "u_{m+1}"] \arrow[d, "\mathcal{V}_{m+1}"'] & {} \arrow[r, dotted] & {} \\
\tau : & t_0 \arrow[r, "\mathcal{U}_0"] & t_1 \arrow[r, dotted] & {} \arrow[r, "\mathcal{U}_{m-1}"] & t_m \arrow[r, "\emptyset"] & t_{m+1} \arrow[r, "\emptyset"] & {} \arrow[r, dotted] & {}
\end{tikzcd}$$

For n with $n \geq m$ we have $\mathcal{U}_n = \emptyset$.

Let ρ_m be the complete outermost development rewrite sequence of the set $\mathcal{V}_m$, as defined in Definition 6.2.5:

$$\rho_m : t_m^0 \xrightarrow{\mathcal{V}_m^0} t_m^1 \xrightarrow{\mathcal{V}_m^1} \ldots t_m^{p-1} \xrightarrow{\mathcal{V}_m^{p-1}} t_m^p \xrightarrow{\mathcal{W}_m} t_m.$$

Note that $\mathcal{W}_m = \emptyset$ and $t_m^p = t_m$, since t_m is a normal form. We project the rewrite sequence $\sigma_m : t_m^0 \xrightarrow{u_m} t_{m+1}^0 \xrightarrow{u_{m+1}} t_{m+2}^0 \xrightarrow{u_{m+2}} \ldots$ over the development rewrite sequence ρ_m. In a diagram:

$$\begin{tikzcd}
\sigma_m^0 & t_m^0 \arrow[r, "\mathcal{U}_m^0"] \arrow[d, "\mathcal{V}_m^0"] & t_{m+1}^0 \arrow[r, "\mathcal{U}_{m+1}^0"] \arrow[d, "\mathcal{V}_{m+1}^0"] & t_{m+2}^0 \arrow[r, "\mathcal{U}_{m+2}^0"] \arrow[d, "\mathcal{V}_{m+2}^0"] & {} \arrow[r, dotted] & {} \\
\sigma_m^1 & t_m^1 \arrow[r, "\mathcal{U}_m^1"] \arrow[d, "\mathcal{V}_m^1"] & t_{m+1}^1 \arrow[r, "\mathcal{U}_{m+1}^1"] \arrow[d, "\mathcal{V}_{m+1}^1"] & t_{m+2}^1 \arrow[r, "\mathcal{U}_{m+2}^1"] \arrow[d, "\mathcal{V}_{m+2}^1"] & {} \arrow[r, dotted] & {} \\
 & {} \arrow[d, dotted] & {} \arrow[d, dotted] & {} \arrow[d, dotted] & & \\
 & {} \arrow[d, "\mathcal{V}_m^{p-1}"] & {} \arrow[d, "\mathcal{V}_{m+1}^{p-1}"] & {} \arrow[d, "\mathcal{V}_{m+2}^{p-1}"] & & \\
\sigma_m^p & t_m^p \arrow[r, "\emptyset"'] & t_{m+1}^p \arrow[r, "\emptyset"'] & t_{m+2}^p \arrow[r, "\emptyset"'] & {} \arrow[r, dotted] & {}
\end{tikzcd}$$

with $\mathcal{U}_n^0 = \{u_n\}$ for $n \geq m$. Note that $t_n^p = t_n$ for $n \geq m$.

Let $q(n)$ be the smallest number such that $\mathcal{V}_n^{q(n)} \neq \emptyset$ for all $n \geq m$. Every redex occurrence in $\mathcal{V}_n^{q(n)}$ is an outermost redex occurrence in s_n. Since the rewrite sequence σ is outermost-fair, a redex occurrence in $\mathcal{V}_n^{q(n)}$ will eventually be eliminated. Because for all $n \geq m$ we have that $t_n^p = t_m$, which is a normal form, elimination of a redex occurrence $w_n^{q(n)}$ in $\mathcal{V}_n^{q(n)}$ can happen in only two ways:

1. by contracting $w_n^{q(n)}$,
2. by contracting a redex occurrence that is overlapping with $w_n^{q(n)}$.

It cannot be the case that the outermost redex occurrence $w_n^{q(n)}$ is eliminated by creating a redex occurrence above $w_n^{q(n)}$, since otherwise t_n^p would not be a normal form. We conclude that for every $n \geq m$ there exists an $n' \geq n$ such that $\mathcal{V}_{n'}^{q(n)} = \emptyset$. So there exists a $q \geq m$ such that $\mathcal{V}_q^0 = \ldots = \mathcal{V}_q^{p-1} = \emptyset$, and hence $t_q^0 = s_q = t_m$. Hence the rewrite sequence σ ends in the normal form t_m. □

The following theorem states the main result of this chapter.

Theorem 6.2.13. *Let $\mathcal{H}$ be a higher-order rewriting system and let s_0 be a weakly normalising term. All outermost-fair rewrite sequences starting in s_0 eventually end in a normal form.*

Proof. Let s_0 be a weakly normalising term with normal form t_0'. We fix a rewrite sequence $\rho : s_0 \twoheadrightarrow t_0'$ from s_0 to its normal form consisting of m rewrite steps. Let $\tilde{\sigma} : s_0 \xrightarrow{\tilde{u}_0} s_1 \xrightarrow{\tilde{u}_1} s_2 \xrightarrow{\tilde{u}_2} s_3 \xrightarrow{\tilde{u}_3} \ldots$ be an outermost-fair rewrite sequence starting in s_0. We prove by induction on m that $\tilde{\sigma}$ ends in the normal form of s_0.

Suppose $m = 0$. Then the term s_0 is a normal form.

Suppose $m > 0$. Then the rewrite sequence ρ is of the form $\rho : s_0 \xrightarrow{v} t_0 \twoheadrightarrow t_0'$. We construct the weakly orthogonal projection of the rewrite sequence $\tilde{\sigma}$ over the rewrite step $v : s_0 \rightarrow t_0$. Let σ be the simulation of $\tilde{\sigma}$ and let τ be the orthogonal projection of σ over the rewrite step $v : s_0 \rightarrow t_0$. In a diagram:

$$\begin{tikzcd}
\tilde{\sigma} : & s_0 \arrow[r, "\tilde{u}_0"] & s_1 \arrow[r, "\tilde{u}_1"] & s_2 \arrow[r, "\tilde{u}_2"] & s_3 \arrow[r, "\tilde{u}_3"] & \cdots \\
\sigma : & s_0 \arrow[r, "u_0"] \arrow[d, "\mathcal{V}_0"', "\circ" description] & s_1 \arrow[r, "u_1"] \arrow[d, "\mathcal{V}_1"', "\circ" description] & s_2 \arrow[r, "u_2"] \arrow[d, "\mathcal{V}_2"', "\circ" description] & s_3 \arrow[r, "u_3"] \arrow[d, "\mathcal{V}_3"', "\circ" description] & \cdots \\
\tau : & t_0 \arrow[r, "\mathcal{U}_0", "\circ" description] & t_1 \arrow[r, "\mathcal{U}_1", "\circ" description] & t_2 \arrow[r, "\mathcal{U}_2", "\circ" description] & t_3 \arrow[r, "\mathcal{U}_3", "\circ" description] & \cdots
\end{tikzcd}$$

Since by hypothesis the rewrite sequence $\tilde{\sigma}$ is outermost-fair, by Proposition 6.2.2, the rewrite sequence σ is outermost-fair. Hence by Proposition 6.2.11, the rewrite sequence τ is outermost-fair as well. By the induction hypothesis, τ eventually ends in a normal form. This yields by Proposition 6.2.12, that the rewrite sequence σ eventually ends in a normal form, which yields that the rewrite sequence $\tilde{\sigma}$ eventually ends in a normal form. Note that since $\mathcal{H}$ is confluent, both $\tilde{\sigma}$ and τ end in the term t_0'. □

Conclusions and Related Work. In [O'D77], O'Donnell introduces the notion of outermost-fair rewriting and shows that outermost-fair rewriting is normalising for almost orthogonal first-order term rewriting systems. An alternative proof of this result is given by Bergstra and Klop in the appendix of [BK86]. This proof is concerned with the class of orthogonal CRSs, so there are no critical pairs but on the other hand it concerns higher-order rewriting. The proof in [BK86] is however not entirely correct. In particular, claim 2 in the proof of Proposition 7.9 does not hold, which can be seen by considering the higher-order rewriting system with the rule for μ-recursion and a rule $z.fz \rightarrow z.z$.

I have claimed in a previous version the result presented in this chapter to hold for the class of weakly orthogonal higher-order rewriting systems, but unfortunately the proof was incorrect.

An extension to weakly orthogonal instead of almost orthogonal rewriting systems turned out to be, after all, a serious complication. To start with, it is not easy to show Proposition 6.2.2 for weakly orthogonal rewriting systems, since in that case the relation $\sharp$ is not transitive. Consider for instance the higher-order rewriting system defined by the rewrite rules

$$\begin{array}{rcl} x.f(gx) & \rightarrow_{\mathrm{R}_1} & x.f(gb) \\ ga & \rightarrow_{\mathrm{R}_2} & gb \\ a & \rightarrow_{\mathrm{R}_3} & b \end{array}$$

It is left-linear and weakly overlapping but not almost overlapping. The outermost redex occurrence $(0, \mathrm{R}_1)$ in the term $f(ga)$ is eliminated in the rewrite step $(10, \mathrm{R}_2)$: $f(ga) \rightarrow f(gb)$ because we have $(0, \mathrm{R}_1) \sharp (10, \mathrm{R}_2)$. If the redex occurrence $(11, \mathrm{R}_3)$ in $f(ga)$ is a residual of the redex occurrence that induces the rewrite step over which is projected, then the rewrite step $(10, \mathrm{R}_2) : f(ga) \rightarrow f(gb)$ will be simulated (in a weakly orthogonal projection) by the rewrite step $(11, a \rightarrow b) : f(ga) \rightarrow f(gb)$. This rewrite step does not eliminate the outermost redex occurrence $(0, \mathrm{R}_1)$.

More dramatically, the second part of Lemma 6.2.4 does not hold for weakly orthogonal rewriting systems, as remarked by Vincent van Oostrom. The reason is that in the last part of the proof we do not have, in that case, that $v_1 \parallel \mathcal{U}_2'$.

The statement of Proposition 6.2.2 seems to be true also for weakly orthogonal and fully extended higher-order rewriting systems. Moreover, in spite of the fact that Lemma 6.2.4 doesn't hold for weakly orthogonal higher-order rewriting systems, the conjecture that outermost-fair rewriting is normalising for weakly orthogonal and higher-order rewriting systems still seems plausible.

Bibliography

[Acz78] Peter Aczel. A general Church-Rosser theorem. University of Manchester, July 1978.

[Aka93] Yohji Akama. On Mints' reduction for ccc-calculus. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications (TLCA '93)*, pages 1–12, Utrecht, The Netherlands, March 1993. Volume 664 of Lecture Notes in Computer Science.

[AL94] Andrea Asperti and Cosimo Laneve. Interaction systems I: The theory of optimal reductions. *Mathematical Structures in Computer Science*, 4:457–504, 1994.

[AL96] Andrea Asperti and Cosimo Laneve. Interaction systems II: The practice of optimal reductions. *Theoretical Computer Science*, July 1996. To appear.

[Asp94] Andrea Asperti. Linear logic, comonads and optimal reductions. *Fundamenta Informaticae*, 22:3–22, 1994. Special Issue devoted to Categories in Computer Science.

[Bar71] H.P. Barendregt. *Some Extensional Term Models for Combinatory Logics and λ-calculi.* PhD thesis, Rijksuniversiteit Utrecht, 1971.

[Bar74] H.P. Barendregt. Pairing without conventional constraints. *Zeitschrift für mathematischen Logik und Grundlagen der Mathematik*, 20:289–306, 1974.

[Bar84] H.P. Barendregt. *The Lambda Calculus, its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Company, revised edition, 1984.

[Bar92] H.P. Barendregt. Lambda calculi with types. In S. Abramsky, Dov M. Gabbay, and T.S.E Maibaum, editors, *Handbook of Logic in Computer*

Science, volume 2, pages 117–310. Oxford University Press, New York, 1992.

[BBKV76] Henk Barendregt, Jan Bergstra, Jan Willem Klop, and Henri Volken. Degrees, reductions and representability in the lambda calculus. Technical Report 22, University Utrecht, February 1976.

[BK86] J.A. Bergstra and J.W. Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32:323–362, 1986.

[BTG90] Val Breazu-Tannen and Jean Gallier. Polymorphic rewriting conserves algebraic strong normalization. *Theoretical Computer Science*, 83:3–28, 1990.

[BTG94] Val Breazu-Tannen and Jean Gallier. Polymorphic rewriting conserves algebraic confluence. *Information and Computation*, 14(1):1–29, 1994.

[CFC58] Haskell B. Curry, Robert Feys, and William Craig. *Combinatory Logic, volume I.* Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Company, Amsterdam, 1958.

[CH88] Th. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[Chu41] A. Church. *The Calculi of Lambda Conversion.* Princeton University Press, 1941.

[CR36] Alonzo Church and J.B. Rosser. Some properties of conversion. *Transactions of the Americal Mathematical Society*, 39:472–482, 1936.

[Čub93] Djordje Čubrič. Embedding of a free cartesian closed category into the category of sets. McGill University, 1993.

[Cur95] Pierre-Louis Curien. Algèbre universelle, introduction au λ-calcul et aux logiques combinatoires (notes de cours). Technical Report LIENS-95-30, Ecole Normale Supériere, November 1995.

[Dan90] Vincent Danos. *La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du λ-calcul).* PhD thesis, Université Paris VII, 1990.

[DCK94] Roberto Di Cosmo and Delia Kesner. Simulating expansions without expansions. *Mathematical Structures in Computer Science*, 4:315–362, 1994.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 6, pages 243–320. Elsevier Science Publishers B.V., Amsterdam, 1990.

[DR89] Vincent Danos and Laurent Régnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[GAL92] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optimal lambda reduction. In *Proceedings of the 19th ACM Conference on Principles of Programming Languages (POPL '92)*, pages 15–26, 1992.

[Gan80] R.O. Gandy. Proofs of strong normalization. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 457–477. Academic Press, 1980.

[Ghi94] Sylvia Ghilezan. Application of typed lambda calculi in the untyped lambda calculus. In A. Nerode and Yu.V. Matiyasevich, editors, *Proceedings of the Third International Symposium on Logical Foundations of Computer Science (LFCS '94)*, volume 813 of *Lecture Notes in Computer Science*, pages 129–139, St. Petersburg, Russia, July 1994.

[Gir72] Jean-Yves Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[GK94] J.R.W. Glauert and Z. Khasidashvili. Relative normalization in orthogonal expression reduction systems. In N. Dershowitz, editor, *Proceedings of the 4th International Workshop on Conditional and Typed Term Rewriting Systems (CTRS '94)*, volume 968 of *Lecture Notes in Computer Science*, pages 144–165, Jerusalem, Israel, 1994.

[GK96a] J.R.W. Glauert and Z. Khasidashvili. Minimal and optimal relative normalization in orthogonal expression reduction systems. University of East Anglia, 1996.

[GK96b] J.R.W. Glauert and Z. Khasidashvili. Relative normalization in deterministic residual structures. In *Proceedings of the 19th International Colloquium on Trees in Algebras and Programming (CAAP '96)*, April 1996. To appear.

[GLM92] Georges Gonthier, Jean-Jacques Lévy, and Paul-André Melliès. An abstract standardisation theorem. In *Proceedings of the seventh Annual IEEE Symposium on Logic in Computer Science (LICS '92)*, pages 72–81, Los Alamitos, California, 1992.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[Gro93] Philippe de Groote. The conservation theorem revisited. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications (TLCA '93)*, pages 163–178, Utrecht, The Netherlands, March 1993. Volume 664 of Lecture Notes in Computer Science.

[Hin78] R. Hindley. Reductions of residuals are finite. *Transactions of the Americal Mathematical Society*, 240:345–361, June 1978.

[HL91] Gérard Huet and Jean-Jacques Lévy. Computations in orthogonal rewriting systems, I and II. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in honor of Alan Robinson*, pages 395–443. MIT Press, Cambridge, Massachusetts, 1991.

[HO80] Gérard Huet and Derek C. Oppen. Equations and rewrite rules, a survey. In R.V. Book, editor, *Formal Language Theory, Perspectives and Open Problems*, pages 349–405. Academic Press, 1980.

[HS86] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-Calculus*, volume 1 of *London Mathematical Society Student Texts*. Cambridge University Press, 1986.

[Hue80] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the Association for Computing Machinery*, 27(4):797–821, October 1980.

[Hue94] Gérard Huet. Residual theory in λ-calculus: a formal development. *Journal of Functional Programming*, 4(3):371–394, 1994.

[Hyl73] J.M.E. Hyland. A simple proof of the Church-Rosser theorem. Technical report, Oxford University, 1973.

[JO91] Jean-Pierre Jouannaud and Mitsuhiro Okada. A computational model for executable higher-order algebraic specification languages. In *Proceedings of the sixth annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 350–361, Amsterdam, July 1991.

[Joi93] J.-B. Joinet. *Etude de la normalisation du calcul des séquents classique à travers la logique linéaire.* PhD thesis, Université Paris VII, 1993.

[Kah94] Stefan Kahrs. Compilation of combinatory reduction systems. In J. Heering, K. Meinke, B. Möller, and T. Nipkow, editors, *Proceedings of the First International Workshop on Higher-Order Algebra, Logic and Term Rewriting (HOA '93)*, volume 816 of *Lecture Notes in Computer Science*, pages 169–188, Amsterdam, The Netherlands, 1994.

[Kah95] Stefan Kahrs. Towards a domain theory for termination proofs. In Jieh Hsiang, editor, *Proceedings of the 6th International Conference on Rewriting Techniques and Applications (RTA '95)*, volume 941 of *Lecture Notes in Computer Science*, pages 241–255, Kaiserslautern, Germany, April 1995.

[Ken92] J.R. Kennaway. On transfinite abstract reduction systems. Technical Report CS-R9205, CWI, January 1992.

[Kha90] Z.O. Khasidashvili. Expression Reduction Systems. In *Proceedings of I. Vekua Institute of Applied Mathematics*, volume 36, pages 200–220, Tblisi, 1990.

[Kha92] Z.O. Khasidashvili. The Church-Rosser theorem in orthogonal combinatory reduction systems. Technical Report 1825, INRIA Rocqencourt, France, 1992.

[Kha94] Zurab Khasidashvili. The longest perpetual reductions in orthogonal expression reduction systems. In A. Nerode and Yu.V. Matiyasevich, editors, *Proceedings of the Third International Symposium on Logical Foundations of Computer Science (LFCS '94)*, number 813 in Lecture Notes in Computer Science, pages 191–203, St. Petersburg, Russia, July 1994.

[Klo80] J.W. Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. CWI, Amsterdam, 1980. PhD Thesis.

[Klo92] J.W. Klop. Term rewriting systems. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, New York, 1992.

[KO95] Z. Khasidashvili and V. van Oostrom. Context-sensitive conditional expression reduction systems. *Elsevier, Electronic Notes in Theoretical Computer Science*, 2, 1995. Available at http://www.elsevier/nl/mcs/tcs/pc/volume2.htm.

[KOR93] J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993. Special issue in honour of Corrado Böhm.

[Kri93] J.L. Krivine. *Lambda-Calculus, Types and Models.* Masson, Paris, France, 1993.

[Laf90] Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL '90)*, pages 95–108, San Francisco, USA, January 1990.

[Lam90] John Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of the 17th ACM Conference on Principles of Programming Languages (POPL '90)*, pages 16–30, San Francisco, USA, January 1990.

[Lan93] Cosimo Laneve. *Optimality and Concurrency in Interaction Systems.* PhD thesis, Università di Pisa, Pisa, Italy, March 1993.

[Lév78] Jean-Jacques Lévy. *Réductions correctes et optimales dans le lambda-calcul.* PhD thesis, Université de Paris VII, 1978.

[Loa95] R. Loader. Normalisation by translation. Note distributed on the 'types' mailing list, April 1995.

[LP95] Olav Lysne and Javier Piris. A termination ordering for higher-order rewrite systems. In Jieh Hsiang, editor, *Proceedings of the 6th International Conference on Rewriting Techniques and Applications (RTA '95)*, volume 914 of *Lecture Notes in Computer Science*, pages 26–40, Kaiserslautern, Germany, April 1995.

[Mel96] Paul-André Melliès. *Description Abstraite des Systèmes de Réécriture.* PhD thesis, Université de Paris VII, 1996. To appear.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[MN94] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. Technical Report TUM-I9433, Technische Universität München, August 1994.

[Ned73] R.P. Nederpelt. *Strong Normalization in a Typed Lambda Calculus with Lambda Structured Types.* PhD thesis, Technische Hogeschool Eindhoven, June 1973.

[New42] M.H.A Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, April 1942.

[NGV94] R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science B.V., 1994.

[Nip91] Tobias Nipkow. Higher-order critical pairs. In *Proceedings of the sixth annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 342–349, Amsterdam, The Netherlands, July 1991.

[Nip93a] Tobias Nipkow. Functional unification of higher-order patterns. In *Proceedings of the eigth annual IEEE Symposium on Logic in Computer Science (LICS '93)*, pages 64–74, Montreal, Canada, June 1993.

[Nip93b] Tobias Nipkow. Orthogonal higher-order rewrite systems are confluent. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications (TLCA '93)*, volume 664 of *Lecture Notes in Computer Science*, pages 306–317, Utrecht, The Netherlands, March 1993.

[Nip96] Tobias Nipkow. More Church-Rosser proofs (in Isabelle/HOL). In *Proceedings of the 13th International Conference on Automated Deduction (CADE '96)*, 1996. To appear.

[O'D77] Michael J. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *Lecture Notes in Computer Science*. Springer Verlag, 1977.

[Oos94] Vincent van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit, Amsterdam, March 1994.

[Oos96] Vincent van Oostrom. Development closed critical pairs. In G. Dowek, J. Heering, K. Meinke, and B. Möller, editors, *Proceedings of the Second International Workshop on Higher-Order Algebra, Logic and Term Rewriting (HOA '95)*, volume 1074 of *Lecture Notes in Computer Science*, pages 185–200, Paderborn, Germany, 1996.

[OR93] Vincent van Oostrom and Femke van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. Technical Report CS-R9361, CWI, September 1993.

[OR94a] Vincent van Oostrom and Femke van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. In J. Heering, K. Meinke, B. Möller, and T. Nipkow, editors, *Proceedings of* *the First International Workshop on Higher-Order Algebra, Logic and Term Rewriting (HOA '93)*, volume 816 of *Lecture Notes in Computer Science*, Amsterdam, 1994.

[OR94b] Vincent van Oostrom and Femke van Raamsdonk. Weak orthogonality implies confluence: the higher-order case. In A. Nerode and Yu.V. Matiyasevich, editors, *Proceedings of the Third International Symposium on Logical Foundations of Computer Science (LFCS '94)*, volume 813 of *Lecture Notes in Computer Science*, pages 379–392, St. Petersburg, July 1994.

[Par90] Michel Parigot. Internal labellings in lambda-calculus. In B. Rovan, editor, *Proceedings of the 15th Symposium on Mathematical Foundations of Computer Science (MFCS '90)*, volume 452 of *Lecture Notes in Computer Science*, pages 439–445, Banská Bystrica, Czechoslovakia, August 1990.

[Pkh77] Sh. Pkhakadze. Some problems of the notation theory. Technical report, I. Vekua Institute of Applied Mathematics, Tblisi, Georgia, 1977. In Russian.

[Plo77] G.D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[Plo81] Gordon D. Plotkin. A structured approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, September 1981.

[Pol94] J.C. van de Pol. Termination proofs for higher-order rewrite systems. In J. Heering, K. Meinke, B. Möller, and T. Nipkow, editors, *Proceedings of the First International Workshop on Higher-Order Algebra, Logic and Term Rewriting (HOA '93)*, volume 816 of *Lecture Notes in Computer Science*, pages 305–325, Amsterdam, The Netherlands, 1994.

[Pol96] Jaco van de Pol. Two *different* strong normalization proofs? In G. Dowek, J. Heering, K. Meinke, and B. Möller, editors, *Proceedings of the Second International Workshop on Higher-Order Algebra, Logic and Term Rewriting (HOA '95)*, volume 1074 of *Lecture Notes in Computer Science*, pages 201–220, Paderborn, Germany, 1996.

[Pre95] Christian Prehofer. *Solving Higher-Order Equations: From Logic to Programming*. PhD thesis, Technische Universität München, 1995.

[PS95] Jaco van de Pol and Helmut Schwichtenberg. Strict functionals for termination proofs. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications (TLCA '95)*, number 902 in Lecture Notes in Computer Science, pages 350–364, Edinburgh, 1995.

[Raa92] Femke van Raamsdonk. Representations of λ-terms as proof nets. Master's Thesis, 1992.

[Raa93] Femke van Raamsdonk. Confluence and superdevelopments. In Claude Kirchner, editor, *Proceedings of the 5th International Conference on Rewriting Techniques and Applications (RTA '93)*, pages 168–182, Montreal, Canada, June 1993. Volume 690 of Lecture Notes in Computer Science.

[Rég92] Laurent Régnier. *Lambda-calcul et réseaux.* PhD thesis, Université Paris VII, 1992.

[Rey74] John C. Reynolds. Towards a theory of type structure. In *Proceedings of the 'Colloque sur la Programmation'*, number 19 in Lecture Notes in Computer Science, pages 408–425, Paris, France, 1974.

[Ros73] Barry K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the Association for Computing Machinery*, 20(1):160–187, January 1973.

[Ros82] J. Barkley Rosser. Highlights of the history of the lambda-calculus. In *Proceedings of the ACM symposium on LISP and Functional Programming*, pages 216–225, New York, 1982.

[Ros96] Kristoffer H. Rose. *Operational Reduction Models for Functional Programming Languages.* PhD thesis, University of Copenhagen, 1996.

[RS95] F. van Raamsdonk and P. Severi. On normalisation. Technical Report CS-R9545, CWI, June 1995.

[Sch24] M. Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924.

[Sch65] D.E. Schroer. *The Church-Rosser Theorem.* PhD thesis, Cornell University, 1965.

[Ste72] S. Stenlund. *Combinators, λ-terms, and Proof Theory.* Reidel, 1972.

[Tai67] W.W. Tait. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32:198–212, 1967.

[Tak93] Masako Takahashi. λ-calculi with conditional rules. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications (TLCA '93)*, volume 664 of *Lecture Notes in Computer Science*, pages 406–417, Utrecht, The Netherlands, March 1993.

[Tak95] Masako Takahashi. Parallel reductions in λ–calculus. *Information and Computation*, 118:120–127, 1995.

[Vri85] Roel de Vrijer. A direct proof of the finite developments theorem. *Journal of Symbolic Logic*, 50(2):339–343, June 1985.

[Vri87a] Roel de Vrijer. Exactly estimating functionals and strong normalization. *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen*, 90(4), December 1987.

[Vri87b] Roel de Vrijer. *Surjective Pairing and Strong Normalization: Two Themes in Lambda Calculus.* PhD thesis, Universiteit van Amsterdam, January 1987.

[Wad71] C.P. Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus.* PhD thesis, Oxford University, 1971.

[Wol91] D.A. Wolfram. Rewriting, and equational unification: the higher-order cases. In *Proceedings of the 4th International Conference on Rewriting Techniques and Applications (RTA '91)*, 1991.

[Wol93] D.A. Wolfram. *The Clausal Theory of Types*, volume 21 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1993.

Index

Samenvatting

Een berekening zoals die bijvoorbeeld door een computer uitgevoerd wordt, bestaat uit het stapsgewijs veranderen van een expressie. Als voorbeeld bekijken we de volgende berekening:

$$(1+2) \times 3 = 3 \times 3 = 9.$$

Hier wordt de expressie $(1 + 2) \times 3$ in twee stapjes veranderd in de expressie 9. De berekening bevat een bepaalde volgorde, die we expliciet maken door haar te schrijven als

$$(1+2) \times 3 \to 3 \times 3 \to 9.$$

Dit voorbeeld illustreert een belangrijk punt, namelijk het verschil tussen *betekenis* enerzijds en *representatie* anderzijds. De betekenis van de drie expressies in de bovenstaande berekening is steeds dezelfde. Echter, de representatie verandert van het toch tamelijk ingewikkelde $(1 + 2) \times 3$ via het eenvoudiger 3×3 in het elementaire 9. Vaak zullen we geïnteresseerd zijn in een transformatie waarin de representatie niet zomaar verandert, maar vereenvoudigd wordt.

Het onderwerp van dit proefschrift is de theorie van dergelijke transformaties. Het veranderen van een expressie noemen we *herschrijven*, en een opeenvolging van herschrijfstapjes een *herschrijfrij*. Uiteindelijk dient het uitvoeren van een berekening een resultaat of uitkomst op te leveren. Dat wil zeggen dat we een expressie willen herschrijven tot ze een vorm heeft aangenomen die om de een of andere reden het meest geschikt is. Zo is het goed voorstelbaar dat we 10^{10} als resultaat verkiezen boven 10000000000. Bij rekenen met natuurlijke getallen als in het voorbeeld gaan we ervan uit dat een berekening uiteindelijk in een natuurlijk getal dient te eindigen. In de bovenstaande berekening is dit heel goed gelukt. Echter, het kan ook anders:

$$(1+2) \times 3 \to (2+1) \times 3 \to (1+2) \times 3 \to \ldots.$$

Deze herschrijfrij eindigt niet in een resultaat.

Eén van de vragen die in de theorie van herschrijven bestudeerd wordt is dan ook: eindigt een herschrijfrij na eindig veel stapjes in een resultaat? Een expressie die niet het begin is van een oneindige herschrijfrij heet *sterk normaliserend*. Een expressie die tot een resultaat herschreven kan worden wordt *zwak normaliserend* genoemd. Een voorbeeld van een expressie die wel zwak maar niet sterk normaliserend is, is de expressie $(1+2)\times 3$ in het voorbeeld. Het moge duidelijk zijn dat het voor zwak normaliserende expressies van belang is dat we een methode tot onze beschikking hebben die voorschrijft hoe de expressie herschreven dient te worden, opdat de herschrijfrij op den duur in een resultaat eindigt. Tenslotte zijn er ook expressies die niet zwak en dus zeker niet sterk normaliserend zijn, namelijk de expressies die op geen enkele manier tot een resultaat herschreven kunnen worden. In rekenen met natuurlijke getallen komen dergelijke expressies niet voor, maar bijvoorbeeld in een herschrijfsysteem met $a \rightarrow a$ wel.

Een andere belangrijke vraag in de theorie van herschrijven is of de uitkomst van een berekening, zo die er is, afhangt van de manier waarop de berekening is uitgevoerd. Een herschrijfsysteem waarin het resultaat niet afhangt van de manier van herschrijven heet *confluent*. Rekenen met natuurlijke getallen vormt een confluent herschrijfsysteem. Hoe we de expressie $(1 + 2) \times 3$ ook herschrijven, er is altijd een manier om de herschrijfrij zo af te maken dat hij in 9 eindigt. Niet elk herschrijfsysteem is echter confluent. Een herschrijfsysteem dat bijvoorbeeld alleen de herschrijfstapjes $a \rightarrow b$ en $a \rightarrow c$ bevat, is niet confluent: de expressie a kan tot twee verschillende resultaten, namelijk b en c, herschreven worden.

In dit proefschrift wordt een zeer uitgebreide klasse van herschrijfsystemen bestudeerd, bestaande uit de zogenaamde hogere-orde herschrijfsystemen. Deze klasse bevat bijvoorbeeld λ-calculus en alle eerste-orde termherschrijfsystemen. We laten zien dat hogere-orde herschrijfsystemen met een bepaalde eigenschap, zwakke orthogonaliteit geheten, confluent zijn. *Orthogonaliteit* en ook de zwakkere versie *zwakke orthogonaliteit* geven aan op welke manier verschillende herschrijfstapjes kunnen interfereren. Voorts wordt aangetoond dat er een manier is om een expressie in een bijna orthogonaal hogere-orde herschrijfsysteem te herschrijven, zodat op den duur zo mogelijk een resultaat verkregen wordt, namelijk door bepaalde herschrijfstapjes niet oneindig vaak niet te doen. Preciezer, we laten zien dat voor bijna orthogonale hogere-orde herschrijfsystemen het zogenaamde outermost-fair herschrijven altijd tot een resultaat leidt, zo dit er is.